Re-Crafting Games:

The Inner Life of Minecraft Modding

Nicholas Watson

A Thesis

In the Department

of

Communication Studies

Presented in Partial Fulfillment of the Requirements

For the Degree of

Doctor of Philosophy (Communication) at

Concordia University

Montréal, Québec, Canada

May 2019

CONCORDIA UNIVERSITY
SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared

By: Nicholas Watson

Entitled: Re-Crafting Games: The inner life of Minecraft modding

and submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY (Communication)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

Dr. Martin Lefebvre	Chair
Dr. David Nieborg	External Examiner
Dr. Darren Wershler	External to Program
Dr. Charles Acland	Examiner
Dr. Owen Chapman	Examiner
Dr. Mia Consalvo	Thesis Supervisor

Approved by _____
Dr. Jeremy Stolow, Graduate Program Director

June 20, 2019 _____
Dr. André Roy
Dean, Faculty of Arts and Sciences

ABSTRACT

Re-Crafting Games: The inner life of Minecraft Modding

Nicholas Watson, Ph.D.

Concordia University, 2019

Prior scholarship on game modding has tended to focus on the relationship between commercial developers and modders, while the preponderance of existing work on the open-world sandbox game *Minecraft* has tended to focus on children's play or the program's utility as an educational platform. Based on participant observation, interviews with modders, discourse analysis, and the techniques of software studies, this research uncovers the inner life of Minecraft modding practices, and how they have become central to the way the game is articulated as a cultural artifact. While the creative activities of audiences have previously been described in terms of de Certeau's concept of "tactics," this paper argues that modders are also engaged in the development of new strategies. Modders thus become "settlers," forging a new identity for the game property as they expand the possibilities for play. Emerging modder strategies link to the ways that the underlying game software structures computation, and are closely tied to notions of modularity, interoperability, and programming "best practices." Modders also mobilize tactics and strategies in the discursive contestation and co-regulation of gameplay meanings and programming practices, which become more central to an understanding of game modding than the developer-modder relationship. This discourse, which structures the circulation of gaming capital within the community, embodies both monologic and dialogic modes, with websites, forum posts, chatroom conversations, and even software artifacts themselves taking on persuasive inflections.

ACKNOWLEDGMENTS

This work is dedicated to Seanna and Steve Watson, for their unwavering support over the years that it took to complete.

I wish to extend my heartfelt thanks to my thesis supervisor, Dr. Mia Consalvo, and to Drs. Bart Simon and Darren Wershler, for advice, guidance, and support; and also to the rest of my examination committee, including Drs. Charles Acland, Owen Chapman, and David Nieborg; and to Drs. Maude Bonenfant, Jeremy Stolow, and William Buxton, who offered feedback on early portions of this project.

Thank you also to advisers and mentors of my undergraduate and Master's programs, who helped to set me on the path that led here: Drs. John Haslem, Larry Breitborde, Celia Pearce, and Amy Bruckman.

I am especially grateful to all those in the Minecraft modding community who made this work possible. Some of these people were interviewees and participants in my ethnographic research; others did not participate directly, but provided advice, technical assistance, and guidance on what I should explore and whom I should talk to. In alphabetical order: Alta, asie, capitalthree, card1null, Cervator, copygirl, Icoso, JAvery, LShen, Mezz, modmuss50, Nikky, Omira, Qwil, SarahK, ScriveShark, Tothor, Velo, VicNightfall, XCompWiz, Zoll, and several anonymous helpers and participants.

Finally, thank you to Jason Wakeland, Laura Lewellen, Trish Audette, Ryan Scheiding, Victoria Puusa, and Tugger the Siamese cat, for moral support.

Funding for portions of this research and related projects was provided by the mLab at Concordia University; the Technoculture, Art, & Games research centre (Concordia University); the Milieux Institute for Arts, Culture, and Technology (Concordia University); and a SSHRC-D award grant from the Social Sciences and Humanities Research Council of Canada.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

1.1 Mod, break, win!

I won *Minecraft* while sitting at my dining-room table in August 2014.

Those familiar with the open-ended nature of Mojang AB's open-ended, block-building sandbox game may consider this an absurd statement, and they may have a point. How can you "win" a game that, by design, lacks end-goals and victory conditions? Some might say that Minecraft's win condition constitutes travelling to a realm named The End, defeating the fearsome Ender Dragon, and sitting through the 10-minute text scroll of the "End Poem"—a sort of philosophical reflection that is beloved by some, derided by others (Chatfield, 2012). But the truth about The End quest, which was added during the game's transition from Beta to the Official (and supposedly "Finished") Released in 2011, is that it ended nothing. For the player, it is little more than a detour: unlike the villains of traditional epics, the dragon's presence is not felt in gameplay prior to its defeat, so its subsequent absence changes little, and the day-to-day dramas of Minecraft life unfold much as they had before. Little changed for the game's developers too, despite The End having been the exclamation mark on the game's official release announcement. Markus "Notch" Persson, the original creator, passed the torch to other developers, who have continued to radically transform the game with incremental updates and new features in the years since.

In any case, that's not what I mean by winning.

I won Minecraft by finding a sneaky solution to the ever-present tension of its Survival Mode gameplay: the problem of resource scarcity. True, a procedurally-generated Minecraft world technically has an inexhaustible supply of food, energy, building materials, and precious minerals, but then so does our real universe—most of it is just very hard to get to. Time, labour, and considerable risk to life and limb is involved in extracting the infinite-yet-sparsely-distributed resource wealth of a Minecraft world. Diamonds, the most coveted material, are found only in the deepest layers of the Earth, scattered between treacherous lava-pools and deadly monsters. Over

time, one's Minecraft gameplay evolves from mere survival and subsistence to enrichment, expansion of wealth, and expansion of the means to acquire yet more wealth; but scarcity and friction are always present in some form. Popular technology and magic *mods*—themed, fan-made changes and additions to the game—preserve this dynamic, even as they add modes of automation and optimization that further expand the player's ability to accrue valuable resources: the new machines and arcane devices require increasing amounts of rare material to function. Even the most successful Minecraft industrialist/wizard still considers diamond a rare and precious commodity.

But not me. I found a way to make as many diamonds as I could ever want, and then some. I found a way to transmute common trash into treasure. I discovered the Minecraft equivalent of the Philosopher's Stone.

I am certainly not the first person to have done something like this, but I was the first on this particular server—operated by a colleague and populated by students and professors at our institution—and with this particular curated set of mods and game rules, or *modpack.* While experimenting with the design of automated machinery, I discovered a curious interaction between a well-known technology mod and an equally popular magic mod that were probably never intended to be used together. By letting a robotic arm wield a magic wand, I could build a factory that perpetually transformed blocks of cobblestone (one of the few truly infinitely-generable resources) into diamonds, gold, or anything else I wanted.[1]

Of course I was cheating, but in a manner that I had arrived at through play, one that respected the non-negotiable rules of code—what games scholars Salen and Zimmerman (2003) would call the "constituative rules" (*sic.*)—if not the design (operational rules) and spirit (implicit rules) of the game. As an assistant to the server operator, I could also have used administrator commands to give myself diamonds at any time, but to do so would have been to step outside the

[1] The wand is normally used to rapidly exchange large numbers of blocks in a player's inventory (e.g. diamond blocks) with blocks on the ground (e.g. cobblestone). It is supposed to remove said blocks from the player's inventory when they are "spent," but because of the way the two different mods were programmed, this failed to happen when the robotic arm was holding the wand.

player role; as an option that had been available all along, rather than one cleverly derived from play itself, this would hardly make for an interesting form of "winning."

There was a player on our server—let's call him 'HT'—who had achieved a sort of demi-god status. Through many hours of gameplay, he had pursued every mod's tech tree to its conclusion, built every kind of machine, crafted every kind of equipment, defeated every kind of monster, and collected some of every kind of material. He had one thing I lacked: amethyst, available only in a hard-to-reach dimension which HT alone had so far managed to visit. My alchemical factory could mass-produce almost anything, but only if it started with a small sample of the target material, so I could not make my own amethyst without first getting my hands on at least one block of the stuff.

So I offered HT a trade that, to him, must have seemed generous and too good to refuse, but in which my sacrifice was trivial: 81 diamonds (nine "blocks") for 9 shards of amethyst (one block).[2] The diamonds I spent meant nothing to me, but the trade revealed that even to a demi-god who had exploited every mod to its fullest, 81 diamonds was still a small fortune. With my exploit, I had completely subverted the notion of a fair trade.

But this is not really a story about me and my alchemical trickery. What I want to highlight with this anecdote is that these unusual transformations of Minecraft's play dynamics were enabled by the vibrant—but at times chaotic—modding practice and community that has coalesced around Minecraft. Almost since the game first appeared as a playable work-in-progress in 2009, players have been hacking and making changes to the code, tweaking rules and adding features. Nearly 10 years on, Minecraft is a multi-billion-dollar property with hundreds of millions of users. All throughout, the work of fan modders has been central to Minecraft's evolving articulation as a product and cultural phenomenon.

This is their story.

[2] My actions were not entirely beyond the spirit of play on our particular server. The play environment was highly experimental and not competitive in nature. I also made a disclosure to the server administrator about my discovery and subsequent activities. After the trade was concluded, I let HT in on the secret, as it felt cynical and opportunistic not to.

1.2 Background on Minecraft

Figure 1-1. Minecraft gameplay screenshots. **Left:** the player explores the surface of a world made of cuboids. The player's various mining tools and building blocks are shown in the toolbar at the bottom. **Right:** players can explore vast underground cave systems in order to find resources to mine, but have to watch out for monsters that spawn in the darkness, like this explosive green Creeper. (Screenshots by N. Watson.)

As of the mid-2019, *Minecraft* (Mojang AB, 2011) is the second-best-selling video game of all time, with over 176 million copies sold across all platforms (Dent, 2019). It has even been adapted for practical use by educators, national governments, and United Nations agencies. It is a single- or multi-player "sandbox game" played in an open-ended virtual space (a "WORLD" or "MAP")[3] where players explore (Figure 1-1, above), craft tools (Figure 1-2, below), harvest resources, fight monsters, hunt for treasures, erect structures out of cubic blocks, create art, and construct elaborate machinery. Broadly speaking, there are two ways to play: in SURVIVAL MODE, players must harvest resources from the monster-ridden environment and use them to make food, shelter, and equipment, perhaps eventually reaching the point of constructing massive palaces; in CREATIVE MODE, all objects and materials are available in unlimited quantity to invulnerable players, so that they can build freely as if with a virtual Lego set.

Minecraft's rise to popularity occurred while the product was still under iterative development, so that millions of early adopters played an incomplete game, in which major features were promised but not yet implemented. In fact, although the official release dates to November 2011, the first playable prototypes were made publicly available in 2009, and quickly became hugely

[3] These and other terms that appear in SMALL CAPITAL LETTERS are defined in the Glossary.

popular, with over a million sales almost a year before official release (Orland, 2011). This means that while Minecraft still had the form of an unfinished, evolving niche/indie game, it had the uptake of a mainstream blockbuster.

The presence of significant bugs and the addition of new content every few months meant that game play was constantly in flux. Even while the game's developers have gradually, incrementally shaped the game towards a finished form—a process which has continued long after the game was declared "finished" in 2011—independent coders and artists in the player community have also produced hundreds of unofficial bug-fixes, hacks, mods, texture packs, plug-ins, and companion programs of their own. This player-made content, which far outweighs the official content in terms of sheer quantity, has come to play a key role in defining Minecraft's identity as both a commercial product and a cultural object. Far from remaining on the periphery or in the realm of a minority of hardcore fans, the creation and use of player-made mods occupies a central position in this play ecosystem.

Figure 1-2. The crafting interface. This screenshot shows a user interface that allows common materials to be crafted into more advanced tools. Here, seven wooden sticks are being used to build three pieces of ladder. The darkened background shows part of a player-built home, which acts as a safe place to store materials and craft items. (Screenshot by N. Watson.)

1.3 Variants and lineages of Minecraft

Duncan (2011) speaks of there being "many Minecrafts" due to the game's iterative open alpha and beta development, which gave rise to different play practices in each version. This is further discussed in Chapter 4. However, in the years since Duncan's observation, officially differentiable EDITIONS (not to be confused with VERSIONS and GAME MODES) of Minecraft have also proliferated. Microsoft's purchase of Mojang AB in 2014 further reshaped the landscape of Minecraft editions. I offer the following descriptions of these editions to provide clarity for subsequent discussions. Figure 1-3 (page 9-10) shows the timeline of three major "lineages" of Minecraft—based on implementation and platform availability—and key modding tools for *Java Edition*, the Minecraft variant under investigation here.

1.3.1 Java Edition

The original edition of Minecraft, released by the author in its earliest form in 2009 and officially published as complete in late 2011, was coded using the Java programming language. Although it is sometimes referred to as the "PC version," that label is misleading since, in the case of games, this usually means that the software is designed for Microsoft Windows. *Minecraft: Java Edition* runs on Windows, Mac OS/X, and Linux, thanks to Java's platform-independent nature.

While this variant was originally simply "Minecraft," Microsoft rebranded it to "Java Edition" in 2017, concurrently with the rollout of their new Bedrock-based versions (see below). Java Edition continues to be the easiest variant to mod and thus the target of most modding. Furthermore, its modding practices have a long history and arose in an almost entirely organic and fan-driven manner, with little to no prescriptive guidance from Mojang.

The focus of this research is Java Edition modding. When the word "Minecraft" is used herein without other qualification, it will refer either to the general family of Minecraft games (i.e. all variants), or specifically to Java Edition, but not to Bedrock editions. (Readers who are well-

acquainted with the Minecraft franchise may note that this usage differs from Microsoft's current preferred branding, in which the unqualified name "Minecraft" means Bedrock.)

1.3.2 Pocket and Bedrock editions

Minecraft: Pocket Edition (MCPE) for mobile devices first appeared in late 2011 on Android and iOS, just as Java Edition (which was the primary product at the time) was gearing up for its official 1.0 release. This mobile edition was built on a codebase called the "Bedrock codebase" or "Bedrock engine," written in the C++ programming language, and was developed concurrently with Java Edition. The Windows 10 Edition, first released mid-2015, was derived from MCPE and foreshadowed Microsoft's plans to expand the Pocket Edition family to non-mobile platforms. In September 2017, they did just that with the "Better Together" update, which allowed cross-platform multiplayer gaming between mobile devices, Windows 10 PCs, Xbox One, and Nintendo Switch. The game on all four of these platform families was rebranded as "Minecraft" with no subtitle (with the original game being renamed Minecraft: Java Edition at that time). Since the Xbox One and Nintendo Switch had previously had their own Minecraft versions not related to Bedrock/MCPE, the "Better Together" update also supplanted those. The *Minecraft Wiki* (https://minecraft.gamepedia.com) and much of the player community often refer to all of the Bedrock-based versions collectively as "Bedrock Edition," although the administrators of *MinecraftForum.net* (an affiliate site of the *Wiki*) advocate for using Microsoft's official nomenclature:

> There is no such thing as "Bedrock Edition" so we will not be using that name. Bedrock is the name of the underlying platform used by the cross-platform Minecraft game. The game built on the Bedrock platform is called **Minecraft** and that is the name we are using. (citricsquid, 2017)

Because C++ programs are compiled into native code for each target platform, they can be faster and more efficient than Java programs at performing similar tasks; Java is sometimes bogged down by the VIRTUAL MACHINE's increased overhead. Since processor lag is a perennial problem for modded Minecraft—especially in multiplayer—the idea of an efficient C++ implementation was

celebrated by some server owners and modpack creators. However, C++ programs resist the kind of wide-open hacking that has made Minecraft modding so pervasive and vibrant, because the compiled binary code is much more difficult to reverse-engineer than Java BYTECODE. Furthermore, without the help of a comprehensive modding API,[4] modders would have to compile and release their mods for each target platform.

MCPE/Bedrock's affinity with mobile and touch-screen devices also gives rise to play styles that are markedly different from those seen in *Java Edition* (see Subsection 2.5.3).

1.3.3 Legacy console editions

A third family of Minecrafts was developed in C++ (but separately from Bedrock) from 2012 onward by 4J Studios for gaming consoles—Xbox 360, Xbox One, PlayStation 3, PlayStation 4, PlayStation Vita, Wii U, and Nintendo Switch. The Xbox One and Nintendo Switch versions from this family have been discontinued as they were supplanted by Bedrock versions (see above). Furthermore, in the flurry of rebranding that accompanied the Better Together update, the remaining console versions were collectively dubbed Legacy Console Edition(s).

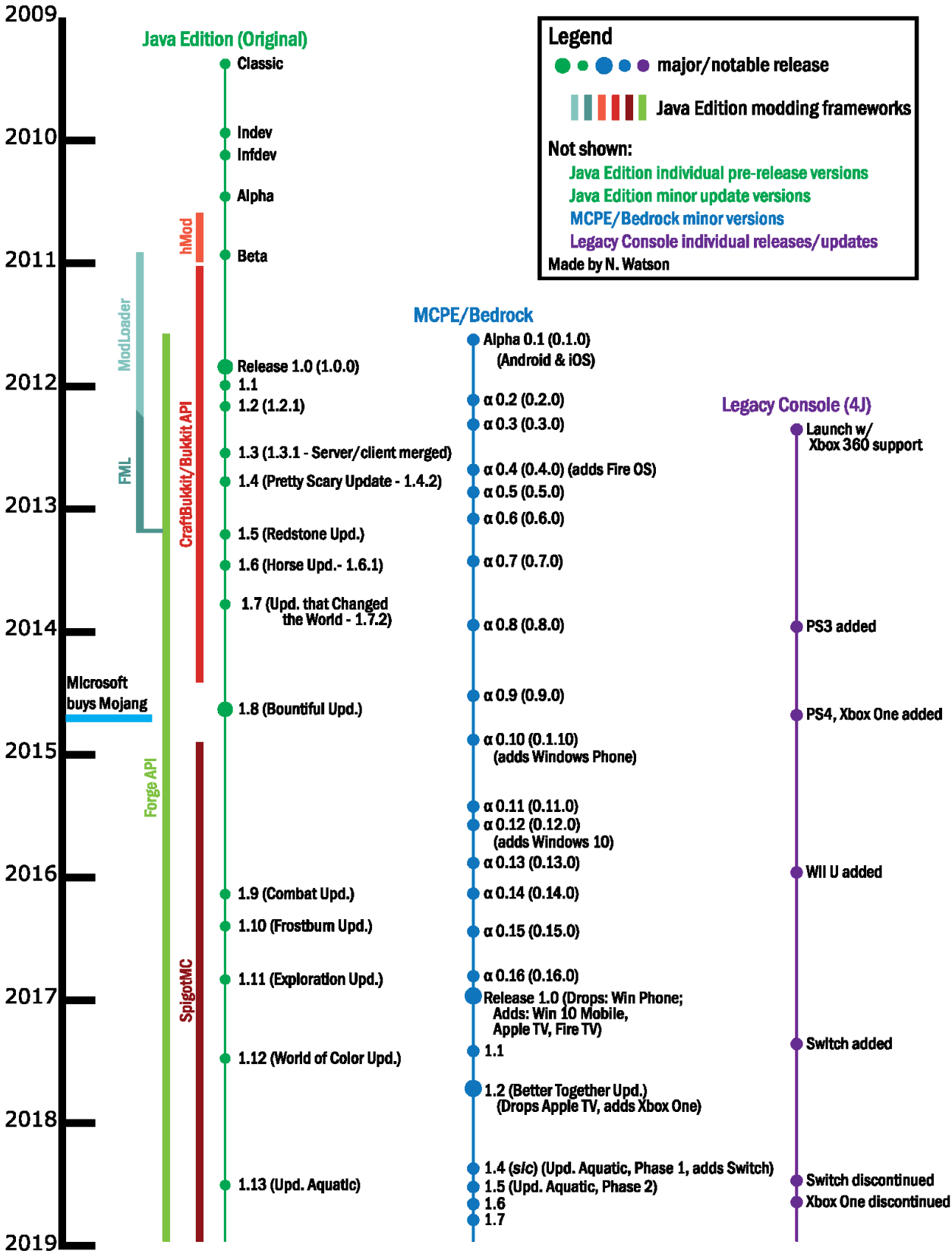
1.3.4 MinecraftEdu and Minecraft: Education Edition

MinecraftEdu began as a mod of Java Edition designed for classroom use and distributed from 2011 to 2016. The developer, TeacherGaming, had a license from Mojang to sell MinecraftEdu as a separate product. Microsoft purchased MinecraftEdu in 2016 and replaced it with a new product, Minecraft: Education Edition (MCEE), which is based on the Bedrock engine. Although I will not be discussing MinecraftEdu/MCEE in detail (and it is not shown on the timeline diagram in Figure 1-3), it deserves mentioning as an example of a total conversion mod being re-absorbed by the game's original developer, much like the example of *Half-Life* mod *CounterStrike* that often comes up in existing literature on modding (e.g. Kücklich, 2005). Furthermore, the majority of scholarly work

[4] In this context, an Application Programming Interface (API) would be a set of programmed tools and standards that would allow modders to easily write code that communicates with core Minecraft components.

on Minecraft so far has come from the fields of education and child development, and much of that work is specific to the educational editions (see Subsection 2.5.2).

Figure 1-3. (Page 10) **Timeline: 10 years of Minecraft development**, showing the major updates and changes for the three key edition-families, as well as the rise and fall of fan-developed modding toolsets and frameworks for Java Edition.



Data sources:

Minecraft Wiki: https://minecraft.gamepedia.com/Version_history
ModLoader legacy downloads: http://www.mediafire.com/?jc2n88a51xdfd
FML downloads: http://files.minecraftforge.net/fml
Minecraft Forge legacy files on Sourceforge: https://sourceforge.net/projects/minecraftforge/files/1.0.0/
hMod legacy downloads: http://hey0.net/minecraft/
Minecraft Forge downloads: http://files.minecraftforge.net/
Bukkit forums: https://bukkit.org/forums/

1.4 Beginning to define "mods"

> *Minecraft mods are mods because that's what the community calls them. Translating that terminology to another game is a pain. Each community has a different meaning and language. Trying to define it is a fool's errand.*
>
> —Alta, Minecraft modder

Let us, then, embark on a fool's errand.

"Mod" is short for "modification," and "modding" is the practice of creating such a modification, but what actually constitutes a mod and modding is ambiguous and contested. For the purposes of this research, I propose the following definition:

> **A mod is a piece of software or a body of digital data that modifies the process and experience of a digital game, such that it differs tangibly from its original form, on either a procedural level or a cultural/experiential level.**

This is similar to the emic (internally-understood) definitions espoused by modders whom I interviewed for this study. For instance, when asked to define "mod," modder ScriveShark responded:

> Any modification to the content or mechanics of a game through the manipulation of a game's files or software, including adding files to be processed in the normal execution of the software and in ways the software was designed to expect.

Exactly what constitutes a mod, and what is involved in making one, can vary from one game to another. The process of Minecraft modding involves reverse-engineering the original software product into source code, making changes to that source code, recompiling, and posting the new files on internet websites for players to download and "inject" into their own copies of the game. Over time, members of the community have developed utilities to automate the reverse-engineering and code injection parts of the process, such that modders can focus on the actual content they wish to alter, and players with minimal technical knowledge can install pre-built mods on their own computers.

Typical Minecraft mods might add a new kind of machine or a new species of animal to the game, or might alter the user interface to serve some utility function, such as automatically highlighting dark areas of the world where dangerous monsters could appear. Figure 1-4 (page 14) depicts gameplay with some popular mods. In discussions, players distinguish between modded Minecraft and "VANILLA"—meaning non-modded standard Minecraft, although we will soon see (in Chapter 4) that even without mods, there is no one standard Minecraft.

Minecraft mods come in many shapes and sizes. I provide the following list in order to give a picture of the sorts of things that mods can do, and to provide reference points for the discussions to follow. These categories are of course socially constructed, unstable, and inconsistently applied by scholars and players alike—with much of the confusion stemming from the differences between games. The categories I describe here are grounded in my modder-informants' own understandings of these terms, based on interview responses and the organization of mods at the Better Than Minecamp convention described later on in Chapter 5.

Texture packs straddle the boundary between mod and not-mod. Since they are supported natively by the vanilla software, they do not alter the running code in any way. They can change the appearance of blocks, items, and creatures, but usually have no effect on gameplay itself.[5] New textures can be designed using widely available image-editing software, and are easily distributed online, downloaded, and loaded into the game. In Minecraft version 1.6.1 (July 2013), texture packs were folded into the more expansive concept of RESOURCE PACKS, which include the ability to alter simple three-dimensional shapes, sound effects, music, splash screens, and fonts.[6] According to my informants, Minecraft players tend not to call these mods at all, while admitting that texture swaps

[5] As usual, there are boundary cases. Swapping in a texture file with transparent pixels for a block that Minecraft considers to be opaque can create a sort of X-ray effect that allows the player to see through cave walls—a boon for diamond-hunters but definitely a circumvention of the intended rules of play.

[6] Version 1.13 (July 2018) introduced *data packs*, which allow further customization of Minecraft gameplay without any actual coding. Using data packs containing JSON files (see glossary), players can do things that were previously only possible with code-altering mods, such as defining custom crafting recipes or changing the rules for generating in-world dungeons and loot chests. Data packs came on the scene towards the end of my research and are not analyzed in detail here.

in other games can be considered mods, based primarily on whether or not the game has a built-in function to support them. According to ScriveShark, "There should be a change or addition to the mechanical operation of the game. Adding *more* paintings would be a mod, or adding more cards or even variations of visuals, but simply replacing the appearance or sound of things […] is more of a customized localization" (emphasis added; see glossary for LOCALIZATION).

Atmospheric mods, similar to texture packs, alter the look and feel of the Minecraft world, without radically changing the underlying game mechanics. These mods, however, require actual changes to the code because they add new artifacts and scenery to the world, such as new plants and animals, rather than re-skinning existing ones as texture packs do.

Unofficial patches are designed to fix bugs and performance problems in the vanilla product. Fans make these when they do not want to wait for the developer to get around to issuing official patches. This was particularly common in Minecraft's pre-release days of open Alpha and Beta testing.

Tweaks and rebalancing mods attempt to address what some players perceive as flaws in game design. One of the most common examples is RECIPE-tweaking, in which the materials required to craft certain items are changed to make those items either easier or more difficult to create.

Gameplay mods add entirely new game mechanics and/or significantly alter existing ones. These are usually the largest and most complicated mods. In extreme cases, these mods amount to what Scacchi (2011) calls *total conversions*, particularly if they replace the original mechanics. A gameplay mod typically has its own trajectory of player progression—its own learning curve, materials to acquire, and skills to master. **Tech mods** and **magic mods** are two broad sub-genres of gameplay mod. The former focuses on creating large factories of complex, interconnected, industrial machinery, with the goals of automation, optimization, and multiplication of resource-based wealth (Figure 1-4, bottom). Magic mods are quite similar to tech mods, but they have a fantasy/arcane-arts

theme instead of an industrial/technological theme. Tech and magic mods also often add entirely new spaces (called dimensions) to the game, e.g. a mystic forest realm that can only be reached through an eldritch portal, or a lunar colony accessible by rocket ship.

Figure 1-4 . Minecraft gameplay screenshots, featuring mods. Top: One popular mod, *Millénaire*, adds autonomous villagers who gather their own resources, build their own houses, and will trade goods with the player. **Bottom:** Popular "tech mods" add industrial machinery and lead players to construct massive automated factory systems for the production of useful materials. Pictured here are pipes (*Thermal Expansion*), storage tanks, and steam engine boilers (*RailCraft*) that have been damaged by a recent explosion, due to a malfunction in the water-pumping system. (Screenshots by N. Watson.)

Utility and user interface mods are designed to facilitate interaction between the user and the game. One example is *InvSort*, a mod that can automatically sort and organize a player's inventory with a single keypress.

Fans have created several **external programs** that are not strictly mods in that they do not change the running code for the game, but they still modify the gameplay experience. The most notable of these are the map editing programs that provide players with powerful tools to create and edit Minecraft worlds from outside the game itself.

Finally, **custom maps** are sometimes called mods even though they may be created entirely within the vanilla program: a player uses the in-game "Creative Mode" building tools to create a space for others to explore—perhaps with puzzles or prescribed adventuring goals. The map or SAVE FILE associated with this world is then distributed online through sites like MinecraftForum.net and PlanetMinecraft.com for others to download and play with. Nothing about the game program has actually been changed, but some commentators apply the "mod" label nonetheless (e.g. Christiansen [2014] on the "Skyblock" mini-game; see Section 2.4 for further discussion). The temptation to refer to custom maps as mods may come from first-person-shooter (FPS) modding in which the virtual spaces (maps) are immutable by vanilla means, so all custom maps are *de facto* mods. Since modding of commercial games—and academic work on the subject—largely originates with FPS games, it is not surprising that their terminology sticks even as context changes. Furthermore, while a custom map can in principle be designed in-game, it is often much easier to use a fan-made external editor program like MCEDIT. Finally, by radically re-orienting play goals around a different environment with different constraints, some custom maps transform Minecraft in a way akin to what total conversions accomplish, creating a subjective player experience that is difficult to distinguish from what more prototypical mods do. However, as with texture packs, Minecraft players are reluctant to refer to custom maps as mods.

Machinima—the use of video game engines and video capture software to create movie-like narratives—is often lumped together with modding (see e.g. Scacchi, 2011), yet it works purely on the cultural/experiential level and creates an end product of a wholly different, non-game media form. For instance, watching the comedic video series *Red vs. Blue* (Rooster Teeth Productions, 2003) is clearly a different sort of activity from playing *Halo: Combat Evolved* (Microsoft Game Studios, 2001), the game environment in which the series was "filmed." Therefore, although machinima will come up in discussions of modding literature to follow, I do not classify it as a form of modding and do not investigate the rich creative domain of Minecraft machinima in this study.

1.5 Modding frameworks

The earliest form of Minecraft modding involved a technique known as "jar modding," in which modders hack the game's core files. This process is described further in Chapter 4.

However, various platforms, tools, and APIs were created by Minecraft fans to facilitate modding and to make different mods compatible with each other. I briefly introduce these here so that their names will be familiar when they reappear in the subsequent pages.

This research focuses mainly on mods that are made using the Forge API (https://minecraftforge.net), a powerful and popular mod-making framework that has been around since 2011 (see Figure 1-3 on page 10). Forge is designed to reduce (and, increasingly, eliminate) the need for most modders to directly edit Minecraft's core code; instead, they create modules that can intervene, via channels facilitated and sanctioned by Forge, to alter how Minecraft works. Forge was initially used alongside a separate utility called ModLoader, which is what made modularity possible, but ModLoader was succeeded by Forge's own built-in mod loader, FML. More detail is provided in Chapter 4.

Making mods using Forge involves writing Java source code, but at least one tool exists that positions the modder at a distance from the code itself. MCreator is a mod-developing tool developed by Slovenian software startup Pylo. Backed by machine-generated, Forge-compatible

Java code, it offers a graphical interface with which modders with minimal programming experience can create mods by piecing together the sorts of "elements" that mods typically add—e.g. new blocks, tools, or administrator commands. While this model of assembling mods out of a collection of templates does not allow for as much flexibility as direct Java programming, MCreator allows access to its Java back-end for advanced modding.

Jar modding and Forge—with or without MCreator—are not the only ways to go about modding, but almost all mods that add new blocks to the game (and many other mods besides) are made using one of these two methods. Other methods, described briefly below, are not considered in this research except in passing. A system called LiteLoader (https://www.liteloader.com) is intended to provide a "minimally invasive" way to install mutually-compatible mods, particularly CLIENT-side mods that don't change core game mechanics, like user-interface tweaks (thus, LiteLoader-modified clients are compatible with vanilla SERVERS). The Bukkit API, along with its predecessor hMod, embody a very different modding paradigm. These alter Minecraft SERVER code to change game mechanics, but are limited in what they can accomplish. Bukkit plugins can do things like adding new typed administrator commands, providing scripted encounters with NPCs (non-player characters), implementing trade systems, enforcing property ownership regimes on multiplayer servers, teleporting players around, and changing world generation rules. The Minecraft "mini-games" industry, with its *Hunger Games*-themed deathmatch tournaments and Skyblock competitions (and many other Minecraft-based play scenarios), relies largely on Bukkit mods to implement special game rules. What Bukkit *cannot* do is add new kinds of blocks or machines, or otherwise make any changes that would need to be reflected in each player's client program. This is both a limitation and an asset: it means that people can play on Bukkit servers with vanilla clients. In fact, hundreds of thousands of people who regularly play on multiplayer servers are themselves running vanilla Minecraft, and may be completely unaware of the fact that the servers they play on are heavily modded.

The term "Bukkit" refers to an API for making Minecraft plugins. The actual server wrapper—the component that allows those plugins to talk to the core Minecraft program—is called CraftBukkit, but it has been defunct and unavailable since 2014. However, other Bukkit-compatible server wrappers such as SpigotMC have taken its place.

Regardless of the platform or API in use, most Minecraft modding efforts ultimately rely on the Minecraft Coder Pack (MCP), a suite of tools for converting Minecraft binary files into human-readable Java source code. A few, like the Fabric system (https://fabricmc.net/) that emerged in late 2018, implement their own toolchains for these purposes.

1.6 Blueprint

1.6.1 Purpose and scope

The overarching goal of this work has been to provide a description of the culture of a community of Minecraft modders and their practices of modding, including the details of the technical and social work performed by modders. I have further sought to determine the sites of contested meanings—among modders, between developers and modders, and between modders and players; how these conflicts arise and are mediated; and what social, cultural, and discursive resources are mobilized by stakeholders. These findings serve to add a new perspective to ongoing scholarly efforts to explicate the causes and circumstances—social, cultural, and technical—that have given rise to, and sustained, active fan participation in the co-creation of Minecraft.

These purposes are revisited in Chapter 3, where they are connected to my research questions and theoretical frameworks.

1.6.2 About this document

The current chapter has served to introduce Minecraft and the concept of mods. Although modding has a deep and rich history, with roots in early 1990s first-person-shooter (FPS) games, I have refrained from recapitulating a comprehensive history and typology of modding in general, as

this work has already been accomplished by others (e.g. Nieborg and van der Graaf, 2008; Sotamaa, 2009; Sihvonen, 2011; Champion, 2012a; Christiansen, 2012).

In the following chapter (Chapter 2 – Review and Commentary on the Literature), I undertake a review of current academic thought in the areas of fan studies and participatory culture, before proceeding to a brief discussion of prior work on game modding. Subsequently, I summarize existing scholarship specific to Minecraft, with particular attention to participatory production and modding, followed by a commentary on areas that the literature has yet to address.

Chapter 3 – A Workbench of Theory and Method introduces the major theoretical frames that I have used to make sense of Minecraft modding, setting the stage for their reappearance in subsequent chapters. Here I also outline my methodological approach and the contours of my research activities.

In Chapter 4 – Settling Minecraft, I trace the story of how early Minecraft came to be "settled" by innovative player practices and "soft modding," with player improvisations gradually becoming fully implemented and integrated into the software. "Hard" modding is described as having developed along a similar trajectory, from ad-hoc improvisations to established frameworks and methods.

Not satisfied with terrestrial frontiers, I take a virtual journey to a space station in Chapter 5 – Better Than Minecamp: An unconventional convention. This is a first-hand look at an online event from 2017 that celebrates Minecraft modding while also mocking (with varying degrees of playfulness or severity) Minecamp, Mojang's official Minecraft convention. Here we will see a snapshot of the "settled" state-of-the-art of Minecraft modding, including how gleefully *unsettled* it still is in many ways. We will also get a preview of some of the key issues on the minds of prominent modders.

Back on Earth, Chapter 6 – Operational Logics and the Rationalization of Modding delves into the technical bits of Minecraft and modding, and considers how modders grapple with this

technical work. I expand Wardrip-Fruin's (2009) notion of *operational logics* into a model that extends through the domains of computation, programming, and play, to provide an account of how modding practices have been subjected to increasing strategic rationalization.

Chapter 7 – Modding Discourses examines the role that modders' discursive activities play in innovating, settling, contesting, and remaking modding practice. A brief summary of the means that modders use to communicate is followed by a study of how modding is contested among modders (not just between modders and commercial developers), how "proper" practices are constructed, and how gaming capital (Consalvo, 2007) circulates within the community. Programming practices themselves are also considered as a kind of discourse.

In the Conclusion, I consider the potential lessons of this research for the co-creative development of video games. Following a discussion of potential directions for future research on Minecraft and game modding, I focus critical attention on notions of Minecraft as an exceptional, endlessly extensible *platform* for constructing play experiences. I close with some final thoughts on the oscillations between tactical and strategic modes of cultural production.

I have included three appendices to help illustrate the occasionally-opaque technical concepts discussed herein. Appendix A – A Very Brief Summary of Object-Oriented Programming Concepts in Java provides a crash-course in the major jargon and main concepts of Java programming, while Appendix B – Custom Block Mod Code presents an annotated inventory of all of the source code for a simple example Minecraft mod. Appendix C – Forge Hooks: Registries and events, provides an explanation of some of the inner workings of the Forge modding framework that are mentioned elsewhere in the paper.

The use of SMALL CAPITAL LETTERS indicates a technical term from Minecraft or programming that has a specific definition for the purposes of this text; each such term has a corresponding entry in the Glossary. (Small capital letters are only used when the term is initially introduced.)

To protect the privacy of research participants, their names have been changed in these pages, except in cases where they requested, or gave permission for, identification. The names of some mods have also been obfuscated for the same reason. Non-participant public persons in the Minecraft community, for whom all data reported here is derived from publicly-accessible archives, are (in most cases) identified by the screen-names they actually use.[7]

It is common for documentaries to begin with a disclaimer that the opinions expressed by interviewees are their own, and do not necessarily reflect those of the author or publisher. While that is true of this document, it is equally important to note that my opinions as the author are my own, and the participation of any research informant or interviewee does not imply that they endorse or agree with my conclusions.

Technical data and information regarding Minecraft gameplay features are valid for Minecraft: Java Edition versions up to at least 1.12.2, which was released September 18, 2017.

To ensure the long-term availability of web-based sources, URLs for Internet Archive Wayback Machine (http://web.archive.org) versions of referenced web pages are provided where practical (archived versions are not provided for general website URLs, downloadable files, videos, articles hosted on an academic journal's website, and resources that already have a DOI or database presence). Where a suitable snapshot of a web page did not already exist in the archive, a new one was created. The inclusion of an Archive.org link does not necessarily mean that the original document is no longer available. For archive links in this document, the original URL always forms the second part of the archive link. To make it easier to separate out the original addresses, the portion corresponding to the original URL is underlined in this document's footnotes. In the bibliography, archive URLs are provided in *italics* at the end of applicable bibliographic entries.

[7] I have maintained the typographic conventions used by individuals in rendering their own screen names. For instance, if they do not normally capitalize the first letter of their screen name, I also do not capitalize it in this document, unless it appears at the beginning of a sentence.

II. REVIEW AND COMMENTARY ON THE LITERATURE

2.1 Under the influence: Mass media and audience passivity

> *Of course the media affect emotions and behavior. That is why people use them.*
>
> —Jeffrey Goldstein (2005, p. 350)

I begin by addressing an assumption, present in both popular discourse and significant bodies of scholarship—especially early-to-mid-20th century cultural studies—that mass media entails a sharp division between powerful, corporate media producers on the one hand, and powerless consumer audiences on the other, who are the passive recipients of whatever "messages" those producers choose to disseminate. The flow of information is seen as moving in one direction only—from the media elite to the masses. The discourses favoured by the elite must therefore dominate, so that they have unmatched persuasive influence over the creation and consumption of culture. The "culture industry," as Adorno and Horkheimer (1944/2002) originally named it, is hegemony calculated to produce homogeneity, and consumers are cultural "dupes."

While Adorno, Horkheimer, and other Frankfurt School theorists had good reason to be concerned about the possibility of mass media being used to distort truth, encourage docility, and spread propaganda, the uncompromising cynicism of this view tends to erase the real and worthwhile work done by the "masses" to interpret, answer, configure, rework, and re-circulate media objects. Despite decades of discourse to bear this out, the claim of passivity has proven tenacious, enjoying the occasional resurgence as a go-to assumption for researchers who find themselves in the position of having to make sense of the next new trend in media.

2.1.1 The spectre of effects studies

Videogames found themselves under this lens from the 1980s to the turn of the century, the heyday of *effects studies* (e.g. Anderson & Ford, 1986; Provenzo 1991; Dill & Dill, 1998; Kirsh, 1998). Highlighting themes of addiction, aggression, delinquency, and violent crime, this research focused

on what sorts of messages games were communicating to players (especially children), and what impact this has on a player's thought, behaviour, and perception of the world. According to Provenzo (1991, p. 65), a significant body of work has been devoted to importing two models for understanding the relationship between television content and aggression—catharsis theory and stimulation theory—into the realm of videogames. Goldstein (2005) is highly critical of this "willy-nilly" application of television research, since "video games differ from television and film not only in their interactivity, but in the nature of their stories, in their open-endedness, and in their ability to satisfy different needs of their users" (p. 342).

Some discourses feature an odd mixture of psychological experiments that fail to provide evidence in support of alarmist hypotheses, coupled with vague analysis that seems to try to keep the anxiety alive by leaving open the possibility of a *future* verification of the social ills of gaming. Kirsh (1998) claims a weak, short-term correlation (in children) between playing a violent game and attributing hostile motives to the actions of others in the real world—a finding that supports the notion that *games have effects* (however brief), but offers no substantiation of popular claims linking teen violence to gaming. Provenzo (1991, p. 70) admits that there is no evidence that games contribute significantly to deviant behaviour, yet leaves open the door for worry about other unnamed social and cultural "impacts" of gaming. His claim that "concern about the games is in fact justified" (1991, p. 50) apparently hinges not on direct demonstration of effects but rather on the presumption of effects based on game content. The content itself is considered outside of its actual use context and through reports of an anecdotal nature, which cherry-pick what is most exceptional and sensational and re-frame it as typical: Provenzo holds up *Custer's Revenge* (Mystique, 1982), a fringe game made by an obscure pornographic games producer for the Atari 2600, as evidence that videogames "have a history of being sexist and racist" in addition to being violent (1991, p. 52). While I do not deny that racist and sexist undertones are and have historically been present in many

mainstream games (although not generally more so than the average Hollywood blockbuster), *Custer's Revenge* is far from representative of the medium's offerings.

Williams (2003) argues that the "story of vilification and partial redemption" of videogames is tied to the social tensions of the times, particularly the conservative anxieties of the 1980s—thus they probably tell us more about reactionary politics than about games. In fact, the early framings of games research in terms of violence and addiction may have taken its cue from popular media and political discourse, such as the U.S. Surgeon General's infamous 1982 claim that games were producing "aberrations in childhood behaviour" (Provenzo, 1991, p. 50).

Goldstein (2005), who provides an extensive review of behavioural research on videogame violence from the 1980s through 2001, is unequivocal in his criticism: "Discussions of violent video games are clouded by ambiguous definitions, poorly designed research, and the continued confusion of correlation with causality" (p. 341). Further, "there is no evidence that media shape behavior in ways that override a person's own desires and motivations" (p. 350).

Effects studies were not universally negative. Williams (2003) describes "utopian" research narratives standing in opposition to these alarmist discourses—studies that tried to partially redeem games by pointing to the ways in which they could educate children, build cognitive skills, or provide a context for positive social interaction among family members (e.g. Mitchell, 1985).

The uniting feature of effects studies, however, is the presumption of passivity. "Missing from research," writes Goldstein in a passage that (unintentionally?) echoes Caillois (1961), "is any acknowledgment that video game players freely engage in play, and are always free to leave, or pause. Except in laboratory experiments, no one is forced to play a violent video game" (Goldstein, 2005, p. 353). Effects studies fixate on *what media do to us*, rather than on the far more exciting question of *what we can do with them*.

If anything, it is ironically the theories of creative-participatory media culture, rather than the tropes of corruption through passive consumption, that are more easily linked to examples of

deviance. When in 2013, a nine-year-old boy was arrested in Orlando, Florida for bringing weapons to school, his father claimed that the boy was indulging in a Minecraft character fantasy, and further argued that there was no chance that the child would have harmed anyone (Thompson, 2013). The following year, 12-year-olds Anissa Weier and Morgan Geyser *did* harm their classmate Payton Leutner while acting out a twisted fantasy inspired by the "Slenderman"[8] internet fandom. Actual incidents of concern resemble dangerous live-action fanfictions more than cases of violent behavioural conditioning. They also remain exceptionally rare, and are seldom explainable in terms of media fandom alone—serious mental illness being implicated in the case of Weier and Geyser (Almasy, 2017). It seems that there is not much worthy of true concern for alarmist media researchers to find in the realm of participatory culture, either.

2.2 Voices from the listeners

The assumption of audience passivity, while tenacious, has been thoroughly dismantled by decades of research in media and cultural studies, making it more difficult in turn to maintain the illusion of a clear division between media producers and consumers. Uses and gratifications theory, typified by the approach of Blumler and Katz (1975), offered the initial rebuttal to effects-focused thinking by reframing media consumption as an active pursuit. It forms no small part of Goldstein's indictment of effects research that was cited above. The turn to participatory culture steps beyond uses and gratifications, identifying not only the ways in which audiences can resist hegemony, re-interpreting and re-signifying the meanings offered by media objects, but also how they can become active participants in shaping mainstream of media texts.

[8] The Slenderman neo-folklore meme did not originate with videogames, but with "creepypasta"—short, homebrew horror stories and images that are shared through online message boards. It has been further embellished through YouTube videos. However, it has also been the subject of videogames, such as *Slender: The Eight Pages* (Parsec Productions, 2012). Minecraft features a Slenderman-inspired monster called an Enderman. Minecraft is, of course, also the setting for another creepypasta/fanfic custom featuring a humanoid monster known as Herobrine who, despite having a radically different appearance, is seen in some stories to operate in a manner similar to Slenderman, and is also often associated with the Endermen (see Gu, 2014, p. 139-140).

First and foremost, consuming media is work![9] Audiences never merely receive, but engage actively in the construction of something meaningful out of media material, even when there is no overt experience of resisting cultural hegemony. Stuart Hall (1980/2001) is well known for describing how audiences apply available cultural codes in understanding media messages—and the considerable interpretive latitude they have in doing so, producing "oppositional" or "negotiated" readings as alternatives to the intended (hegemonic) reading. Fiske (1987/2001) tells us that texts are understood (decoded, interpreted) in terms of their relations to other texts (which may or may not explicitly reference one another). Readers must therefore do the active work of mobilizing cultural resources—the knowledge of other texts—in order to construct a coherent reading of the text in question: every text, through its reading, is always in the process of becoming another text's paratext.

Michel de Certeau (1984) argues that media audiences ("readers") are active in meaning-making, but marginalized by a productivist discourse of sanctioned and regulated meanings. Audiences are nomadic "poachers," living and "making do" on the periphery of texts, employing "tactics" (de Certeau, 1984) of reinterpretation to construct their own readings—what Murray (2004, p. 20) calls "semiotically self-determining."

Jenkins (1992) borrows this metaphor of poaching to describe specific examples of how television fans engage in a participatory culture, reading and reworking the texts that they consume. Jenkins finds that fans are constantly asserting their right to renegotiate and redefine the terms of media consumption, and to freely attach their own meanings, or reject those offered by producers. They grab (poach) aspects of the text that are consistent with their preferred readings, and reject those parts of the text (or of a paratext) that are not.

So far, these theories address the work that goes into audiences' *use* of media that are provided to them, but we need to extend these in order to arrive at an understanding how audiences *make* media. Jenkins' "convergence culture," the notion that the differences between media forms

[9] This does not imply that it is *labour*, or that it cannot be playful or fun.

are of decreasing importance given the sharing of techniques and the portability of narratives across media (Jenkins, 2006), may not seem at a glance to pertain directly to fan production, but in fact it directly implicates participatory culture by collapsing distinctions between "old" media (mass media, few-to-many dissemination) and new media. Jenkins connects participatory and convergence cultures by recognizing how new media have made the once-invisible work of fans visible, and how media properties—*including* books, films, and other forms that were once considered old media—are articulated through the joint efforts of corporations and fans (2002). The blurring of media forms goes hand-in-hand with the blurring of producer/consumer distinctions. Although media corporations still frequently attempt to exercise a monopoly over the legitimization of meanings within a media text, they are increasingly dependent on fans to re-circulate and remediate their content, and are increasingly likely to acknowledge that dependence (Jenkins, 2002). In this context, there are *only* negotiated readings, or more accurately, co-constructed ones.

In game studies, several scholars have already taken production-as-poaching and participatory culture as lenses for understanding a range of creative/productive play practices and modding activities (Jenkins, 2002; Pearce, 2006; Sotamaa, 2009, pp. 16, 87; Sihvonen, 2011, pp. 83-84). Parallels are drawn between classic (literary/television/film) fandom studies and digital gameplay. Pearce sees the way that player communities such as *Myst: Uru* fans "expand the game narrative and eventually begin to take it over" as similar to Jenkins' account of the "Trekkie phenomenon" (Pearce, 2006, p. 19). Sihvonen notes that "the [non-game media] industry's interests in promoting fan activities for its own profit resembles the practices of game corporations that have supported amateur game development and modding for decades" (Sihvonen, 2011, p. 84). However, some game scholars also warn that such prior theories of the semiotically self-determining work of media consumers are not enough to describe the full range of player activities. Sihvonen is critical of the tendency to apply—without significant modification—the frame of creative resistance and

subversion of media power from fandom studies to gaming and game modding. Analogies between operations of industry power are seen to break down because of

> a significant difference in scope and the dedication with which traditional media corporations seek to secure and maintain their intellectual property . . . in comparison to game developers, who seem to utilise different strategies to safeguard their economic interests. (Sihvonen, 2011, p. 84)

Sotamaa summarizes the point, stating that "the dynamic between players, games and the game industry . . . requires more multifaceted approaches" (2009, p. 26).

2.2.1 Wandering lines: How videogames invite participatory action

Even those who dabble in effects studies admit that gaming involves doing *something* active that is not seen in television viewing:

> To begin with, television is a passive medium; viewers have virtually no control over what takes place on the screen. Video games, in contrast, represent an active medium. Television does not require the viewer to pay constant attention to it, whereas video games require total concentration. (Provenzo, 1991, p. 66)

To claim that the salient active component of playing a game manifests in the game's requirement for total concentration is to narrowly construe the meaning of play as reflexive response to fast-paced stimuli—a position that might have been understandable in the 1970s and 80s when most (though not all) popular videogames were action-oriented, but falls apart completely in a world that has long contained the likes of *Zork*, *SimCity*, *Myst*, *Final Fantasy*, *Portal*, *Skyrim*, *Gone Home*, *Firewatch*, *Stardew Valley*, and (of course) Minecraft, to name just a few games that clearly do not require constant and total concentration and are even known to prominently feature quiet, meditative moments.

However, even as we acknowledge that reading a book or watching a film demands that the media consumer do *something*, it still seems that games require the player to do *something more*. Espen Aarseth (1997) has argued that digital games are "ergodic texts," which makes them a different sort of thing from other media texts. Ergodic texts are simultaneously objects and processes (Aarseth,

2001). They are understood to exhibit two key characteristics: in order to be used, they require intentional, non-trivial effort on the part of the user (even a tranquil effort or a divided attentiveness imply intentionality); and rather than presenting a linear narrative, they provide a possibility space to be explored and navigated. These characteristics give rise to additional user functions:

> [T]he central user function of literature or film is interpretive . . . The *interpretive function* is obviously present also in the case of games. In addition, game players are required to perform *explorative functions*, as in deciding which path to take, and *configurative functions*, as in choosing and creating parts of the game. (Sotamaa, 2009, p. 67; emphasis original)

Golding (2013) suggests that instead of a configurative function, games require a "navigation" function. He argues that configuring is strategic, requiring top-down knowledge: "to configure is to imply a holistic knowledge of a text; an operator at a switchboard, or a city planner with a map" (2013, p. 44). A player's actual lived experience, he argues, is more like de Certeau's figure of the street-level walker/wanderer, who navigates by tactics rather than configuring by strategies.

Following Sotamaa, Sihvonen writes that "all computer games are inherently configurative and participatory in that they 'emerge' as a result of the players' inputs, offering feedback, rewards and further challenges" (2011, p. 39). She further emphasizes, drawing on foundational ludology theory from Aarseth, Frasca (1999), and Juul (2001), that interactive engagement with a digital game is not just about being able to perceive the game world, but about being able to manipulate it and perceive the results of those manipulations (Sihvonen, 2011, p. 109). Thus gameplay is driven by a recursive feedback loop: the player explores the possibility space, discovering both simulation rules (Sicart, 2009) and signifying possibilities in the feedback that the game returns in response to their exploration; they then reconfigure their gameplay in response to this new information, which results in a new pattern of exploration and new feedback from the machine. This symbiotic player/machine loop drives home the point that the player is never solely a function of the game, and games cannot be understood in isolation from their players (Sicart, 2009; Sotamaa, 2009, p. 68).

As de Certeau (1984) established and Sotamaa (2009) reaffirmed (see above), the interpretive function is already present in the reading of conventional texts. However, Jenkins' (1992) study of the creative products of television fandom also invites the explorative and configurative functions into the realm of conventional media consumption, although in this case they would appear to be elective functions, limited to a subset of users, not required as Sotamaa asserts they are for games. Sihvonen's claim (above) that a game's text "emerges" from a player's interaction with the world foreshadows a way for understanding play as co-constructed between player, machine, and game-designer, of an ephemeral linear text (2011, p. 31). The player's navigation through an ergodic text produces a linear text similar to the "wandering lines" of de Certeau's nomadic readers (1984, loc. 130); the traces of a navigation in ergodic space make explicit the work that players and game rules do to configure the shape of those wandering lines.

But is that work a kind of *co-authorship*? Does the reader/user/player do a kind of writing? Tremblay, Colangelo, and Brown think so (2014). They address Minecraft specifically, arguing that the product—the collection of binary code authored and compiled by Notch and Mojang—needs to be understood as a kind of language or grammar. Minecraft gameplay is

> ultimately derived from Minecraft's code, but in largely the same way that every work of literature—this essay included—is derived from the rules of a language. Thus, the specific gameplay produced, just like a work of literature, is, as Joyce (1986 [1922]) phrased it, 'an actuality of the possible as possible' (p. 21)—the act of playing, like the act of writing, takes what existed only as potential and brings it into actuality. It is thus the player who is the "author" of the game. (Tremblay et al., 2014, p. 77-78)

Tremblay et al. find themselves in the position of having to speak to a pre-existing assumption that games are like books, static objects authored by an identifiable entity and then distributed to users. But there is another way to arrive at play-as-writing: if we remember that games are, after all, software, and think of playing a digital game not as consuming a media product, but as a kind of computing activity. Daniel Punday (2015) demonstrates that writing has proven to be a pervasive and captivating cultural metaphor for all manner of computing activities (that is, human

interactions with a computer), from configuring, programming, and data entry, to calculation, communication, and online research. Punday cautions, however, that the analogy is fraught with tension and contradiction. What of, for instance, the researcher's inscribed "trail" through the archive, the one Vannevar Bush (1945/1995) imagined as an output of his Memex? Does it (Punday asks) create anything new (2015, pp. 6, 150)? Is it a form of writing? The answer certainly has implications for how we understand the wandering trails described by de Certeau, too. De Certeau, for his part, specifically said that textual poachers were "far from being writers," as writing "accumulates, stocks up" (1984, loc. 2532).

But gaming does not seem to even get consideration under the fraught rubric of computing as writing, since it seems to stand apart from these other acts of 'true' computing. Perhaps it is simply a binary calculation: if a game programmer is already understood as a writer, then a game player must be a reader. In any case, game designers can hardly be faulted for trying to mobilize comparisons to literature and authorship in order to lend legitimacy to their oft-maligned medium (for instance, Punday notes that the Miller brothers, creators of *Myst*, preferred to describe themselves as authors rather than programmers or directors [2015, p. 148]). But the extension of the "writer" mantle to those who make digital games may paradoxically lock out those who play. More likely, the tendency to think of play as reading or watching, rather than writing, reflects forty-plus years of games being produced, packaged, and sold like books and movies. It was not always the case—the distinction between maker and player was once much blurrier (discussed further Subsection 2.3.2, below).

2.3 Commercial media meet the digital commons

Discussions of game modding and other player/fan activity are littered with slippery, overlapping terminology—terms like *participatory culture*, *media convergence*, *digital commons*, *fan production*, *volunteerism*, *prosumer* (producer-consumer), *produser* (producer-user), *elite fan*, *hacker*,

modder, and *maker*. Each term has a subtly different valence, and in some cases the difference is important to note.

I briefly summarize the important differences here so as to be able to subsequently discuss what the concepts have in common. Not all produsers are modders, because modding involves actually changing the behaviour of software in a manner not specifically supported by its developers, whereas produsage can include making fanfiction, machinima, game-inspired knitted plush toys, or even just a really nifty in-game creation. "Maker" specifically invokes parallels between media produsage and non-digital maker culture, importing connotations that users of this term wish to highlight. Similarly, "hacker" imports the implication of a specific motivation and orientation towards altering a technological artifact.

The common thread, however, is that we are now talking about users who go beyond interpretation, self-determination, configuration, and creation: they are sharing, or *visibly re-signifying and re-circulating*, what they have done with media. To be clear, it is not strictly necessary that users' creative outputs be shared for them to be authentic: a player who makes a quick mod to address their own gameplay-related pet peeve, and who never shares it with the world, is a modder nonetheless ("every good work of software starts by scratching a developer's personal itch," claims Eric Raymond [2000]). Moreover, since some now-visible modders started out with such invisible activity, it stands to reason that at any given time there are some "lurker" modders (to borrow the Internet term for one who passively reads a chat or message board without speaking up) who are in the process of becoming public-facing modders. However, this study does not attempt to decrypt such invisible activity, except after-the-fact as part of a now-visible modder's personal history, and therefore does not address what implications (if any) such activity has for media industries and fan cultures.

Where participation and produsage become most exciting is where they impinge on how other people understand and use a media object, and especially where they are imbued with practices appropriated from commercial media production and distribution.

2.3.1 Making, hacking, remixing

Scholars have long discussed how the coupling of digital media—which can be disassembled and reconfigured with an ease not seen in other media forms—with the connectivity and collaborative potential of Internet communications could give rise to a democratized ecosystem of creativity that challenges the hegemony of productivist mass-media models. Amy Bruckman argued in 1995 that "on the net, everyone is becoming an artist":

> Cyberspace is not Disneyland. It's not a polished, perfect place built by professional designers for the public to obediently wait on line to passively experience. It's more like a finger-painting party. Everyone is making things, there's paint everywhere, and most work only a parent would love. Here and there, works emerge that most people would agree are achievements of note. The rich variety of work reflects the diversity of participants. (Bruckman, 1995)

Around the same time, Eric Raymond identified two models of free software development, which he called the "cathedral" and the "bazaar" (Raymond, 2000). The cathedral meant software built by an elite group of designers according to their own standards, to be unveiled when the project leads judged it polished and ready. Cathedral-type software can still be free (*libre*) software, with source code provided to the public at release time, and is thus a challenge to the intellectual property regime favoured by purveyors of mass media (the Disneys, to link back to Bruckman's argument), but there remains the implication of a clear distinction between producers and users: the cathedral is constructed by the inner circle, *not* co-constructed by the community. In the image of the bazaar, on the other hand, Raymond imagines decentralized production in which participants can move fluidly between being providers and consumers, and in which creative outputs can be passed horizontally, with value added at each exchange. Raymond believes in particular that a greater number of eyes and creative minds attending to software products in the bazaar would be able to address bugs more

quickly and effectively. In this view, extending collaboration would actually lead to higher quality products, not just messier finger-paintings.

Benkler (2006) uses the terms "social production" and "commons-based peer production" to describe creative/productive activities that are enabled by networked society. He argues that commons-based peer production is non-hierarchical and is able to operate outside of the systems of financial incentive that dominate in mass media production. Benkler's two terms, though used more or less synonymously, actually imply the need for two different (though related) ingredients: the networked society of practitioners (the "social" or "peer" aspect), and a collection of materials and goods held in common that are available for practitioners to use as needed (the "commons" aspect). In feudal Europe, commons were portions of a noble's estate on which commoners had certain collective rights to use or extract resources—for instance, putting their own livestock out to pasture. A *digital commons* suggests a reservoir of available cultural "stuff"—ideas, characters, images, sounds, texts, bits of software code—that is available for practitioners to take and transform as they please, and into which they can return the fruits of their labours. As Raymond (2000) says, "Good programmers know what to write. Great ones know what to rewrite (and reuse)." Importantly, they need not "despoil" the commons like de Certeau's poachers (1984, loc. 2532) to do so.

Lawrence Lessig's term for the digital commons is "RW" (read-write) culture. His 2008 book *Remix* provides many examples of what this process looks like up close. "Remix," he writes, "is an essential act of RW creativity. It is the expression of a freedom to take 'the songs of the day or the old songs' and create with them" (2008, p. 56).[10] As Gu (2014) points out, Lessig's remix does not necessarily entail an "underlying reverence" for the material being remixed: "remix involves appropriation without implying fandom" (p. 138).

[10] Lessig is talking about cultural outputs in general, not just songs, but he is borrowing a quote from John Philip Sousa's 1906 testimony before U.S. Congress regarding the emergence of mechanical devices that could play musical recordings; see Lessig, 2008, p. 23-24.

The activities of the digital commons and RW culture are not the marginal practices of an underground counterculture producing esoteric postmodern art; they are at the heart of a transnational creative media ecology—especially in videogames, according to Dennis Redmond:

> Today's videogame commons is comprised of the non-commercial institutions, communities and practices of digital artists, videogame players, and videogame fan communities. Far from being a marginal phenomenon or a relic of the videogame industry's start-up era, the videogame commons has become an increasingly powerful and pervasive feature of videogame culture. In fact, today's videogame commons is radically democratizing nearly every aspect of digital media production, distribution, and consumption. (Redmond, 2014, p. 7)

It is through this videogame commons that media users are re-signifying and re-circulating their creative outputs.

2.3.2 Converging on co-creation

The influence of the commons has apparently arrived (or, some would argue, returned—see the end of this subsection) to the point that its yield is now being incorporated into the definition and design of commercial games. This process apparently began outside of gaming with Jenkins' "convergence culture" and producers' increasing dependence on fan activity in their business models. A classic example of contested convergence is seen in *Harry Potter* author J.K. Rowling's admission that she consulted the web-based, fan-written *Harry Potter Lexicon* to fact-check her further writing about her own fictional world (Punday, 2015, p. 145).

In videogame production, terms like "participatory" and "convergence" have given ground to "co-creation." As early as 2003, Morris advocated moving beyond the term "participatory media," stating that "multiplayer FPS games have become 'co-creative media'; neither developers nor players can be solely responsible for production of the final assemblage regarded as 'the game'" (Morris, 2003, p. 8). Eight years later, Banks (2011) argued that commercial videogames are increasingly produced through a symbiotic co-creative process between industry and players—including modders and non-modding fans who provide feedback during the development process, in

hopes of shaping its trajectory. Gamers, Banks argues, "increasingly expect and demand that developers will not only listen to their views, but also enter into active dialogue with them" (2013, p. 65). Nieborg (2005) has claimed that the relationship between industry and game modders tends to be harmonious:

> Modders are currently able to directly influence the game development industry on all levels, ranging from a technological influence to influencing (future) game design and supplying a workforce for the commercial game development industry. This symbiotic relationship is one of mutual respect and dependency. (Nieborg, 2005)

As for Minecraft, Redmond (2014) asserts that Notch's reliance on early-adopter feedback and suggestions during Minecraft development in 2009-2010 means that the game was ultimately co-created by hundreds of individuals in an example of "audience-led production." Indeed, my own thesis hinges on the claim that Minecraft cannot be understood as a cultural product separate from its fan-contributed mods: they are an intrinsic part of what the concept of "Minecraft" means.

On the other hand, numerous scholars have argued that audience-led production of games, including modding, is not new (Champion, 2012a). Rather, these activities have become notable in the present moment because they represent a break from three decades of commercialized, centralized game development, which has drawn a distinction in the popular imagination between game designer and game player where previously no such distinction existed. Sotamaa writes that "it may be problematic to assume that there actually ever was a moment when we were able to distinguish game producers from consumers with relative ease" (2009, p. 142). The first cultural objects to be considered computer games were themselves created through unstructured, incremental homebrew-style development: games like *Spacewar!* were passed around and played primarily among the hacker elite at universities and research institutions, and it was these same players who continually made and recirculated changes to the game code (Dyer-Witheford & de Peuter, 2009; Sihvonen, 2011; Christiansen, 2012). Digital games were co-creations of the commons in the very beginning. The dominance of opaque, gargantuan media conglomerates in game development was

not a timeless order that is now finally giving way in the Internet era, but a moment of history like any other—with a beginning as well as, possibly, an end.

2.3.3 Enclosure of the digital commons

"What Minecraft suggests," writes Redmond, "is that the videogame commons, in lockstep with the institutions of the larger digital commons, is beginning to break the chains of corporate advertising and corporate oligopoly over media production" (2014, p. 17). But there are those who caution that celebration of democratized culture and empowered consumers may be premature. It is not that corporate interests persist in trying to stamp out non-controllable, non-rationalizable grassroots creativity wherever it arises (although this has certainly been attempted at various times). Rather (the argument goes), even as they increasingly recognize the value of open culture and the contributions from the commons, corporations seek to appropriate and enclose that distributed creativity for their own profit. Murray (2004) is critical of the tendency in cultural studies towards the "valorization of fan agency" in uncritically celebrating the apparent new-found openness of media companies and their willingness to encourage fan agency (p. 21). She warns that these do not represent authentic new attitudes, but rather strategic calculations to enclose useful fan activity within the fences of capital:

> Stripped of its communitarian rhetoric, New Line's novel willingness to cultivate fan communities is merely a conditional agreement not to enforce its IP rights for the precise period during which fan activities further its commercial interests. Once fan behaviours cease to build business for rights holders, tacit permission for such 'fair comment' usage will almost certainly be revoked. The ultimate discretion of media conglomerates to shut down fan creations around corporate-owned properties renders hollow much cultural studies championing of fans' contrarian impulse. (Murray, 2004, p. 21)

The games industry's time-honoured tradition of clamping down on creativity in the first place is explicable in terms of the anxious aversion, under capital, to that which is unpredictable and uncontrollable—whether it be piracy, cheating, user production of illegal or morally objectionable content, or losing the ability to completely dictate brand identity (see, for example, Consalvo's

[2007] discussion of industry attempts to classify console mod chips as dangerous and illegal, or Dyer-Witheford and de Peuter's [2009] account of Blizzard Entertainment's police-state-like governance of *World of Warcraft*). Malaby (2009) argues that the rationalizing, bureaucratic logic of the games industry (as with corporate commerce in general) is oriented towards reducing and controlling such unpredictability as much as possible (p. 106).

However, *unpredictable outcomes* are precisely what drives innovation and enables emergent play, and video game capitalism therefore carves out a space for them. Dyer-Witheford and de Peuter (2009) describe how the autonomy of hacker "playbour" (2009, p. 23; see also Kücklich, 2005) drives innovation while at the same time capital attempts to "constrain this autonomy within the limits of profit" (Dyer-Witheford & de Peuter, 2009, p. 5); this struggle between autonomy and constraint are "often what drives capital forward to new horizons as it attempts to crush, or co-opt and capture, resistances" (p. 5). They speak of an ongoing "dance of capture and escape" (p. 27) by which innovation emancipates itself from capitalism and moves to the periphery of industry, only to be reappropriated by the commercial engine once again when it generates something of marketable value. The cycle of escape and capture is highly beneficial—in fact, crucial—to the commercial games industry: "Even in the commodity form . . . games have continued to depend for their vitality on a constant infusion of energies from a do-it-yourself player-producer culture that embodies the autonomous capacities of the new echelons of immaterial labor" (Dyer-Witheford & de Peuter, 2009, p. 6). Notoriously risk-averse companies (Kücklich, 2005), whose business models depend largely on the trading—rather than the synthesis—of intellectual property (Dyer-Witheford & de Peuter, 2009, p. 38), profit by allowing others to absorb the risks of experimentation (Sihvonen, 2011, p. 41). Thus large game publishers spin off and subsequently reabsorb small studios at which the labour environment exhibits a kind of playfully productive "working anarchy." In this way, they "increasingly determine when, where, and for how long the more anarchic enclaves will exist" (Dyer-Witheford & de Peuter, 2009, p. 42). Big business reaps the benefits of the micro-innovation

of small start-up companies, the majority of which fail spectacularly, "perishing only to provide an emergent industry with a critical mass of free creations from which a handful of winners could be picked" (p. 24). The game industry "has increasingly learned to suck up volunteer production as a source of innovation and profit" (p. 27). Kücklich (2005) illustrates how game developers have profited directly from modding labour, with reference to the *Half-Life* mod *Counter-Strike*. The license agreement for Half-Life stipulates that the developer, Valve Software, would maintain intellectual property rights for all mods created using their engine and development tools. As a result, when Valve transformed Counter-Strike into a commercial offering, they were spared the overhead marketing costs that would normally be required to establish a brand, because "this was [already] done for them by the creators and players of the game" (Kücklich, 2005). Modding further adds value to commercial games, and thus increases industry profits, by extending the shelf-life of the product (2005).

Thus a debate exists regarding whether modding empowers audiences to take determining roles in cultural production, or merely provides industry with a source of free labour and innovation to exploit. Because modding is "immaterial labour" (Dyer-Witheford & de Peuter, 2009), it is also invisible labour, readily framed as mere play—something that the labourers find valuable and *want* to do anyway—in order to absolve the beneficiaries of any responsibility to provide remuneration (Kücklich, 2005). In that case, modders would be doubly duped—not only paying to consume the commercial product, but also providing uncompensated labour towards the creation of that product. Even having claimed a "symbiotic relationship of mutual dependency and respect" between modders and industry, Nieborg admits some doubts about the authentic participatory nature of modding:

> The question is, is this process of co-creating media and participatory design still participatory culture? On the one hand it is, user[s] are still actively engaging with media texts out of free will and are eager to use (and create) free tools and software. On the other hand the question is whether these participatory and collaborative elements within

> FPS mod culture are as bottom-up as some argue it to be. In other words, are we witnessing the rise and subsequent fall of FPS mod culture based on an open-source ethos or are we at the beginning of a new era featuring cleverly commoditised user-created content? (Nieborg 2005)

It is a question that remains unresolved, as it continues to be played out in the political dramas of game development and modding. Murphy (2015) highlights contradictory and paradoxical aspects of Minecraft's supposedly open and democratic development, which he says actually "oscillated between the promotion of audience participation and the subsequent reassertion of developer control":

> While popular narrative framings [around Minecraft] emphasize harmonious audience-developer relations, the game's history is littered with tensions stemming from the official support of particular player activities and subsequent discouragement of others. (Murphy, 2015)

The latest such political drama in the Minecraft saga comes with Microsoft's nascent efforts to monetize user-created mods through a central marketplace (Newman, 2017).[11]

Considering the above, it seems that while capital is capable of capturing and exploiting particular enclaves of audience-led creativity, it is not able to enclose the entirety of the commons. Despite corporations having control over the creation of temporary "anarchic enclaves" for their own benefit, true indie blockbusters like Minecraft still manage to spring seemingly out of nowhere, and may (for a time) make more of a symbiotic-tactical, rather than exploitative-rationalized, use of commons-based peer production. As Redmond says, "while transnational media corporations remain influential, they are no longer the only game in town" (2014, p. 17).

[11] At least for the time being, this only applies to the console, mobile, and Windows 10 editions based on Microsoft's new "Bedrock" engine, and is beyond the scope of this research. The original PC edition, now rebranded Java Edition, retains an active user base and modder community that remains free to distribute mods on its own terms.

2.4 So… which of these things qualifies as modding?

Thus far, I have given a perfunctory definition of "mod" (Section 1.4), and in the above sections I have tended to discuss modding on the same terms as other fan creations and creative outputs of participatory culture. However, there is no one easy definition of modding. The practice and the academic literature both are rife with difficult-to-classify boundary cases that turn out to not really be on the boundary at all—they are central to discourses about what modding is as a practice. Nor are these discourses merely concerned with pedantic taxonomy: there are real cultural, financial, political, and legal stakes in the contested definition of modding, as in conflicting understandings of creativity and creative production more generally.

Games scholars have tended to treat modding as an extension of play, or as part of a continuum of play-like activities. Sihvonen (2011, p. 88) provides a "typology of modding" that organizes the play activities identified by Aarseth (1997), Raessens (2005), Alexander Knorr, and Nieborg and van der Graaf (2008). Sihvonen finds that these activities can generally be divided into those that are "game-provided" and those that are enabled by user extension. Aarseth's interpretive, explorative, and configurative functions, for instance, are all game-provided. Raessens and Sihvonen also speak in terms of interpretation and configuration or "reconfiguration," while Knorr characterizes the whole range of game-provided player activity as "taking into possession." Nieborg and van der Graaf identify game-provided activity as simply "play"—implying that the user-extended functions are something other than play.

Sihvonen herself proposes categories of game-provided interpretation and configuration, and user-extended reworking and redirection. "Mods" that operate through interpretation, configuration, or reworking can engage with either aesthetic or operational characteristics of the game (Sihvonen, 2011, p. 88). Aesthetic interpretation and configuration are exemplified by the "assembly and fabrication of game elements from a selection of existing parts" (p. 88). Operational interpretation includes discovering and exploiting the boundaries of play—e.g. taking advantage of glitches and

bugs. Using built-in cheat codes or developer commands are examples of operational configuration. Reworking means modifying or extending the game beyond its commercial-off-the-shelf (COTS) capacity—aesthetically, by editing textures, skins and 3D models, or operationally, by altering the rules of the game itself. Finally, redirection (which Sihvonen does not subdivide into aesthetic and operational modes) is the use of the game engine to create other out-of-game products or paratexts, such as screenshot albums and machinima videos.

Significantly, Sihvonen uses the term "modding" to describe *all* of these activities. The inclusion of game-provided player functions would seem to be at odds with her own initial definition of modding, which is

> various ways of extending and altering officially released computer games, their graphics, sounds and characters, with custom-produced content. Modding can also mean creating new game mechanics and new gameplay levels (maps) to the point where the original game transforms into a completely new title. (2011, p. 6)

Furthermore, "game modding in practice takes place on the level of altering and tinkering with game data files as the access to game engines is not normally granted to players" (2009, p. 131). The appearance of inconsistency itself need not trouble us overmuch: Sihvonen's definition evolves on an intentional trajectory throughout the course of the book, as practices that might be made invisible by a strict classification system are acknowledged as part of a continuum of active player creativity. For instance,

> Modding does not . . . constitute a simple category of 'constructing' or 'adding' new game elements to an existing game—it also includes fuzzy and incoherent practices such as taking advantage of bugs and glitches that have an effect on the game's functions and inner mechanics. (Sihvonen, 2009, p. 132)

However, choosing a custom avatar skin or exploiting a game glitch is not what comes to mind for most players when they think of "modding"—certainly not in Minecraft, anyway (recall Section 1.4) . The game-provided creative functions that Sihvonen describes are what makes games interesting in terms of their differences from other media, but it is user extension of software (code

and data) that makes *games like Minecraft interesting in terms of their differences from other games*. For this reason, there is value in limiting analysis of modding practice to user extension, as this study does.

Offering a typology of *mods as objects*, rather than *modding as an activity*, Scacchi (2011) identifies four categories of mod: user-interface customizations, conversions that alter game rules (including *total conversions* that transform the entire gameplay experience), machinima and art mods (grouped together as one category of artistic production), and the hacking of closed game systems (sometimes with the goal of tweaking or cheating, or as an end in itself). Leaving machinima out of the equation, Postigo (2007) classifies the "add-on" productions of "fan-programmers" (mostly in the FPS scene) in three categories: mods, maps, and skins—a distinction that is helpful in further analyzing Minecraft modding. Christiansen (2014) draws attention to the ambiguous status of player-made objects when he consistently refers to Minecraft's "Skyblock" mini-game as a mod, while admitting that "the single player version would more accurately be called a custom map, as there are no new mechanics *per se*" (p. 23).[12] Skyblock may fit with other redirections of Minecraft gameplay that Duncan (2011) identifies as examples of Gee and Hayes' "soft modding" (2009).

Although some kinds modding can only be accomplished through hacking—for example, reverse-engineering proprietary data formats, or injecting new instructions directly into a program's binary code—researchers are keen to highlight instances of commercial developers enabling modding. Kelland (2011) points to id Software's decision in developing *DOOM* (1993) to separate asset data (level layouts, 3D models, texture graphics, etc.) from process code as an early example of industry enabling and encouraging fans to tinker with their product. Id's *Quake* series of multiplayer first-person shooter (FPS) games is credited with affording the kind of configurative potential necessary to give rise to the first examples of machinima (Kelland, 2011). Other writers have pointed

[12] In Skyblock, the player appears on a small, isolated dirt island in the sky, and has to carefully manage limited resources and restricted space. The single-player version can be made using an external map editor to create the terrain setup. The save-game files can then be shared, with the recipient dropping them into the appropriate folder to enable play in vanilla Minecraft. In the multiplayer version, a new modded rule is required to override Minecraft's normal semi-random player-spawn locations and ensure that players are properly placed on the islands.

to the more recent trend of developers releasing modding toolkits alongside blockbuster games (Nieborg, 2005; Dyer-Witheford & de Peuter, 2009, p. 25; Champion, 2012a). These kits—notably absent in the case of Minecraft—feature many of the same tools of the trade that were used in the commercial development of the game.[13] Such tools have a complicated status and are not always embraced by everyone. Banks (2013) describes how, in the case of Auran Games' *Dark Reign*, the game engine was developed before the game itself, with the goal of providing an extensible framework that both Auran's programmers and fan modders could work from. In Banks' account, this approach is seen to have generated enthusiasm in the player community, while causing considerable anxiety and frustration for Auran's designers who often felt that the requirements of the engine were getting in the way of crafting the gameplay proper. This foreshadows a tension seen in the Minecraft modding community, between developing with a strategically-planned, extensible framework, versus making simple, ad-hoc, low-level edits that get the job done faster but may frustrate future efforts at extension. Minecraft is also notable for having a vibrant modding scene despite its *lack* of the sorts of official editing tools that have enabled prolific modding for other games, like Valve's Source SDK and Bethesda's Skyrim Creation Kit.

2.4.1 Why mod anyway?

Motivations for modding are manifold. Sotamaa cautions against the assumption that there is such a thing as an "average" modder or a typical motivation, pointing to "a spectrum of contradictory modder identities and the different contexts that govern the production of them (2010, p. 252). In his study on Operation Flashpoint modders, he finds that they could be partitioned based on what sort of mods they liked to focus on (new missions, add-ons, large-scale conversions), and identifies hacking, self-expression, community-building, and seeking a professional career in game development as personal motivations.

[13] Some prominent examples of these tools include BioWare's Aurora Toolset for creating custom content in *Neverwinter Nights* (Wardrip-Fruin, 2009; Champion, 2012b), Valve's Source SDK for *Half-Life 2* mods, and Bethesda's Elder Scrolls Construction Set (Fassbender, 2012) along with its various "Creation Kit"-branded successors.

Nieborg (2005) and Champion (2012a) both document how modding can serve as a gateway to career success: for enthusiasts with limited resources hoping to become employed in the industry, it provides a way to build a design portfolio and get noticed by established developers. It also provides a means of honing game design skills in an experimental and relatively low-risk environment, and can serve as an effective teaching tool in university-level game design classes (Ashton, 2010; Champion, 2012a; 2012b). Kringiel (2011) argues that making mods and machinima improves one's "game literacy"—a subset of a more general set of competencies, termed "procedural literacy" by Bogost (2005), that are concerned with the ability to decipher the messages produced by computing machines and understand the underlying processes that govern that production.

Another motivation for modding identified in the literature is simply the player's desire for autonomous self-determination, the ability to reconfigure or improve the game according to one's preferences. Jenkins' television fans could revise and expand the canonical narratives through video editing, art work, and fan fiction, but these practices are both painstaking and marginalized, framed by the dominant discourse as foreign material loosely attached to the core text. For conventional media, the re-signification of reinterpretation through the production of fan works is something that happens *after the fact*. The reconfiguration of a digital game is the transformation of the product itself. Champion writes that "designing a game mod is one of the few fields of criticism where one can design and test an alternative to the current offering from the comfort of an armchair" (2012a, p. 15). The same sentiment is expressed in Joubert's observation that "since time immemorial, players have looked at the games they've played and decided that they could do one better" (2009). Nieborg identifies this as one of the reasons for the reduced tension between modders and developers, because "if you don't like the game, [you can] simply mod it!" (2005).

2.5 Research on Minecraft

To date, the majority of Minecraft research has been concerned with education or child psychology. A smaller but vibrant thread of scholarship explores the game's creative affordances, and the participatory nature of its play and development.

2.5.1 A tool for teachers

Short (2012) describes how the configurative affordances of Minecraft's building block structure enable teachers to create a variety of builds for immersive lessons in biology, ecology, physics, chemistry, geometry, mathematics, and geography. These practices, Short writes, are already being used by educators. Short advocates the utility of the MinecraftEdu mod (now superseded by Minecraft: Education Edition, which is published by Microsoft and branded as an "edition" rather than a mod—see Section 1.3). However, the teaching affordances he describes arise from Minecraft's core mechanics. Short does not specifically describe the added utility contributed by MinecraftEdu, which is in the tools it provides to facilitate classroom communication and the teacher's ability to track and manage multiple students (it also provides a means by which Minecraft can be affordably licensed for classroom use).

Several scholars discuss Minecraft's utility for developing digital and media literacy skills in youth (Macron and Faulkner, 2016; Abrams, 2017; Elliot, 2018; Willett, 2018; Dezuanni, 2018). Schifter and Cipollone (2015) suggest that Minecraft is what Seymour Papert calls a "transitional object" in constructionist pedagogy, in which learners as players are able to "experiment with knowledge in meaningful contexts" (p. 213).

Brand, de Byl, Knight, and Hooper (2014) describe their use of Minecraft as a 3D Virtual Learning Environment for constructivist teaching in a university classroom, claiming that it is highly suitable for constructivist learning goals (p. 60-61). Furthermore, it affords "multiple opportunities for play-learning" by enabling both the *ludus* (structured and role-bound) and *paidia* (spontaneous and free) modes of play identified by Caillois (1961). Notably, Brand *et al.* assert that Minecraft's

blocks serve as what Seymour Papert calls objects-to-think-with (Brand *et al.*, 2014, p. 61). For Callaghan (2016), the MinecraftEdu mod is yet another example of "edugames" that can enhance student learning in school settings, particularly due to affordances for collaboration and "the creation of authentic tasks" (p. 253). Kuhn (2018), in his review of MinecraftEdu's successor, Minecraft: Education Edition, says that the whole game is an object-to-think-with in which "the game expects players to learn through trial and error . . . there is little expectation for getting it right the first time" (p. 217). Further discussion of the utility of Minecraft in progressive—largely constructivist/constructionist—pedagogies is found in Fanning and Mir (2014), who situate Minecraft in a long historical tradition of construction toys that were framed as providing children with wholesome play experiences with key psychodevelopmental benefits.

To be effective, the use of Minecraft in educational settings needs buy-in from teachers, and its integration is not always seamless. For Ellison and Evans (2016), "there appears to be a disconnect between some teachers' and parents' understandings about the Minecraft world's mechanisms, uses, and benefits" (p. 25). Hanghøj and Hautopp (2016) note variation in the effectiveness of Minecraft as a teaching tool based on teachers' level of game literacy and their attitudes towards incorporating Minecraft into lesson plans. Kuhn (2018) has similarly identified a tension between the curriculum-centric culture of schools and the open-ended potential of blank-slate lesson-building tools and constructivist learning spaces like Minecraft: "Minecraft has been a more challenging classroom fit for educators due to its lack of content. The game has required educators to use it less as a text for content delivery . . . and more of a tool for constructivist learning and content creation" (p. 218). Nebel, Schneider, and Rey (2016) note the success of games and mods that were specifically designed with pedagogical research in mind, but acknowledge that the technical barriers to engaging in such research are quite high due to researchers not having the skills or resources to develop or modify games. Based on their review of additional research, Nebel *et al.* further caution that *too much* buy-in on the part of students can spell trouble: players who are already

experts may fall back on pre-existing behaviours rather than experimenting creatively, and experienced players may dominate in collaborative sessions (p. 360).

2.5.2 Minecraft in children's social and psychological development

Given the common perception of Minecraft as a game for children, and the intense interest from educators, it is not surprising that it has also attracted attention from those interested in children's social and psychological development outside of the context of schooling.

Pauls (2017) uses Minecraft to interrogate Parten's theory of social play, finding that all six of Parten's categories (non-play, onlooker activity, solitary, parallel, associative, and co-operative) were observable in children's Minecraft play. Pellicone and Ahn (2018) track how a young Minecraft server administrator engages in "the design of sociotechnical systems" across multiple "affinity spaces" (p. 455). Voiskounskya, Yermolova, Yagolkovskiy, and Khromova (2017) designed a Minecraft-based experiment to study child creativity in individual and dyadic sessions. This study also points to the potential uses of Minecraft as a research platform for designing and carrying out behavioural experiments. Finally, Mavoa, Carter, and Gibbs (2018) urge researchers to focus less on measuring children's "screen time" with digital games, and instead collect descriptive information of what children are actually doing during play. They indicate that Minecraft is the most popular digital game for children aged three to 12 years, and that most of them play on tablet devices.

2.5.3 Minecraft in play

Minecraft's open-ended sandbox environment gives rise to a wide range of creative play possibilities, which games scholars have been eager to explore and catalogue. Goetz (2012) understands Minecraft's play dynamics in terms of its compelling "tether and accretions" fantasies,[14] which are discussed further in Subsection 4.3.1. Canossa (2012) found a range of play styles and

[14] Goetz uses Minecraft as part of his larger argument as to the utility of "fantasy" as a concept for uniting gameplay and story in formal analysis. Gameplay and story are the two supposedly-distinct elements of analysis that are emphasized by the two solitudes of early (c. 1995-2010) game studies: narratology and ludology. Goetz is placing the nucleus of what distinguishes games from other media forms within the psychodynamic fantasy that connects the two.

motivations in Minecraft, some of which were correlated with conventional personality metrics. Brackin (2012), Keogh (2012), and Thomét (2014) have taken different approaches to exploring how Minecraft play structures, and is structured by, storytelling within and about the game. Addressing the fact that a substantial portion of Minecraft storytelling takes place on YouTube, Gu (2014) studies the processes by which video creators appropriate Minecraft material to create video/machinima fan fictions, while MacCallum-Stewart (2014) explores the popular YogsCast "Let's Play" Minecraft videos in terms of storytelling and games criticism. Niemeyer and Gerber (2015) argue that Minecraft video commentaries and walkthroughs should be understood as serious objects of study in research on maker cultures.

Bull (2014) criticizes Minecraft for making an unsuccessful attempt to deconstruct gender, noting that its implied (and procedurally overdetermined) frontier-settler motif cannot escape the gendered histories that inspire it. Minecraft's broader entanglement in colonialist fantasies is addressed further in Section 4.3. However, Phillips (2014) offers a counterpoint to this perspective, arguing that "Minecraft conceals within its thoroughly exploitable mathematical models queer possibilities of reproduction, temporality, and occupation of space that move beyond the purposes for which they have been coopted" (p. 106).

Several authors have considered the kinds of work that are implicated in Minecraft gameplay. Petry (2018) studied how children situated their Minecraft activities in terms of play, work, labour, and leisure, finding that "the same activity can change its meaning, depending on the intention behind it . . . almost all children affirmed that 'work' was a nice word to use when they were talking about Minecraft" (p. 13). He argues that his empirical results reinforce the need to distinguish "work" from "labour." Others have described various types of specialist work that arise in Minecraft in ways that mimic professional activities. Apperley (2015) sees a "post-digital" artistic curation—one that self-consciously draws attention to algorithmic malfunction—in the practices of players who document and share their encounters with software glitches in Minecraft. I have previously argued that Minecraft players engage in a process of "procedural elaboration" in which

"significant work is invested in discovering esoteric details" about the game's inner-workings, both as a coping mechanism for the procedural inscrutability of an early-access game always in flux, and as a means of articulating cultural capital (Watson, 2017). Simon, Wershler, and Watson (2015) have described how, in the context of a dizzying proliferation of technology mods with competing ways of algorithmically representing the concept of electrical energy, modders—and players conversant in the selection and curation of mods for compatibility—position themselves as expert elites akin to the electrical experts of the late industrial revolution.

Finally, Schneier and Taylor (2018) draw attention to how different Minecraft editions (defined in Section 1.3) lend themselves to different modes of play. MCPE/Bedrock variants target tablet and handheld devices (in addition to Windows PCs and certain consoles), and are consequently more likely than other editions to be played on touch-screens. The authors found evidence of space-time biases (as originally defined by Harold Innis) in young people's Minecraft play practices, wherein play on PC or game console tended towards the time-biased "monumentary"—long-lasting grand structures, strong tethering to specific in-game locations, and repeated play in the same Minecraft world—while play on mobile devices with touch-screens exemplified the space-biased "momentary"—nomadic exploration, survivalist fantasies, and ephemeral worlds that were routinely deleted and replaced. This is significant because it shows a divergence of play practices between platforms, before even accounting for how Java Edition play on PC is shaped by modding. Furthermore, if we accept Schneier and Taylor's claim, it implies the balance of Minecraft modding is in service of the monumentary—large, complex industrial and magical facilities especially—because that is the bias of the PC-centric Java Edition.[15]

[15] To be clear, Bedrock edition is not exclusive to mobile devices, and would therefore support both time- and space-biased play practices, depending on where it was played. At the time of Schneier and Taylor's study, the Better Together update and Bedrock convergence had not yet occurred. The authors were thus contrasting earlier versions of MCPE (circa 2015) with the contemporary Java variant (which they and others called the "PC version" at the time) and the PS4 edition (now classified as a Legacy Console Edition).

2.5.4 Produsage, platform rhetoric, and Minecraft exceptionalism

A significant body of scholarship is concerned with trying to pinpoint what exactly makes Minecraft special: what accounts for its resounding success, or how it may be a "game-changer" in media production. This Minecraft exceptionalism comes in many forms, identifying different aspects of the game as definitive—although most highlight open-ended play, co-creative development, and extensibility (modability) as decisive. Scholars should treat claims about the uniqueness or exceptional nature of any media artifact with a healthy skepticism, but not necessarily outright dismissal. Indeed, my own rationale for this research rests on some qualified claims of Minecraft exceptionalism.

Gerrelts (2014) introduces the anthology *Understanding Minecraft: Essays on play, communities, and possibilities* by claiming that "Minecraft has engaged the popular imagination so profoundly that it has transformed videogame culture," likening its proliferation and enthusiastic adoption to the "Pac-Man Fever" of the 1980s (p. 1). He further links its success to the fact that "Mojang generously allows users to modify the game and share derivative works" and to the disruption of producer/consumer distinction that is characteristic of participatory culture as discussed in Section 2.3:

> Unlike other videogames that are produced by teams of programmers working largely in isolation from their audience, *Minecraft* has been shaped collaboratively with the help of the global gaming community. (2014, p. 1)

This sort of participatory development is commonly cited as the thing that makes Minecraft tick. Christiansen (2014) argues that the "mechanics of the game shape the behavior of the player, configuring her not only as survivalist or an adventurer, but as a builder, a programmer, and a hacker," resulting in an "almost seamless continuum between player, modder, and developer" (2014, p. 34). Redmond (2014), who was introduced in Section 2.3, sees Minecraft as the inflection point in a transition towards a new "videogame commons" production model. For Lastowka (2013), it is no accident that Minecraft showed up in the era of YouTube, social media, metadata

crowdsourcing, and the other forms of user-generated content that are the hallmark of Web 2.0, of which, he argues, Minecraft has been both a benefactor and a beneficiary. Re-iterating that "amateur creativity" has been key to Minecraft's success, Lastowka notes that the current intellectual property regime discourages and disincentivizes this sort of participatory production model. Schlinsog (2013) expresses a similar concern, arguing that merely granting contractual permission for users to make mods (as Mojang does) is not sufficient to protect participatory creativity,[16] and copyright law must expand to recognize user-generated content as a new form of authorship if the Minecraft-like production model is to endure: this is not only a legal issue, but a cultural one as well, because copyright law can shape how we perceive authorship (Schlinsog, 2013, p. 206).

Leavitt (2013) describes "a verifiable ecosystem of player-creators" that "creates *alongside* the official production process, making Minecraft possibly one of the largest and broadest participatory cultural productions ever" (p. 2; emphasis original). Importantly, Leavitt argues that the vision of the "alpha artist" (i.e. Notch/Mojang) continues to matter to participatory contributors: rather than redirecting the game entirely according to their own whims, they recognize "experiential and creative boundaries based on the core game produced by Mojang's developers" (p. 26). Murphy (2015) asserts that fans' freedom to participate in creating Minecraft is fraught with tension. Boundaries pertaining to the alpha artist's priorities are not merely recognized and respected by fans, but are actively re-imposed by a company periodically trying to claw back some creative control over its product. Criticizing the "popular narrative framings" that "emphasize harmonious audience-developer relations," Murphy claims that:

> [Minecraft's] production oscillated between the promotion of audience participation and the subsequent reassertion of developer control. Far from being a simple tale of web 2.0 entrepreneurialism, Minecraft's unique rise in popularity is inherently linked to paradoxical notions of freedom and openness. (2015)

[16] For one thing, Schlinsog argues, US federal copyright law can pre-empt the enforcement of contracts under state law, and provides virtually no protection for creators of derivative works.

Another thread of Minecraft exceptionalism considers how the game can act as a techno-cultural substrate for further computation and culture-work. Duncan (2011) writes that Minecraft serves as an "experiential platform" that "works to provide players with experiences that are somehow 'about' something other than the game's presumed original intent" (p. 17-18). Leavitt, and the players he interviews, also speak of Minecraft as a platform, "not only literally as software but also metaphorically as a foundation on which participants build ancillary media" (2013, p. 26). Tremblay, Colangelo, and Brown (2014) do not mention "platforms," but appear to be addressing the same phenomenon:

> Notch is not the creator of an art object which can be analyzed like a work of literature, but is instead *the creator of the vocabulary and grammar* through which a player may articulate the game of *Minecraft* itself.... What separates *Minecraft* from games like *Mass Effect* or *World of Warcraft* is that the language created in *Minecraft*'s development is open and unrestricted enough to allow for the kind of "decentralized creativity" which has made it notable. (2014, p. 77; emphasis original)

For Simon and Wershler (2018), the fruits of Minecraft are not only computational and experiential, but allegorical and evocative too. The fundamental operational mechanism of Minecraft, they argue, is *not* the block but rather the *grid*, a cultural technique that serves as "the condition of possibility for the integration of aesthetics, governmentality and computational logic" (2018, p. 290). For its part, "Minecraft embodies that integration, and presents us with a powerful allegory for how it functions in 21st century culture" (p. 290).

2.6 What's missing

The scholars discussed in the preceding sections have considered mods as cultural objects, explored how and why people make mods and how and why the industry enables them, and explicated the status of mods as sites of reconfiguration and co-authorship within a broader play

ecosystem.[17] Surprisingly little, however, is said about the inner cultural life of modders—who they are as people, and how they communicate and work together (or fail to). The conflicts that arise among modders are as important to study as those that arise between modders and industry, but much more attention has been given to the latter in existing scholarship, which on occasion has a tendency to treat modders as a homogeneous group. Morris (2003), Sotamaa (2010), and Leavitt (2013) are among the few who have taken modders *themselves* as objects of study—individually or in small groups, rather than as a uniform mass. One of the aims of this work is to add further modder-centric research to this small but growing collection.

There is also a paucity of studies attempting to engage directly with the technical dimensions of modding. Although many authors have touched on how software architecture overall can enable or constrain modding (e.g. Malaby [2009], Kelland [2011], Banks [2013]), little is said about how and why the technology works as it does. My intention in these pages is not to describe technology for technology's sake, but to show how modders' social acts and cultural practices are tied to the particular ways in which they navigate these idiosyncratic technological landscapes.

Finally, notions of strategies, tactics, and textual poaching, as they are currently understood in fan studies, may not fully capture the nature of modding activity. I argue for rethinking these categories in the context of modding.

[17] This term, which I borrow from Pearce (2009), refers to a complex of games, paratexts, virtual worlds, online communities, players, player activities, individual game play-throughs, developers, and publishers. It recognizes that different games, gaming communities, and player activities are not neatly partitioned off from one another, and that all of these actors are involved in constructing the meaning of gameplay.

III. A Workbench of Theory and Method

3.1 Rationale

There is a need to re-think some of the ways that modding—and Minecraft—have been discussed in prior scholarship. The focus of this research is on the *actions of and interactions between modders*, and it rests on the fundamental premise that how a modder participates in game culture as a fan, a player, and a creative practitioner must be understood in terms of how other modders do the same, and what opportunities or restrictions they create for each other in the process. The relationship between modders and commercial developers is secondary, although it will appear on occasion as a means of providing context for how modder practice has developed.

3.2 Research questions

Each of these research questions corresponds to one of the three "key themes" outlined in Section 3.3 below, where they are further developed and theorized in light of my findings.

RQ1. Are the categories of tactics, strategies, and textual poaching applicable to modder activity? How so/not?

RQ2. Has a consensus emerged around standard ways to mod Minecraft? Has Minecraft modding been professionalized or converged with commercial development practices?

RQ3. Are modding practices contested? What sort of strategies and tactics do modders deploy in discourse around modding, and how does gaming capital (Consalvo, 2007) circulate through these discourses?

3.3 Key themes

3.3.1 Tactics and strategies, poaching and settling

Throughout my analysis, I cast modder actions in terms of de Certeau's (1984) concepts of *strategy* and *tactics*, in direct response to prior work that has done the same for other fan activities (e.g. Jenkins, 1992). It is worth reviewing de Certeau's definitions here:

> I call a "strategy" the calculus of force-relationships which becomes possible when a subject of will and power (a proprietor, an enterprise, a city, a scientific institution) can be isolated from an "environment." A strategy assumes a place that can be circumscribed as proper (*propre*) and thus serve as the basis for generating relations with an exterior distinct from it (competitors, adversaries, "clienteles," "targets," or "objects" of research). (1984, loc. 142)

A tactic, on the other hand, is…

> … a calculus which cannot count on a "proper" (a spatial or institutional localization), nor thus on a borderline distinguishing the other as a visible totality. The place of a tactic belongs to the other. ... It has at its disposal no base where it can capitalize on its advantages, prepare its expansions, and secure independence with respect to circumstances. The "proper" is a victory of space over time. On the contrary, because it does not have a place, a tactic depends on time—it is always on the watch for opportunities that must be seized "on the wing." Whatever it wins, it does not keep. It must constantly manipulate events in order to turn them into "opportunities." The weak must continually turn to their own ends forces alien to them. (1984, loc. 144)

Although the tendency has been to cast fan activity as creative, tactical resistance to capitalist media strategies, the Minecraft modding scene makes it apparent that fans develop strategies too. This goes hand-in-hand with my conviction that de Certeau's "poaching" metaphor, applied to television fandom by Jenkins, falls short of describing what modders are actually doing. With modding, fans are not living nomadically on the periphery, stealing textual resources to make invisible and ephemeral remixes. They are moving to the center and settling down in the text. At their most powerless they are squatters, asserting squatters' rights; at the other end of the power

scale, they are landed immigrants and permanent residents. The "settling" process has tactical elements, but is also replete with strategic action. In fact, settling is a process of oscillation between strategy and tactics. Thus Minecraft modding is a "settling" in the sense of a colonization, but also in the sense of the gradual calming of a turbulent fluid.

To complicate matters, we will see in later chapters that actors occasionally make claims to tactical orientations from positions of strategy, or vice-versa. It may be helpful, therefore, to think of strategies and tactics as things that are always becoming, but never finished. In particular, when the prescriptions, intentions, or actions of a modder or developer appear to embody a strategy, we may actually be looking at a *fantasy* about a strategy. A historical example of this dynamic in play comes from Bernhard Siegert's description of the gridded layout of Spanish colonial towns in the 1500s, which "was not planned and built on the basis of the actual number of settlers, or as a means of distributing property, but with a settlement fantasy in mind" (2015, p. 107).[18]

Chapter 4, Settling Minecraft, lays out the case for the settling metaphor and for understanding Minecraft modding in terms of oscillations between tactics and strategies. Chapter 5, Better Than Minecamp: An unconventional convention takes a closer look at what settled Minecraft looks like, based on participant-observation activities, and finds that there are still key ways in which it remains "unsettled."

3.3.2 Rationalization and operational logics

The specific form and trend of strategic action in modding is *rationalization*, through which specific, sanctioned *ways* of modding emerge, which are self- and co-regulated within the community. Rationalization is one way to account for the convergence of modder and developer practice. Gallagher, Jong, and Sinervo (2017) previously observed this convergence dynamic in the case of *Skyrim* modding, but a form of it is present in Minecraft as well.

[18] I am interested here in the general idea that strategies are fantasies at their core. However, for a take on how Siegert's ideas of grid-based fantasy apply to Minecraft play, see Simon and Wershler (2018).

I rely here on Max Weber's (1930) concept of rationalization as the driving force in modernity's principle instrument of power: bureaucratic authority. For Weber, rationalization is the tendency for traditional values and emotional motivations to be replaced by rationally planned, calculated, results-focused prescriptions. Other authors have expanded on Weber's initial critique: Ritzer, for instance, identifies four characteristics of rationalization in contemporary consumerism: efficiency, calculability, predictability, and control (1983). Of interest here is the way that *processes* are subject to rationalization; for modding, that means both the computational processes carried out by the machine, and the design/practice processes carried out by mod developers.

At a finer granularity, I take the constituents of rationalized process to be what Wardrip-Fruin calls operational logics which, in games, are "the fundamental abstract operations—with effective interpretations available to both authors and players—that determine the state evolution of the system and underwrite the gameplay" (Mateas and Wardrip-Fruin, 2009, p. 1; see also Wardrip-Fruin, 2009). In Minecraft, the spatial logic by which blocks made of specific "materials" are slotted into specific locations in a rectilinear three-dimensional grid is an example of an operational logic. The operational logics of the game system actually impinge on modders in ways that alter their practice, while at the same time those practices are shaped (or perhaps justified *post-facto*) by external "professionalization" pressures such as notions about what makes for "best practices" in programming. I suggest that, since the game program's operational logics are already entangled with the processes of making mods, we can expand the concept of operational logics to cover these strategically-conceived units of rationalized action in the activities that modders carry out. Chapter 6, Operational Logics and the Rationalization of Modding, provides specific examples of these dynamics at work.

3.3.3 Modder discourse and gaming capital

Rationalized processes do not appear out of thin air, nor do they succeed in quietly ushering in a new, unchallenged order. Instead, they coalesce out of, and are refined through, social

interactions—i.e. *discourse*—between modders. These interactions take place in online message boards and chatrooms, on game servers, on Twitter and YouTube, in the margins of GitHub source code repositories, and in the design of mods and modding tools themselves (more on this last point, below). In illustrating how this discourse works, I turn to Mia Consalvo's concept of "gaming capital", a reworking of Bourdieu's cultural capital. Consalvo explains:

> Being a member of game culture is about more than playing games or even playing them well. It's being knowledgeable about game releases and secrets, and passing that information on to others. It's having opinions about which game magazines are better and the best sites for walkthroughs on the Internet. (Consalvo, 2007, loc. 220-222)

Different valences of the word "capital" must be distinguished. Following Malaby (2006), this document uses the terms *gaming capital* and *cultural capital* in the specific sense of *resources for action*: "In this sense, capital is always potential, something that is enacted only in the act of applying, invoking, using, or spending" (Malaby, 2006, p. 146). Without qualifiers, in the sense used by political economists (e.g. in the writings of Dyer-Witheford and de Peuter discussed in Chapter 2), "capital" is nothing less than the operational logic on which the strategic apparatus of contemporary capitalism is built.

Gaming capital is easily recognized in the world of modder discourse, where its circulation is precisely what shapes emerging ideas about the "proper" ways to program, mod, or design gameplay experiences. It is the currency with which meanings are contested among modders. Such contestations can argue both *for* (laying claim to) and *from* positions of tactics and strategy alike.

Source code, software architecture, and game rules make a reappearance in this branch of my analysis, as they play a major role in those same contestations of meaning. Dialogue is not the sole province of message boards and chat rooms. It is also present, though often concealed, in computational and practical process as well.

All of these issues pertaining to modder discourse and the circulation of gaming capital are explored further in Chapter 7, Modder Discourses.

3.4 Other theoretical tools

3.4.1 Productive play

Although game studies often traces its origins back to Caillois (1961), his initial claim that play is free, non-productive, and separate from the rest of life has been questioned by cultural anthropologists (e.g. Stevens, 1980) and game studies scholars alike. Pearce (2006; 2009) argues that play can be productive, observing that play-like activities often serve to build objects that have "serious" cultural and social value.

Modding, being a kind of game design, the creation of a cultural text, and the production of something of value, is clearly a kind of culture-work. Yet since modding has often been theorized as an extension of play (as discussed in Chapter 2), and since this categorization is not apolitical, it is necessary to examine the relationship between modding and play through a theoretical lens that allows work and play to occupy the same space. However, in Chapter 5, Better Than Minecamp: An unconventional convention, the relationship appears to be reversed, with *play* appearing as an extension of *modding*. This productive play, tactical in character, drives the project of settling Minecraft to new horizons.

3.4.2 Remix and hacker culture

This subsection actually deals with an interpretive framework that looks promising, but *doesn't quite fit* in the end. I mention it here because the ways in which it doesn't fit are of interest in light of my core arguments. While modders may engage in computer hacking, and while some may identify as hackers based on their broader interests and activities, being a modder is not the same thing as being a hacker—nor is game modding quite like its terminological antecedent, the aftermarket modding of cars.

The *hacker ethic* is a particular philosophical orientation towards software systems. Himanen (2001) describes the hacker as a specific personality that celebrates creative autonomy and a passionate, playful approach to meaningful work. Hackers strive to be self-sufficient, not reliant on

the products of commercial empire—or at least not confounded by their idiosyncrasies. They tend to feel a sense of social responsibility and may orient their work towards the common good, but must always remain a world apart from "ordinary" labour. They are most in their element when pushing at the boundaries of what is possible, and tend to lose interest and move on when the task changes from making something possible to making it practical and popular. Wark (2004) stresses the autonomy of the hacker even more strongly, writing that "Hackers use their knowledge and their wits to maintain their autonomy. . . . Hackers are not joiners. We're not often willing to submerge our singularity" (para. 005-006).

Modding, hacking, and the open-source ethos are closely related to the cross-media concept of *remix*. To remix is to use components of existing media products to create a new product. Sanjek (1994/2001) points to a strong link between hip-hop remixing and modding in the physical, pre-digital sense of the word—the mechanical and aesthetic personalization of low-rider cars. He suggests that modding and remixing alike are motivated by "a longstanding practice for consumers to customize their commodities, command their use and meaning" (p. 243). He goes on to argue that the widespread use of sampling in music production undermines the mystique of the autonomous creator who produces something from nothing—a mystique that is backed by commercial interests and copyright law. Lessig (2008) claims that we live in a "hybrid economy" that mixes Read-Only Culture (commercially produced and sanctioned products) and Re-Writable Culture (whatever is modded, remixed, or reappropriated, and is available as material for further remixing), and advocates for wider acceptance of this paradigm. Raymond (2000) argues that some of the most efficient and effective software development takes place in "the bazaar" of open-source, participatory software development and trade, rather than in "the cathedral" of carefully sanctioned, top-down planning and monolithic solutions (strategy).

This close relationship between hacking, remix, and modding would seem to suggest that Minecraft modders might be understood as hackers and remixers, but these comparisons do not

adequately capture the extent to which modders have constructed *institutions* of modding, and how individual Minecraft modders have, over time, become less autonomous and more strongly attached to a network of social relations that make the practice possible. In other words, hacking and remixing, as imagined by the thinkers cited above, is largely about tactics and thus cannot account for just how strategic game modding can be. The inadequacy of the hacker/remix frame is worth mentioning because the specific way that it falls short highlights the very aspects of modding that I am emphasizing in these pages.

3.5 Methodological orientation

In analyzing the body of data I have collected, my general approach has been modeled after the Extended Case Method (ECM), with shades of grounded theory (Glaser & Strauss, 1967; Corbin & Strauss, 2015), offering a means of further extending theory from observations.

In the grounded theory approach commonly used in cultural anthropology, as introduced by Glaser and Strauss (1967), the researcher does not start out with a hypothesis or a specific theory to be evaluated. Grounded theory begins with unstructured or semi-structured data collection and open coding. Overarching theoretical concepts are suggested by the data themselves during coding. The process is inductive, allowing theory to build or coalesce from the data, leading to conclusions that are "grounded" in observed realities.

The principle activity of grounded theory is ethnography. According to Christine Hine, "the distinctive nature of the claim to being ethnographic . . . is that [the] authors aim to study enduring practices through which the community becomes meaningful and perceptible to participants" (2000, p. 21). Grounded theory does not have a monopoly on ethnographic methods. In his description of ECM, Michael Burawoy calls for applying "reflexive science to ethnography in order to extract the general from the unique, to move from the 'micro' to the 'macro,' and to connect the present to the past in anticipation of the future, all by building on preexisting theory" (Burawoy, 1998, p. 5). The key difference is "elaborating" or even "reconstructing" existing theory, rather than "discovering"

grounded theory (1998, p. 16). Burawoy explains what distinguishes a meaningful, successful reconstruction of theory:

> [W]e seek reconstructions that leave core postulates intact, that do as well as the preexisting theory upon which they are built, and that absorb anomalies with parsimony, offering novel angles of vision. Finally, reconstructions should lead to surprising predictions, some of which are corroborated. (1998, p. 16)

Furthermore, ECM ascribes particular value to the diachronic: the method is most productive when "extended" temporally. Max Gluckman, who developed the original form of ECM, advocated "taking a series of specific incidents affecting the same persons or groups, through a long period of time, and showing how these incidents, these cases, are related to the development and change of social relations among these persons or groups" (1961/2006, p. 17). Burawoy's interpretation further suggests that extending across multiple *different* cases enriches the reconstruction of theory. (Not every researcher is in a position to perform multiple independent case studies elaborating on the same theory,[19] but in my view the different cases need not necessarily be investigated by the same individual.)

As for my research, RQ1 in particular is an application of the extended case method. My initial encounter with Minecraft modding was informed by my prior knowledge of de Certeau's theories and Jenkins' reconstructions thereof. "Settling" is the extension I bring to this paradigm of audience activity, based on my findings. My answers to RQ2 illustrate the connections between de Certeau's strategies and Max Weber's theories of rationalization. In addressing RQ3 (largely in Chapter 7), I illustrate how Consalvo's concept of gaming capital is applied to modders as a subset of gamers (and also in a context in which all of the relevant paratexts are digital rather than physical). I also include a brief consideration of the relations between gaming capital and the multiple inflections of Bourdieu's original cultural capital concept, based on my data.

[19] For instance, I am building on Jenkins' (1992) application of de Certeau's theories to television fandom communities. When that work was published, I was five years old, and my comprehension of qualitative research methods was a little shaky.

Grounded theory comes into my method where I have encountered data that spoke of theoretical interpretations I had not anticipated. For instance, I never set out to see if I could apply the concept of operational logics beyond computation to domains of practice and behaviour (Chapter 6), but links between computational structure and modder practices strongly suggested operational logics as a useful interpretive frame. My conclusions regarding how software architecture acts as discourse, how modders enact authorship, and the contradictions inherent in the way that modding expertise are disseminated (all in Chapter 7) are derived inductively from the coding and categorization of observations.

3.6 Methodological tools and activities

This section takes a look at the specific investigative tools that I have used in this study.

3.6.1 Ethnography in online contexts

Ethnographic methods aim, through the vehicle of participant observation, "to find social meanings as they are implicitly forged and sustained in everyday interaction" (Boellstorff, 2008, p. 75). According to Clifford Geertz, ethnographic accounts should strive for "thick description," in which the cultural context of observed events can be described in a way that attaches meaning to their superficial characteristics, linking emic (insider-perspective) meanings to etic (outsider-perspective) ones. He writes:

> As interworked systems of construable signs (what, ignoring provincial usages, I would call symbols), culture is not a power, something to which social events, behaviors, institutions, or processes can be causally attributed; it is a context, something within which they can be intelligibly—that is, thickly—described. (1973/1988, p. 539)

Participant observation, the primary tool of ethnographic inquiry, is not just observation of the participants in the study as they go about their daily lives—it often entails participation *of* the observer, insofar as it is practical and reasonable. However, virtual and Internet-based field sites pose special challenges for participant observation, and ethnography more generally. Anthropologists and

sociologists conducting research on the internet have used a variety of ambiguous umbrella terms, such as "online ethnography," "virtual ethnography," and "virtual anthropology" to describe projects of multiple different types. There exists a blurry distinction between four different modes of ethnographic research conducted in internet contexts:

1. Using the internet as a tool for collecting data about general social and cultural practices, not specifically related to online activities; Hine notes that the internet is a useful tool for sociologists because it provides a convenient venue to conduct interviews and it naturally produces an enduring textual record of social interactions (University of Surrey, 2013).
2. Studying how people use internet technologies.
3. Studying activities that are (necessarily) mediated through online spaces.
4. Studying semi-bounded virtual worlds as cultures in their own right.

Boellstorff objects to the term "virtual ethnography," arguing that it suggests that there is something virtual or unreal about the culture under study (2008, p. 65). Instead, he favours the term "virtual anthropology" (the method, rather than the object of study, is virtual). However, the research that he refers to as virtual anthropology falls in the realm of the fourth item above: Boellstorff argues that virtual worlds and online communities can be studied not as activities that people do at their computers, but as cultures in their own right, like the proverbial Trobriand Islands of classical cultural anthropology—an approach that he himself and others have adopted (see e.g. Taylor, 2006; Pearce, 2009; Nardi, 2010).

My research falls largely into categories (2) and (3) above. On the surface, playing and modding Minecraft would seem not to be explicitly about the use of internet technologies—surely, the point is the game and its community, not the venue. However, the game and the community are inextricable from their online contexts. Modding is a deeply networked activity, and furthermore, the sociotechnical architecture of Internet communications media is implicated in how modding discourses are carried out (see Chapter 7). A study of Minecraft modding is a study of practices that necessarily take place online, or that exist in a context in which the move from online to offline is

seamless. The nature of the game software itself, the mod distribution channels, the collaboration and discussion venues, and the centrality of multiplayer gaming to the Minecraft experience—all of these take for granted constant, international broadband connectivity. If the platform for the Minecraft software itself is the JAVA VIRTUAL MACHINE, then the "platform" on which the "software" of the modder community runs is the global internet.

On the other hand, participating in a modding community is not merely a *thing that people do on the internet*. The community itself (and the sub-groups therein) has its own social dynamic, its own conventions and linguistic codes. Still, it is too loosely distributed over multiple venues and activities to understand it as a virtual world: unlike *Second Life*, the shared world-space that Boellstorff studied, there is no one *place* called Minecraft (the multiplayer mode runs on individual, user-created servers, which often run significantly different versions and highly personalized sets of mods). There is also no one *place* where the modders hang out and interact online. This research requires understanding that the modding scene has common cultural referents and practices, while simultaneously keeping in mind that it does not fit the Trobriand Islands metaphor the way that virtual worlds and massively-multiplayer online role-playing games do.

For the purposes of this research, engaging in participant observation meant trying to situate myself as an honorary modder, going to the sorts of online places that modders go and doing the sorts of things that modders do. The easiest field activity to reconcile with traditional ethnographic methods was my three-day participant observation at the virtual Better Than Minecamp convention (Chapter 5), because the event entailed real-time, virtually-embodied interaction with modders, and observations of their interactions with each other. Participant observation in lower-bandwidth, text-based communications venues such as IRC and Discord channels, of which I clocked roughly 60 hours across each of 24 channels over a six-month period, was similarly straightforward: there already exists a significant body of anthropological and sociological work based on internal

participant observation in largely text-based virtual spaces (e.g. Smith & Kollock, 1999; Webb, 2001).

Trying to take part in two of the other things-that-modders-do—exchanging asynchronous messages on web forums, and programming mods—is trickier.

For the forums, the researcher can't be everywhere at once, reading tens or even hundreds of thousands of posts across an entire site; there needs to be some way to determine which threads are worth investigating. Nor are the things happening *right now* (the most recent posts, the most recently-updated threads) necessarily the most important and interesting things. In partitioning conversations by topic into more-or-less self-contained threads, and in keeping a permanent record of all past utterances in each conversation to serve as an ongoing referent to future posts, web forums undermine the importance of the immediate that persists in real-time communication. Just as importantly, web forums construct invisible audiences of readers who don't write back, non-participants who decipher and absorb aspects of the discourse and culture they encounter on the message board, perhaps allowing it to inform their actions in other contexts (e.g. an IRC-based conversational meta-commentary on a forum post which all of the IRC participants read, but which only one of them contributed to directly). Since the early days of Internet newsgroups, these ever-present but largely inscrutable invisible audiences have been labeled *lurkers*.

It would be inaccurate to claim that the body of lurkers is not in some way a *part* of the culture and context of an online community—especially since one venue's lurkers may be another venue's core personae. Lurking is a kind of participation. I suggest, therefore, that ethnographers on web forums may be well-served by spending some time conducting *lurker-observation*—not exclusively, of course, but as a supplement to other data collection activities. When lurking, the researcher's choice of threads to analyze should be based on what avenues they would naturally explore if they were approaching the forum with the same goals as the typical lurker. To elaborate: one reason for lurking on a modding-related forum is to absorb information about how to make

mods of one's own. A novice modder can find the answer to many questions by reading threads in which others have already made similar queries. By putting myself in the position of a novice modder looking for answers (as I was actually trying my hand at programming my own Minecraft mods), I was able to *participate* in a social process that provided me with a glimpse of how modding know-how is disseminated online. Researchers might also spend time lurk-browsing forums in order to identify core participants who may make for helpful informants, but the texts encountered along the way should be treated as data and subjected to critical analysis, not discarded as incidental or artificially-selected. As for more active forms of participation, I made very few forum posts myself, primarily for the purpose of recruiting participants, but most of my informants, whom I discovered through other channels, were themselves not active posters on the message boards.

It is not as if researchers aren't already engaging in lurker-observation. Hine suggests that such "unobtrusive methods" follow an established tradition in social sciences, and have a place in the study of online communities (2015, p. 159), though not as a wholesale replacement for ethnographic immersion (2015, p. 161). She does note, however, that "it may be quite normal to lurk without posting" in some online discussion groups, and "it is important to remember that the group itself need not necessarily be treated as a bounded field site in its own right"—that is, the researcher can actively engage with participants through other channels that they naturally use in order to achieve "mutual visibility" (2015, p. 57). In incorporating lurker-observation into my method, I am not necessarily doing anything different from other researchers; what I am suggesting, however, is that we can think of (supplemental) non-reactive methods as being justified, not with reference to the traditional notions of maintaining detached observation of "objective" data, but by noting that in taking on the lurker's role, we are encountering parts of the community in the same way that a substantial number of its invisible participants do.

As to the other issue, observing and participating in making mods, I have already hinted at its relatively simple solution. In order to understand the social processes involved in sharing modding expertise and developing consensus on modding practices, and in order to understand how

those practices were linked to software architectures, I learned how to make mods, and I made mods. I built several of them between 2013 and 2017, which gave me an excellent perspective on how the community's practices evolved over that period. All were fairly simple, minimal, and a bit rough around the edges, but the immersion served its purpose in allowing me to initiate myself into the cultural and linguistic codes of the community I was studying. I made journal entries reflecting on the processes of learning and programming.

By 2016 I was hacking together quick-and-dirty utility mods to fix small annoyances that my colleagues and I had encountered on our research centre's recreational Minecraft server. One of our difficulties was that the heavily-modded server process tended to slow down over time as it ate up more and more of its host machine's memory (some mod or mods kept writing new information to memory, without removing old information to clear up space). This would cause game play to become increasingly laggy, until someone rebooted the virtual server. However, only those with administrator access could do so. I wrote a small mod that would allow any player to reboot the server by means of an in-game button, so that administrators would no longer have to be pestered to take care of the task when they had a moment. In order to enhance my understanding of the modding community's various ideas of "best practices," and of the entire process of building, publishing, licensing, *and* supporting a mod, I later cleaned up the code, ported it to the four most-popular Minecraft versions at the time, and posted it on the CurseForge mod distribution site. As of 2019, the mod is available at https://minecraft.curseforge.com/projects/command-block-server-reset-enabler,[20] and source code is included with the download. Since it was posted in August 2017, the mod has been downloaded 241 times (counting all four files)—and I am *relatively* certain that no more than 50 of those downloads are from me testing the links. To put it concisely, since I could not simultaneously, directly observe *and* participate in modding with other modders, I constructed a

[20] Archived version (without downloads): https://web.archive.org/web/20190401182648/https://minecraft.curseforge.com/projects/command-block-server-reset-enabler

picture of modder practice by observing modder discourses online and doing the "participation" part on my own.

3.6.2 Interviews with modders

Interviews are an important counterpart to participant observation. "At the root of in-depth interviewing," writes Irving Seidman, "is an interest in understanding the lived experience of other people and the meaning they make of that experience" (2006, p. 9). This is precisely the goal of ethnographic analysis in general (with the one difference being that interviews are not necessarily part of a method to study a group or community, but may instead be focused on individuals).

For this study, I conducted in-depth interviews with ten informants. Nine of these were Minecraft modders, either currently or in the past, while one was a programmer and game designer I met at the aforementioned virtual convention, who was creating a Minecraft-like game that was designed with modding in mind. All interviewees were required to be at least 18 years old, although given the online nature of the research, it was not possible to actually verify the ages of participants. On two occasions, modders under the age of 18 expressed interest in participating, but unfortunately this was not practical under institutional ethics standards, which stipulate that minors cannot participate in most research studies without documented parental consent. This points to a need for more "Internet-aware" ethics frameworks that allow for the inclusion of systematically-excluded voices from the under-18 crowd while safeguarding their privacy and well-being.[21]

Interviews were based on a set of open-ended guide questions (Figure 3-1, below), with follow-up questions to pursue new avenues of inquiry based on the interviewee's responses. Participants were given the choice between a voice interview, conducted using voice-over-IP software such as Discord or Mumble, or a text-based interview in a real-time chat venue like Discord or IRC, according to their preference. The validity of text interviews was justified by the fact that

[21] As a privacy safeguard, I did not request any real-world personal identifying information from participants. They indicated consent to participate by "signing" their online screen names on a web-based form. Obtaining documented parental consent would have involved trading off participant privacy, since signed legal documents recording real-world names would have been required.

these are modalities of communication used by the participants on an everyday basis; the interview interaction is thus no less valid or authentic than the participant's day-to-day social interactions with others in the modding community. Three of the interviewees opted for voice communication, while the remainder preferred text. The duration of interviews was between 60 and 120 minutes, with voice interviews generally taking less time than text for roughly the same quantity of content. No difference in the quality, depth, or usability of data was observed between the two methods.

• What is your age? • What is your profession or occupation? • What is your educational background? • How did you start playing Minecraft? • How did you first get into modded Minecraft? • What is your background as a player of digital games? • What is your technical/programming background? Do you have formal training or are you self-taught? • What mods have you made? What are you working on right now? • Do you work on collaborative projects with other modders? Would you? Why or why not? • Do you solicit donations for your work? Do you use advertising links to collect money on your downloads? • What sort of intellectual property licenses do you use for distributing your mods? How do you feel about the licensing options available to you? How do you feel about the choices that other modders make in this respect? • What are some of your favourite mods?	• How do you approach the issue of gameplay balance and compatibility with other mods? • How much time do you spend modding? When do you do most of your work? • Where do you do most of your work? What kind of workspace do you have set up? • Do you see modding as being similar to gameplay, or is it something quite different? • What is the most rewarding thing about modding for you? What is the most difficult thing? • What are your long-term goals? Are you looking for a career in computer programming or game development? • What challenges/struggles is the modding community facing right now? • What do you think the future holds for Minecraft modding? What do you make of the forthcoming modding framework for Bedrock editions? • (BTM attendees) Did you have a booth at BTM? • (BTM attendees) What stood out for you at BTM? • (BTM attendees) Did you watch the Minecamp stream? What was your reaction?

Figure 3-1. List of guiding questions for participant interviews.

3.6.3 Discourse analysis and dual reading

Building on the work of Bahktin, Holt (2004) distinguishes between "monologic" and "dialogic" discourse modes in internet communication. For my purposes, this is at once a theoretical position and an investigative method. On the theory side, instances of communication may appear to be presented (or present themselves) as monologic or dialogic. The monologic-mode text attempts to separate discourse about the world from the world itself, and may even conceal its relationships to other texts; thus it operates like a strategy from a "proper" circumscribed location. Dialogic texts

make apparent the way that communicative utterances are always contextualized by their relationship to other utterances, and are messily embedded into the world to which they refer.

Holt is interested in how dialogue conceives and constructs the world. To this end, he advocates for applying a "dual reading" method to discursive texts, in which they are read through both a "standard" or monologic lens and an "utterance" or dialogic lens, and both readings are subsequently compared. In the standard reading, the message is treated as intentional, well-formed, predictable, and detached from the subjectivity of writer and reader, with a clearly-defined connection between words and meaning. In monologic analysis, any distortion of meaning is typically explained as a "noisy channel" effect. Dialogic analysis treats each message as what Bakhtin calls utterance, a contextual social act that is informed by, and in turn shapes, other utterances as well as conditions in the world. The utterance reading is a *thick description* of discourse.

Holt also identifies four recurring "qualities" of utterance in online dialogues, which serve as useful analytical labels for identifying how communicants position themselves relative to others and to the subject matter (paraphrased from Holt, 2004, pp. 86-87):

1. OTHER MEANING, in which a poster "enlists" concepts (e.g. scientific evidence, conventional wisdom, anecdote) that originate outside of the discussion in order to support an argument.
2. OTHER CONCEPTION, in which a poster makes statements about how they perceive other interlocutors, or the opinions of authorities external to the discussion.
3. EFFORT AT SHARING, in which a poster conflates their own viewpoint with that of someone else, finding commonality.
4. CONTESTING OWNERSHIP, in which a poster distinguishes themselves from others, making claims for or against a person's legitimacy or authenticity.

Holt's dual reading method, including his attention to coding for these qualities of utterance, is applied throughout my analysis to uncover how modder discourses are constructed, and how aspects of modder culture and practices are in turn rooted in those discourses.

3.6.3.1 Forum capture

To expand the dataset available for dual reading, a body of forum posts pertaining to the research questions was identified for analysis based on the "lurker-observation" approach described in Subsection 3.6.1. To offset the potential selection effects of narrowly-conceived lurking, an additional sampling method was used to obtain a representative corpus of modder discourse from three message boards. This served to uncover conversation topics and discursive dynamics that were not visible to other selection methods.

Three web forums were targeted for sampling: the Minecraft Forums (https://minecraftforum.net/forums/), the Forge Forums (https://minecraftforge.net), and MCreator forums (https://mcreator.net/forums/). These websites are described in greater detail in Chapter 7. Threads were also collected from the SpigotMC forums (https://www.spigotmc.org/forums/), but these were ultimately discarded from the dataset as my focus narrowed to Forge-based modding. For each site, specific boards pertaining to modding (see Figure 3-2, below) were targeted. In each target board, the top ten threads displayed on the landing page were selected for capture—saving a copy to local disk—on September 11, 2017. Because of the default sort order used by all four forums, this corresponds to the ten most recently updated threads, and can include a mixture long and short threads, newly-minted ones, and old ones that were "resurrected" by a new reply. (This is not necessarily the way that a registered user might see the forum while logged in, because users can subscribe to threads of interest to them, or mark threads as read to dismiss them.) If pinned threads[22] were among the first ten in the list, these were included in the sample, but additional normal threads were captured to bring the total non-pinned threads up to ten. In some cases, additional threads displayed on the board's landing page were also added to the data set, either because the initial selection of threads contained threads that appeared to depart from the typical discourse of the board, creating the need for replacements, or because the additional

[22] Moderators can "pin" or "sticky" important threads, such as those containing important announcements, so that they always appear at the top of the page.

threads obviously related to topics I had already discussed with informants or otherwise identified as areas of interest.

Viewing a thread means viewing a list of posts in chronological order from oldest to newest. For long threads, this list is paginated, typically with 25 posts per page (10 for MCreator forums). When a thread was paginated, the first page and the final two pages were captured (the final page usually being a partial page with fewer than 25 posts). If fewer than three whole pages were initially captured, the thread was revisited (if still available) on October 11, 2017, and additional posts were collected, up to the three-page maximum. A grand total of 252 threads and 2999 posts were captured. Figure 3-2 shows the number of threads (pinned and normal) and posts that were sampled from each board.

Board	Normal threads	Pinned Threads	Posts
MinecraftForum: Tutorials	10	0	123
MinecraftForum: Mods	10	2	596
MinecraftForum: Mod packs	10	0	131
MinecraftForum: Mod development	10	0	74
MinecraftForum: Mod discussion	10	0	29
MinecraftForum: Requests / ideas for mods	10	0	54
MinecraftForum: WIP mods	10	0	386
MinecraftForum Totals	**70**	**2**	**1393**
Forge: Support & Bug Reports	10	1	50
Forge: Suggestions	10	1	32
Forge: General Discussion	19	1	140
Forge: Modder Support	10	1	149
Forge: User-submitted Tutorials	12	0	181
Forge: ForgeGradle	10	2	105
Forge: Mods	11	0	28
Forge Totals	**82**	**6**	**685**
MCreator: Advanced modding	10	0	72
MCreator: Bugs & Solutions	10	3	107
MCreator: Feature requests & ideas	10	1	53
MCreator: General discussion	10	1	77
MCreator: Help w/ modding	10	1	70
MCreator: Help w/ application	10	0	36
MCreator: Mod ideas	10	0	97
MCreator: Mod showcase	10	1	135
MCreator: User-side Tutorials	10	1	63
MCreator Totals	**90**	**8**	**710**
Spigot: Plugin development	10	4	211
Overall Totals	**252**	**20**	**2999**

Figure 3-2. Counts of captured threads from Minecraft-related web forums.

The overall purpose was not to obtain a statistically significant sample for quantitative analysis, but a sample that would be representative of what a forum reader might see on a typical day, and that would provide sufficient material for identifying and coding patterns in modder discourse. Threads were subject to open and selective coding (Emerson, Fretz, and Shaw, 1995, p. 143) simultaneously: open coding allowed for the identification of unanticipated patterns or repetitions, while selective coding looked specifically for topics that had already come up in my past observations or participant interviews. Additionally, forum posts were coded for exhibiting one of Holt's four qualities of utterance (described above). Codes were later sorted into thematic categories, which are the same basic categories that form the backbone of my analysis in Chapter 7: dissemination and co-regulation of modder expertise, performance of authorship, and performance of authority over terms of gameplay.

3.6.4 Approaches from software and platform studies

My research applies the overlapping techniques of platform studies and software studies in order to explicate the role of computational systems in shaping practice (both modding and playing), as well as the role of discourse in shaping the design of computational systems.

Bogost and Montfort (2009) describe platform studies as "a set of approaches which investigate the underlying computer systems that support creative work," in a way that "connects technical details to culture." The platform studies approach is exemplified in Montfort's five-layer analysis, which he applies to an investigation of the Atari 2600 classic *Combat* (Atari, 1977) in terms of platform, game code, game form, interface, and reception/operation (Montfort, 2006). My analysis (in Chapter 6) of operational logic "families" across the "domains" of computation, play, and modding practice uses a different set of layers, but is based on the underlying function of Montfort's model, which is to connect computational systems to cultural ones.

Software studies is similar to platform studies in that it also connects details of computation to broader, non-computational contexts:

> This includes considering software's internal operations …, examining its constituent elements …, studying its context and material traces of production … observing the transformations of work and its results … and … a broadening of the types of software considered worthy of study. (Wardrip-Fruin, 2009, p. 162)

The turn to operational logics in my analysis is an application of software studies, providing threads to link source code, process, play, and programming. As a "unit of analysis" (Mateas & Wardrip-Fruin, 2009) the operational logic is (like dual reading) at once a tether for theory (as described in Subsection 3.3.2) and a constituent of method—that is, the software studies approach espoused by Mateas and Wardrip-Fruin calls for the identification of operational logics in a system, and the extrapolation of their roles beyond computational process.

3.6.4.1 Teaching modding

Concurrent with this research in the spring of 2018, as part of a semi-separate initiative, I planned and taught two three-hour workshops to introduce the basics of Forge modding to interested students at my university. In these sessions, I provided skeleton code to attendees and walked them through the process of programming a Forge mod for Minecraft 1.12 that does very little besides add a new, relatively-uninteresting solid block with a custom texture. My lesson plan was based on existing online tutorials, particularly those created by Forge-modder *Shadowfacts* (https://shadowfacts.net/tutorials/). A version of the source code for my example mod is provided in Appendix B.

The first workshop was executed as a guest-lesson for students with an introductory-level knowledge of Java in a computational arts class, and was attended by eight students. The second iteration was open to the general university community and was attended by six students. I did not impose any requirements for prior Java knowledge, so I was working with an audience spanning a range of experience levels. After both workshops, students were invited to take an optional feedback survey.

The workshops were primarily intended as an experiment in pedagogy and a service to the institution's community, and a full discussion of the results are not included in these pages. Although the results of the survey are not part of this analysis, certain observations made during the workshops are. In particular, the utility of these sessions for the present research was in using instructional modes and observation of student interactions with Forge modding systems to reveal hidden assumptions underlying the design of the software itself. Selected findings from this investigation are addressed in Chapter 6.

3.7 The expedition begins

Having established the key theoretical mappings and methodological gear we will be bringing along, we are now prepared for a visit to the Minecraft frontier, where ludic trailblazers are forging new settlements.

IV. SETTLING MINECRAFT

> *[T]here have been many Minecrafts, more than the simple "Alpha" and "Beta" labels indicate, and we need to understand how the game has evolved to accommodate both creative construction activities and the survival elements that typify its default settings.*
>
> —Sean Duncan (2011, p. 8)

In 2011, Sean Duncan had already noted that any attempt to discuss Minecraft must necessarily be met with the question, "*Which* Minecraft?" At the time, there were two separate software products[23]—*Classic*, a concept prototype that could be played for free in a web browser and lacked survival elements, and *Beta*, a desktop Java program that was starting to coalesce into a canonical and definitive version. Beta cost money and received major updates every few months, fixing bugs and adding new features that generated buzz, excitement, and sometimes rage on web forums.

Beta was part of a lineage that had previously included *Indev* (**in dev**elopment), *Infdev* (the first version to implement "**inf**inite" procedurally-generated worlds), and *Alpha*, none of which received much post-obsolescence attention, except perhaps for nostalgia purposes. By November 2011, Beta had evolved into the first official release version, Minecraft 1.0. The release coincided with the creator Notch declaring the game "finished" and handing the reins over to his Mojang colleagues, Jeb (Jens Bergensten, also written "jeb_") and Dinnerbone (Nathan Adams). Their continued, incremental addition of new features over the course of the last five years undermines Notch's claim that the game was ever "finished"—over a dozen major updates have followed, with the version number ticking up to 1.13.2 by the end of 2018.

This chapter discusses how emergent player behaviour, including modding, shaped these many Minecrafts, playing a defining role in forging the game's identity. What it now means to play—and talk about—Minecraft is as much as a result of player activities as Mojang's design

[23] This pertains only to the Java Edition lineage, which itself contains these "many Minecrafts".

decisions. As the chaos of early-access Minecraft has settled into a somewhat more stable, canonical form, its cultural terrain has also been "settled" by its players and modders. In the following pages, the practices of commercial developers, players, and modders will be cast in terms of de Certeau's (1984) notions of *strategies* and *tactics*. Subsequently, metaphors of poaching, pioneer settlement, and immigration will be critiqued as possible analogies for modder activity.

In addition to the specific sources cited (such as Duncan, 2011), this historical account of the evolution of Minecraft play practices is based in large part on my own experiences as a player of Survival and Creative modes, a fan of puzzle-based adventure maps, and a consumer of mods since 2011.

4.1 Game Modes: A non-linear proliferation of Minecrafts

The existence of many Minecrafts is not only a result of changes from one version to the next, or the persistence of older vintages: individual versions seem to contain multiple games in one package. As early as Beta, a proliferation of unofficial game "modes" was evident. Each mode constituted its own context of play, with its own ad-hoc rules that were based more on mutual agreement and emerging cultural norms than on game code. Players often kept their different game modes partitioned into different worlds (save files). Forum users spoke of playing *Survival*, *Peaceful*, or *Creative* worlds. Survival meant putting the game's DIFFICULTY setting at Easy, Normal, or Hard, so that monsters appeared in dark areas and threatened the player's creations. The player had to contend with environmental hazards, and in later versions also had to eat regularly to stave off hunger. Those seeking a lower-stress experience would put the setting at "Peaceful", which removes monsters from the world and significantly decreases the effects of hunger. Players in Peaceful still had to gather their own resources, and could be killed by environmental hazards like cliffs or lava pools. The old ad-hoc Creative Mode generally meant using administrator commands—intended for multiplayer servers and debugging, and sometimes considered "cheat codes" when used in a single player context—to obtain unlimited resources. This type of play most closely captures the "Lego"

analogy (Duncan, 2011, p. 6) in which players are free to use building blocks to construct all manner of creations. This mode was well-suited to building massive castles, cities, statues, replicas of real-world monuments, and computing machines (assembled using a curious in-game substance called REDSTONE). Perhaps most importantly for the present discussion, Creative play transformed Minecraft into a kind of game design toolkit, with players building obstacle courses and puzzle mazes for others to solve.

4.1.2 Rules and partitions: From implicit to hard-coded

Since the difficulty setting can be changed at will in a single-player world, and cheat codes could be used as much or as little as a player desired, these different play modes were distinguished from each other largely through the player's own ongoing commitment to one style of play or another. In forums, players would often speak of having multiple active single-player worlds, some being played in traditional survival-mode, while others were dedicated to creative play. All of these would actually take place within the same software context, and using the same underlying rule set and technical constraints, that constitute "Survival Mode" today. The distinction between play modes was not enforced by the software early on, and players could easily slide between game modes in the same world.

By 2014, four canonical game modes had been implemented: Survival, Creative, Adventure, and Spectator. These are afforded—and their rules enforced—directly by the software. Survival can still be played with or without monsters based on the difficulty setting, while the Creative player is invulnerable, can fly, and has easy access to an unlimited block and item inventory through a convenient GUI (the use of typed commands is no longer necessary). Adventure Mode is typically coupled with Creative Mode, in that one player uses Creative to build an in-game dungeon, puzzle, explorable space, or full-fledged adventure game for other players to experience. This creator uploads the finished map file to a website such as MinecraftForum.net or PlanetMinecraft.com for end-users to download. These end-user players are locked into Adventure Mode so that they cannot

freely reshape the world by placing or destroying blocks, but can otherwise move around, collect treasure, manipulate items, use machines, and defend themselves from monsters.

In multiplayer contexts, a server administrator can change an individual player's game mode, allowing for a blended play space in which, for instance, players in Creative Mode can reshape the world for those in Survival or Adventure Mode in real-time. Spectator mode also finds its use here: a spectator is able to fly around invisibly and pass through blocks like a ghost, but cannot affect the world in any way. They can only watch others at play. A spectator is in a position to observe an in-game sporting event or MINI-GAME, such as SPLEEF or a HUNGER GAMES deathmatch, without the possibility of accidental or intentional interference.

The evolution of officially supported game modes has two key implications. First, it serves to solidify rules and distinctions that players previously had to self-enforce. Second, game modes are implemented in such a way that they apply to each player individually, rather than an entire world, as noted above. Thus, game mode has evolved from being an implicit circumstance of the world itself, to a set of possible explicitly defined player roles.

4.1.3 The tactics that helped define game mode

There is an important and striking lesson in the way that the official game modes were derived from the prior ad-hoc game modes that players had developed through active practices. Definitive play modes and styles did not develop strictly within the context of what the game rules most easily afford and encourage, nor were they completely detached from said affordances. Instead, play styles emerged at the boundaries of what was possible, in the space between the impossible and the easy/obvious.

For instance, attempts to shoehorn non-survival play into the original software-provided Survival Mode context often did not go smoothly. Using server commands to obtain building materials in ad-hoc creative play was a tedious process that required one to learn an arcane syntax, and to use a website to look up ID numbers for the desired items (because they could not be

requested by name alone). Additionally, large building projects required pausing construction at regular intervals to summon new materials. Creating tall buildings or bridges meant a lot of extra time had to be spent on placing steps, ladders, or scaffolds that were not intended to be part of the final construction, and would have to be removed later. Breaking unwanted blocks took time and degraded tools.

Some plugins attempted to solve these problems by technical means. The map editing program, MCEdit, in addition to providing an out-of-game building interface, allowed the in-world placement of "Creative chests." Through the direct manipulation of the game's item counters, these chests could include a variety of items in much larger stacks than was normally possible in-game, reducing the need to periodically replenish the stockpile via commands.

There were also several mods/plugins that provided players with flight-like abilities to assist with fast transport and high-altitude construction. One difficulty with these mods is that true flight required substantial changes to both SERVER- and CLIENT-side code. Many worlds ran on the hMod or Bukkit platforms, in which all plugins were supposed to be installed on the server only. Flight plugins had to find a way to allow flight-like abilities without touching the Minecraft client code. One "magic carpet" mod worked around this difficulty by placing and removing glass blocks to form a moving platform in the air that always tracked beneath the player's feet: the player never actually flies freely, but is always standing or walking on a solid surface. The mod was far from perfect (as I learned one day in 2011, while trying to build a massive castle wall): in particular, when trying to place blocks below the player's position, the glass itself got in the way—an issue that no longer occurs with the free-motion flight now natively supported by Creative Mode.

These examples highlight how early creative play was an assembly of kludges, in which players had to constantly find workarounds for logistical problems. The workarounds emerged organically through player practice, and were often at odds with, or resisted by, the underlying software. Players either had to tolerate the awkward and tedious nature of those workarounds, or turn to additional software solutions to smooth out the wrinkles.

Puzzle and adventure mini-games, as well as competitive mini-games, posed further difficulties because players could craft items and alter the world. If the objective of a complex puzzle is to find a way to operate a door-opening mechanism, its purpose is defeated if the player can just punch through the wall instead. If part of the puzzle requires finding a button or switch in a chest somewhere to use as a "key" for the door, the player may be able to circumvent that requirement by crafting their own switch out of a stick and a rock. Although players could certainly refrain from taking such actions, instead playing the game as its designers intended, there is a certain uneasiness inherent in playing this way. Those familiar with video game conventions will tend to expect a game to teach them how to play through its rules, and thus have a normative expectation that the rules will be enforced by the software—anything that goes against the designer's intentions should normally be impossible. The early Minecraft adventure player would instead have to keep in mind that a different set of contextual, voluntarily-observed rules applied in an adventure context. The downloadable world files for adventure maps are therefore accompanied by documents admonishing players to refrain from crafting any items or breaking any blocks.

Let us consider, for instance, two popular puzzle-adventure maps from 2011, Yiker's *Enigma Island* (boasting more than 50,000 downloads)[24] and The_Forgotten's *Monarch of Madness* (200,000+ downloads).[25] Rules for playing *Enigma Island*, as outlined in its MinecraftForum.net post, are simple:

> 1. DON't [*sic.*] break ANYTHING at all
> 2. DON'T place ANYTHING at all
> 3. Play on PEACEFUL
> 4. DON'T use anything else than Diamonds to unlock Hints
> 5. Don't cheat with mods or third party Programms [*sic.*]

Monarch's rules are a little more verbose and suggest possible exceptions, but are substantially similar (only the first two of eight rules are quoted below):

---

[24] According to its official post on MinecraftForum.net: https://www.minecraftforum.net/forums/mapping-and-modding-java-edition/maps/1466504-puz-enigma-island-50000-downloads (thread deleted by its author, no archive available).

[25] Forum post: https://web.archive.org/web/20190401184607/https://www.minecraftforum.net/forums/mapping-and-modding-java-edition/maps/1464203-adv-puz-monarch-of-madness-200-000-dls-works-with

```
1. Do not break blocks and do not place blocks, unless told to. If you
are told to place a block, and you misplace it, you have the right to
break it and replace it. If you're in an impasse where you can only get
out by breaking, say, a wooden plank, go right ahead. Some traps are
like that.

2. Do not craft anything. And that means, anything. Everything you need
will be given to you when needed.
```

Both maps also re-state their rules in the form of in-game signs, posted near the start of the adventure, although *Monarch's* in-game rules are stated through an in-character narrative (Figure 4-1, below).

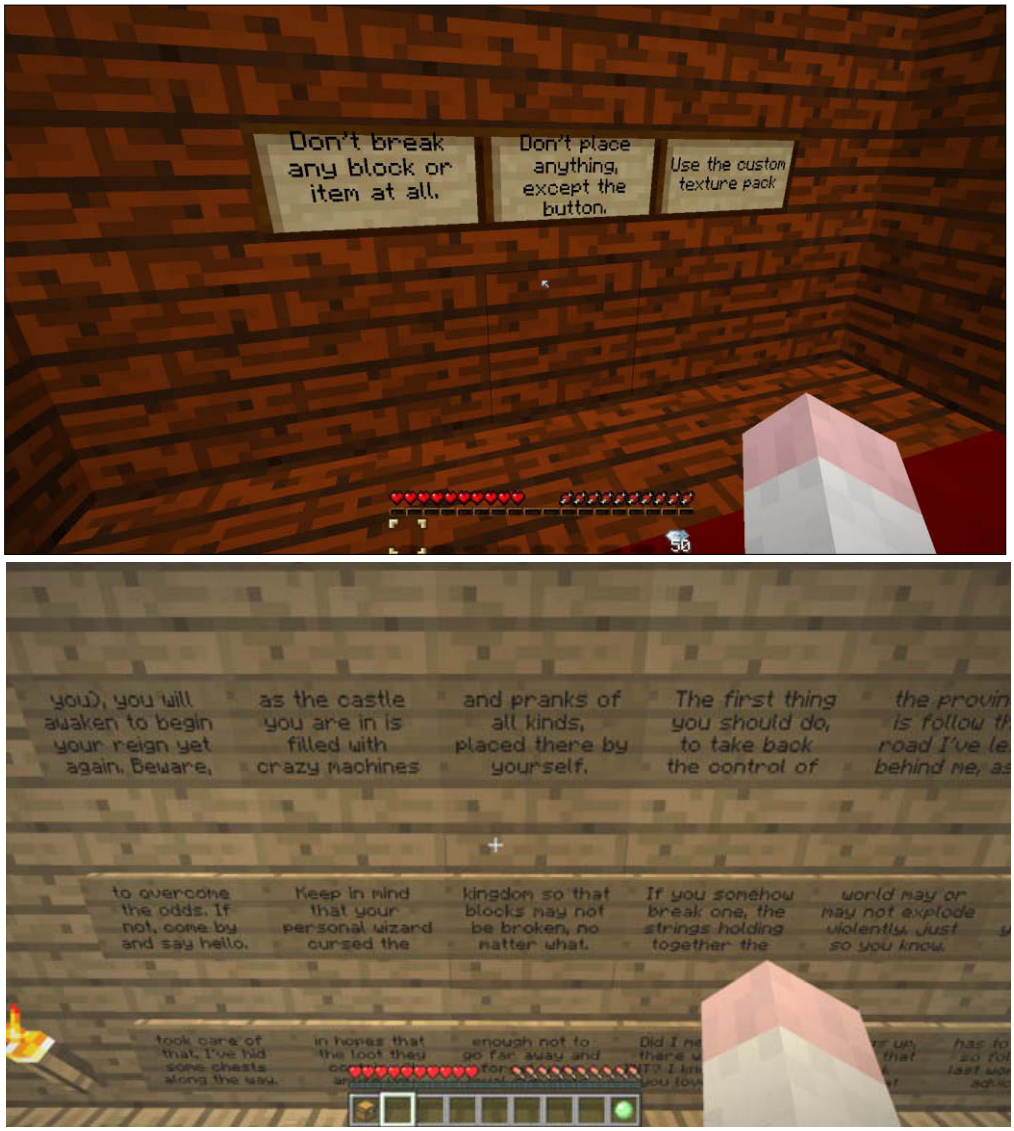


Figure 4-1. The ground rules for *Enigma Island* (top) and *Monarch of Madness* (bottom) are provided on signs in-game. *Enigma Island's* "button" refers to an attachable button provided in some of the puzzles, which has to be placed in the right location in order to activate a mechanism. (Screenshots by N. Watson.)

Enigma Island further reinforced the no-breaking rule by changing the block-fragmentation textures, shown when a block is in the process of breaking down, to display messages accusing the player of cheating (see Figure 4-2, below). This is an early example of a *mod-like* change being used to adapt the game to a different ruleset. However, given that they are natively supported and do not change the game code itself, texture swaps are arguably not mods (see Section 1.4). This texture change *reinforces* the rules via an accusatory reminder, but is not able to truly *enforce* them as code changes could.

Figure 4-2. *Enigma Island* rules enforcement. When using *Enigma Island's* texture pack, attempting to break a block provides progressively more accusatory messages instead of the usual fragmentation effect. The numbered labels indicate the chronological order in which these images appear, as a block is being continuously punched. (Screenshots by N. Watson.)

In these early adventure maps, players themselves were responsible for adhering to a new set of rules that were not actually coded into the game software, but were dictated by the puzzle designer. These rules were often complex, going well beyond the simple maxim of "don't break, don't place." Rules would vary from one adventure to another, and special exceptions were frequent—such as *Monarch's* exception for wooden planks in "some traps," above. Having to navigate these requirements could tax the player's suspension of disbelief, especially in narrative-

driven adventures, as rules would *visibly* intrude in the ludic space. In one area of *The Toybox*,[26] for example, the player is supposed to fight off a gang of zombies in order to get through an area, but with no way for the game to actually detect the status of the battle, only the honour system prevents the player from skipping the fight entirely (see Figure 4-3, below). *Enigma Island*, designed with punishingly-difficult puzzles, implements a hints and scoring system: players start out with 51 diamonds, which they can spend to receive puzzle hints. The hint system works by having the player throw a diamond down a chute, where it lands on a pressure pad, activating a mechanism to display hint text on a sign. The player's endgame score is based on how many diamonds they have remaining—the fewer hints they needed, the higher their score. However, *any* dropped object could activate the pressure pad, so players are on their honour to only use their limited supply of diamonds instead of random trash.

Figure 4-3. "Flip this lever when you have killed all the zombies." Don't do it if you haven't finished the fight – that's cheating! (Screenshot by N. Watson, from *The Toybox* adventure map.)

Such voluntary adherence to rules is not a problem with games in general. After all, a physical chess board cannot actually prevent me from moving my rook along a diagonal, should I wish to do so. When we play board games, sports, or even children's games of impromptu make-

[26] http://web.archive.org/web/20171017103711/http://www.minecraftforum.net/forums/mapping-and-modding-java-edition/maps/1517835-adv-the-toybox-a-community-collaboration

believe, we have little trouble keeping track of, and obeying, complex rule systems. But we seem to expect something different from digital games: an explorable possibility-space that can be charted through experimentation, and renders *impossible* everything that is also *forbidden*.[27] For Miguel Sicart (2009), this is precisely what separates digital from non-digital games, and it hinges having a "hardwired set of rules, which are beyond interpretation and discussion" (p. 27).

Furthermore, for those digital games that profess to provide an immersive and/or narrative experience, it is important that rules integrate with the game program in the ways that gamers have come to expect. Granted, to insist that a game strive to eliminate all non-diegetic frames is to run afoul of the Immersive Fallacy (Salen & Zimmerman, 2003, p. 450). Juul points out that when some part of the fictional game-world seems to directly address us as a *player* (not player-character), it is a positive emotional experience—"it rhetorically befriends us" (2005, p. 183). What can burst the magic circle in a negative way, however, is when "the fictional world gives the impression that many things are possible that are not implemented in the rules" (Juul, 2005, p. 179).

There is something incongruous and uncomfortable in the way early Minecraft adventure maps try to verbally coax players into obedience. We seem to have the opposite problem to Juul's above: the fictional world gives the impression that many things are *impossible* that are not prevented by the rules (or, alternatively, the rules make many things possible that the fiction meekly insists we should not be able to do). The problem is not that we see a sign informing us of the rules (thereby interpellating us as a player, reminding us that we are playing a digital game); the problem is that, having read the sign, we can ignore what it asks us to do.

Nevertheless, the player who wishes to experience the map as its designer intended must consent to follow a set of arbitrary—and voluntary—limitations. That is, until something goes wrong. The vast gap between the hard-wired and the voluntary rules is the source of much ambiguity. How does the player know that they have fallen into one of *those* traps, for which

[27] In a reflection on our experimental Minecraft cheating efforts with our research lab's Minecraft server, 'HT' (from Section 1.1) opined, "Everything not forbidden is mandatory."

breaking something is an acceptable means of escape? Fan-made adventure maps are also not subject to industry-grade testing and are prone to bugs: what if a door-opening mechanism fails to function properly, because the designer made a mistake, or because a Minecraft update subtly altered its behaviour in an unforeseen way? The player has few cues to distinguish between a truly broken puzzle that can only be passed by busting down a wall, and a puzzle that they just have not yet managed to solve. According to Consalvo (2007), gamers often see cheating as legitimate when it is used to restore or re-enter play from a position of being stuck, but such instrumental cheating still potentially diminishes the player's experience (loc. 1208). The ambiguity between a very difficult puzzle and a broken puzzle can interfere with the player's ability to make a satisfying decision about whether to cheat.

Adventure authors also pushed against the boundaries of the early game's representational capacities, often resorting to resources outside of Minecraft to extend the narrative. *Enigma Island* comes bundled with a "secret epilogue," a small Java program that requires the player to enter a password before further narrative is revealed—the password must be found within the Minecraft world. *Accept Your Own Adventure* is a puzzle map thematically inspired by *Portal* and *The Stanley Parable*, two games in which the player navigates through a facility while being manipulated by a disembodied voice. *Accept Your Own Adventure* similarly relies on a voice-over: with no means to play voice clips in-game, the author provided a zip archive of numbered audio files. In-game signs direct the player to use an external program to play the appropriate voice clip for each room (see Figure 4-4, below). The lack of control and the sense of being manipulated or railroaded, so important to the Portal and Stanley Parable themes, ends up in tension with the fact that the game relies on the player acquiescing to listen to the clips at the right time, and in the right order.

These innovative efforts at "soft modding" (Hayes and Gee, 2009) to expand the ludic possibilities of Minecraft were shaky and imperfect, yet they played their part, pushing at the boundaries in creative ways. They presaged the uses of Minecraft as a *platform* on which to build

new play experiences; moreover, they defined the trails that would later be paved by procedural implementations—both official updates and mods.

Figure 4-4. *Accept Your Own Adventure* instructional signs. In *Accept Your Own Adventure*, in-game signs direct the player to listen to audio clips provided as a set of out-of-game files. This sign, attached to a block that is meant to look like an audio speaker, instructs the user to "Play new beginning.wav." (Screenshot by N. Watson.)

4.2 The strategies and tactics of emergent play

I argue for theorizing these player-driven efforts to expand the expressive and procedural possibilities of Minecraft as *tactics*, in Michel de Certeau's terminology (1984). Conversely, it is through the development and enactment of *strategies* that game mechanics and play styles come to be explicitly supported, sanctioned, and readily afforded by the game. First, however, given that de Certeau uses a particular set of metaphors and examples to define these terms, it is necessary to reorient them towards the present discussion of ludic activity within a possibility space afforded by a platform.

In the context of de Certeau's definitions of strategy and tactics (see Subsection 3.3.1), institutionalized and commercialized cultural producers are readily seen as practitioners of strategy, while consumers are the practitioners of tactics. De Certeau described how cultural texts are articulated on both sides of this equation, such that cultural consumers become active producers in their own right. Jenkins (1992) has described the use of tactics by television fans to expand and inflect television narratives in ways that are beyond the control of production studios (and which, in

some cases, push the studios to take different actions from what was originally planned). In prior research, I have characterized the software hacking carried out by *Uru* fans as a tactical response to their loss of an officially sanctioned multiplayer version of the game (they hacked the single-player release to create their own ad-hoc multiplayer server) (Watson, 2012, p. 74). In this context, it seems natural to label software developer Mojang AB (and parent company Microsoft) as the practitioner of strategies that define Minecraft as a product, while players and modders enact tactics to inflect Minecraft play in different ways. However, in light of the above discussion of game modes, as well as closer examination of modding practices, the situation appears to be more complicated than that.

De Certeau uses various metaphors to speak of the tactics-using consumers and their relationship to "proper"[28] institutions. They are "immigrants in a system too vast to be their own, too tightly woven for them to escape" (1984, loc. 157). As readers of cultural texts, they "are travelers; they move across lands belonging to someone else, like nomads poaching their way across fields they did not write, despoiling the wealth of Egypt to enjoy it themselves" (1984, loc. 2537). Walkers in the city employ tactical practices of movement that operate at "street level" beneath the notice—and outside of the control—of the totalizing strategies of the city-as-institution(s), which seeks to subject all of its constituent parts to the orderly logics of mapping, planned transportation networks, and more (1984, loc. 1421-1464).

Where I depart from de Certeau is not in the character or origins of strategies and tactics, but in the way they end up positioning subjects and institutions relative to each other. He insists that what the tactic wins, "it does not keep" (1984, loc. 148), yet the story of Minecraft modding shows that some tactics stick, becoming enduring practices that allow consumers to stake out and defend their own productive territories. As I will argue below, through this process the tactics cease to be tactics as they are captured and transformed into the strategies of a fledging "proper" institution of consumer creativity. Still, it would be a mistake to assert that tactics *qua* tactics accrue nothing, as

[28] The scare-quotes around "proper" are from de Certeau's own usage.

this implies a stable and stagnant relationship that fails to account for how consumer practices become entrenched.

Further, without disparaging de Certeau's immigrant, poacher, and city-walker metaphors, I propose a new metaphor that is well-suited to the Minecraft context, in which players are seen as frontier settlers, spreading outward to colonize a vast platform-provided possibility space. This metaphor will be further fleshed out below.

4.2.1 Strategic design vs. tactical modding

The story of Minecraft modding is replete with examples of modders employing opportunistic tactics. Beta natively supported custom texture packs that could change the appearance of the game, but this feature was found under a menu item labeled "Mods and Texture Packs", hinting that Mojang hoped to provide players with more customization options. With the core game still under development, an official framework to support mods and plugins did not appear to be forthcoming. In response, players resorted to hacking and kludging to get the game to do what they wanted.

Minecraft is written in Java which, by comparison to other programming languages, is easy to reverse-engineer. *Java source code*, written and readable by trained humans, gets *compiled* after writing into Java BYTECODE, which is much more arcane and awkward for humans to work with directly. This platform-independent bytecode is distributed as a software package. The Java Virtual Machine running on the end-user's computer interprets the bytecode on the fly in order to implement the instructions contained within the program (Figure 4-5, below). Because bytecode contains no platform-specific idioms or functions, and Java compilers are fairly standardized, readily-available *decompiler* programs can convert bytecode back into reasonably intelligible source code. There may be a few minor errors to be fixed, but the hacker or modder can nonetheless see the inner workings of the program, make changes, and recompile without having to go to unreasonable lengths.

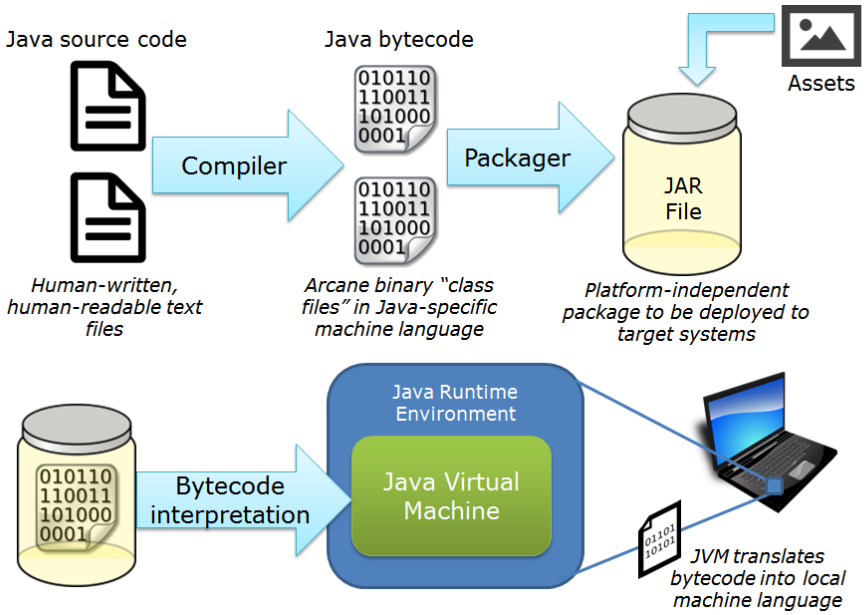


Figure 4-5. Java software deployment chain. Human-written Java source code gets compiled into a Java-specific binary language called bytecode, which is then packaged with asset files (e.g. images) into a distributable JAR file. On the end-user's system, a locally-installed program called the Java Virtual Machine interprets and executes bytecode on the fly. (Diagram by N. Watson.)

The reversibility of bytecode is a natural consequence of the way it was designed to suit the operational strategy of *platform independence*, itself a key component of software development and publishing strategies. In typical Certeau-esque fashion, the strategy inadvertently creates the conditions for its own tactical subversion. Easy decompilation poses a threat to software developers because it could enable software piracy and plagiarism (these "threats" are themselves delineated by strategies of intellectual property imposed by a broader legal-institutional framework). Developers respond with *code obfuscation*. Software tools are used to automatically give variables and functions meaningless names, like "xqn9815546_". Simple instructions might be replaced by functionally equivalent but conceptually confusing versions—for instance, instead of asking if some program variable called **num** is greater than zero, obfuscated code might ask if **num + 1** is not less than or equal to 1. Bytecode can be modified in ways that make the decompiled code invalid,[29] such that the programmer cannot recompile it without first undertaking a lengthy process to fix all of the errors.

[29] For instance, while different variables in Java source code must have differentiable names, one obfuscation technique alters bytecode in such a way that, when decompiled, different variables end up getting assigned the same names in source-code—a problem that needs to be manually fixed before the code can be recompiled again.

Since Minecraft receives updates every few months, and each update is separately obfuscated, early modders had to regularly deobfuscate their decompilations before they could start programming mods, and they had to *reobfuscate* portions of their own code before compiling them together with the Minecraft codebase. (Even updating a mod to be compatible with the newest Minecraft, without changing it in any other way, required recompiling it with the latest Minecraft codebase).

While modding may be a natural response to the open-endedness and unfinished nature of early Minecraft, it is perhaps code obfuscation that made the emergence of a *coordinated modder community* necessary and inevitable. The Minecraft Coder Pack (MCP) sought to provide modders with an automated framework for decompiling, deobfuscating, and recompiling Minecraft. A division of labour appeared in which a small group of people were responsible for creating the basic MCP tools, while a distributed network of modders contributed code comments and suggested sensible variable names whenever they came across bewildering code in their own modding projects. With MCP, each modder no longer had to individually reproduce the time-consuming labour of deobfuscation, and the barriers to entry-level modding were significantly lowered. (Eventually, MCP's developers negotiated with Mojang to get their hands on a Mojang resource called "fernflower", which was able to take care of much of the heavy lifting associated with deobfuscation.) Significantly, MCP explicitly cast itself in terms of players claiming ownership over the game, as the splash screen message in the MCP test environment was changed to read, "Now it's YOURcraft!"

Early mods built with MCP were almost invariably JAR MODS. Minecraft is distributed as a *jar* (or JAR – a Java ARchive) file, which contains a collection of compiled bytecode modules called CLASS FILES. Jar mods work by replacing some of the existing (vanilla) class files, sometimes called BASE CLASSES, with edited versions, as well as (often) adding new files into the jar (Figure 4-6, below). Replacing at least one vanilla class with an edited version is necessary, if only to provide a "HOOK" to tie code from newly added modules into the existing program flow. For the end user,

installing a jar mod was usually a process of using an archive program like 7-Zip to open the primary Minecraft jar, and then drag-dropping a set of modded class files into it. Jar modding is considered a form of *destructive modding*, because of the way that the mod becomes entangled with the vanilla code: far from being "plugins," jar mods must actually tamper with the existing program in order to work. Jar modding has significant disadvantages:

1. Compatibility between jar mods is poor.

   Often two jar mods are incompatible because it is logically impossible to install them both at once. If they both rely on an altered version of a particular class file, whichever mod gets installed later will overwrite the earlier mod's changes. To make matters worse, the class files have meaningless names due to code obfuscation, and these names change with each Minecraft update, such that in order to check for compatibility, users have to look at whether two different mods both contain files called (for instance) "xqh.class".

2. Mod management is difficult.

   There is no easy way to see a list of installed mods, and uninstalling means having to restore altered class files (assuming the user knows which ones were altered) or even the entire jar file from a backup vanilla version.

3. Any minor Minecraft update renders existing jar mod files useless, even if the update contains no changes that would fundamentally alter the way the mod works.

   The mod's original source code might still be valid, but it has to be pasted into the new version of the appropriate vanilla module, recompiled, reobfuscated, and redistributed.

4. Jar mod distribution may violate Minecraft's license agreement, which prohibits the redistribution of any of Mojang's code.

   This poses a problem because every jar mod has to include at least one altered version of a Mojang class file (the hook), which itself contains a substantial amount of Mojang's original work. Unlike many software licenses, Minecraft's agreement does not forbid users from decompiling or

modifying the code in the first place, but when distributing mods requires distributing parts of the original program alongside, this puts jar modders in a legal grey area.

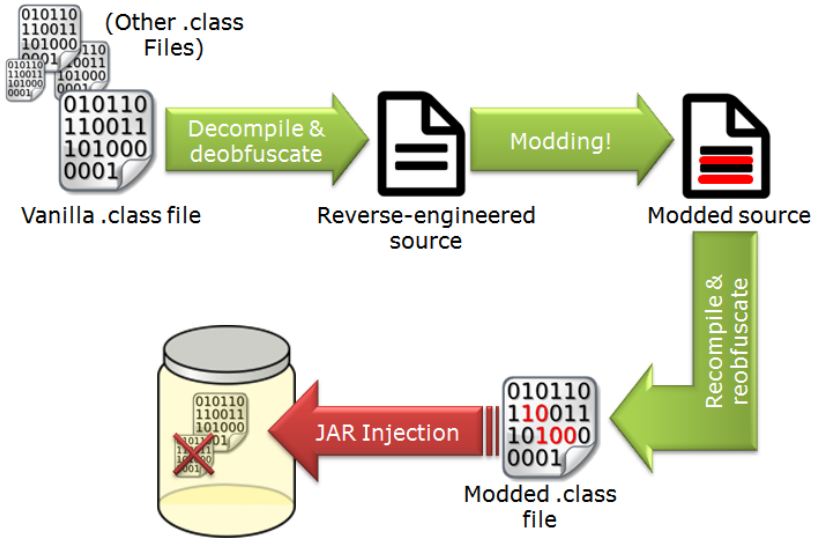


Figure 4-6. Jar modding and jar injection. In Jar-modding, a class file is decompiled and edits are made to the source version. It is then recompiled and injected into the JAR, replacing the entire original version of that class file. (Diagram by N. Watson.)

Despite these shortcomings, jar modding was a vibrant practice early on, making possible several pioneering examples of Minecraft modding. One highly popular jar mod was Zombe's Mod Pack, which included several utilities such as player flight (long before Creative Mode flight was implemented) and a graphical overlay to highlight dark areas where monsters could spawn. Jar modding exemplifies the tactical character of modding: it is unsanctioned, uncoordinated, and opportunistic. It is somewhat clandestine, not in the sense that practitioners try to hide what they're doing, but in that the modded code itself hides beneath the surface of the original program, and mods cannot be easily differentiated, enumerated, or encapsulated as separate entities. It also has difficulty holding onto the territory it claims: like sandcastles below the high tide mark, its gains are regularly erased by the shifting currents of the Minecraft codebase.

Fortunately for the end user who wants to play with mods, modding practices have evolved since the era of jar modding. In doing so, they have increasingly taken on elements that are more characteristic of strategy, turning towards idealized strategies of software design such as interoperability, self-containment, non-destructive modification, standardized code structure,

standardized data formats, unambiguous adherence to license conditions, and extensive use of common APIs to mediate virtually all interaction between mods and the vanilla codebase. This trend is not unique to Minecraft either—modding has become more modular across the board (Yang, 2012).

Risugami's ModLoader was the first major attempt at a framework to support discrete, interoperable mods in Minecraft. Itself a jar mod, ModLoader altered a few key class files in order to provide a common hook or entry point for other mods to integrate with Minecraft. This meant that mods no longer *necessarily* had to make their own destructive changes. ModLoader directly supported a number of important modding tasks such as defining new types of blocks, items, tools, crafting recipes, machines, and creatures. ModLoader mods were still installed by injecting new files directly into the jar, but usually the existing files would be left intact in the process. Some mods still had to make destructive alterations in order to pre-empt certain built-in Minecraft behaviours, but most mods could do without, acting more like the kind of artifact that the word "plugin" implies.

The Forge API was initially developed to solve this latter problem. This jar mod provided a series of code hooks (Figure 4-7, below) and common functions that modders could use to intercept and change the behaviour of Minecraft's procedures without making any of their own changes directly in the class files. The idea was to have a situation in which ModLoader and Forge would be the only two true jar mods, with all other plugins relying exclusively on them to provide an interface with the game. Due to concerns about whether ModLoader's author was going to continue development, and questions as to the legality of the ModLoader distribution package, Forge developer "cpw" eventually decided that Forge needed its own mod loader. He independently coded the whimsically-named FML (Forge Mod Loader). cpw and the Forge developers initially made an effort to point out that Forge and FML were not the same thing and should not be confused for one another, and modders would often state that their mods required "Forge and FML." In fact, they are still technically developed as separate modules today. However, they are distributed together in one package, and are now so deeply integrated that it would make little sense to have one and not the

other: FML, along with the Forge API, are effectively now components of a broader Forge modding platform.

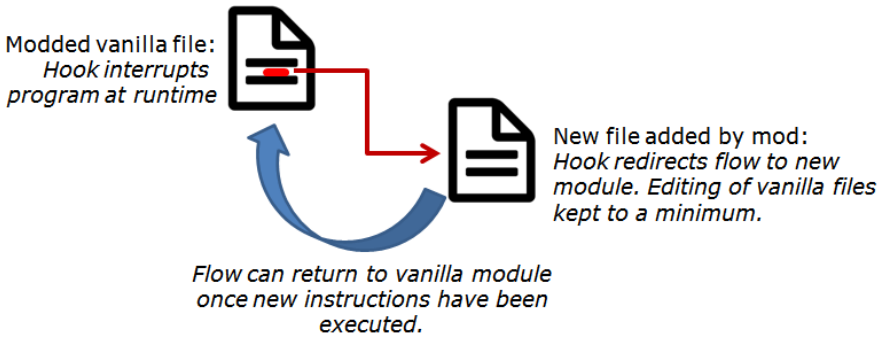


Figure 4-7. Code hooks. A code hook is a minimal change to a vanilla class file, providing a means of interrupting the program to perform some new, modded procedure. Most of the new code is contained in a whole new file, which keeps the code from different mods separate, and minimizes destructive editing of pre-existing code. While jar modders might make hooks for their own use, ModLoader and Forge sought to offer a set of common core hooks to which other modders could attach their new files, without having to do any destructive editing themselves. Multiple different procedures from different mods could depend on the same code hook. If no new code was added, the code hook would have no effect and program flow would simply continue as normal. (Diagram by N. Watson.)

Over the course of years of development, Forge/FML has introduced some significant changes to the way that mods are installed. Perhaps most significant for the average player is that there is no longer any need to mess around inside jar files. The Forge download itself contains an install script that takes care of making the initial alterations to Minecraft files, and FML mods can now be installed simply by placing a cleanly-packaged zip or jar file into a dedicated "mods" folder. Additionally, rather than distributing altered versions of entire Minecraft class files to replace the vanilla files wholesale, Forge uses scripts to make surgical changes to specific portions of the existing files, thus sidestepping the potential legal problems of code redistribution.

While these changes have had the effect of expanding the possibilities afforded by Minecraft as a platform, they have simultaneously constrained the set of "proper" practices for realizing those possibilities—much as de Certeau's city streets widen the range of places that can be visited while limiting the number of possible (sanctioned) paths one can take to get there. In particular, certain types of minor changes to game behaviour are much easier to achieve through a bit of quick jar modding—add a line of code here, delete one there, recompile, and done. Forge/FML mods that achieve the same ends typically require a lot more overhead labour just in setting up a mod that can

make use of Forge functionality. The possible ends have expanded, but the set of acceptable means has shrunk.

4.2.2 Tactics reconfigure strategies

Far from being invisible and inscrutable to institutions of power, as de Certeau suggests, tactical practices provide a raw material from which new strategies can be developed to consolidate power relations. Game developers like Mojang seek (strategically) to solidify and expand their audiences through innovation, but planned, top-down innovation is risky, given that the investment of resources may not pay off if the innovations—new game mechanics implemented through Minecraft updates, in this case—do not resonate with audiences. Instead, a fair amount of innovative labour can be outsourced to the player (modder) community. I am not implying, as Kücklich (2005) does, that game developers necessarily intentionally support and encourage modders in order to exploit their free labour. Rather, the fact that players-as-modders inevitably step outside of the space circumscribed by top-down game design provides developers with an opportunity that is too good to pass up. It is no accident that Minecraft's officially supported game modes are based on the earlier practices of players who pushed uneasily but opportunistically at the boundaries of what the software afforded. In fact, not only has Minecraft formally incorporated player practices, it has added functionality that originated with actual mods, such as mechanical pistons and rideable horses.[30] Mojang is far from the first developer to do this: Kow and Nardi (2010) have documented how Blizzard brought about the end of some *World of Warcraft* mod projects by incorporating them into the game—while other mods that they regarded as undesirable were rendered incompatible by changes to the code-base.

[30] The license agreement that permits modding also allows Mojang to poach game mechanics and ideas (but not actual code or assets) from mods. Even without the license stipulation, it is unlikely that modders would have any legal recourse if their ideas ended up in vanilla, since game mechanics themselves cannot be copyrighted. Although Mojang certainly benefits from being able to harvest modder creativity, the license terms are also arguably necessary to prevent modders from being able to claim priority on ideas in a way that would restrict Mojang's future design choices for Minecraft.

When developers benefit from player tactics, it is not the tactics themselves that are taken up by the developer, since tactics are *means* that only operate from unorganized, nomadic positions of "making do." It is the accomplished *ends* that are taken up and re-implemented through top-down strategies, in a way that effectively suppresses the tactic by robbing it of its reason for being. When tactics do not keep what they gain, it is often because they cede those gains entirely to the power of the commercial game developer: there is little need for the magic carpet kludge anymore, now that player flight is built right into Creative Mode. If, as de Certeau writes, tactics are small "victories of the 'weak' over the 'strong'" (1984, loc. 153), then these moments of uptake are ambivalent: on the one hand, the developer is forced to take notice, change its plans, and ultimately give the players something they want; but on the other, the original tactical practice, in all its ingenuity, is erased, and the developer gets to define the terms by which its original goals are re-implemented.

4.2.3 Commercial developer tactics?

The fact that tactics cease to be tactics when they are strategically incorporated from a position of power does not stop commercial entities from adopting the language of tactics, or a rhetoric of preferring tactical action over strategy. Notch's statements of software development and game design philosophy can illustrate this point.

Although Notch ceded his lead development role to Jeb in 2011, his original "About the game" write-up remained on the official website, Minecraft.net, until late 2012.[31] Here, by way of introducing a discussion of his development practice and philosophy, Notch proclaimed, "Waterfall is dead, long live Agile!" without further explanation. Duncan (2011, p. 8) argues that this statement is indicative of Mojang's flexibility and responsiveness to players, which has given rise to both the phenomenon of "many Minecrafts" and the co-constructed nature of the gameplay.

[31] This determination was made by comparing archived snapshots of the site in Archive.org's Wayback Machine. The original text, in which Notch uses first-person pronouns, is last seen in a October 14, 2012 snapshot (https://web.archive.org/web/20121014094057/minecraft.net/game). The next snapshot, from December 13, 2012, features new text in which Notch is referenced in the third person (https://web.archive.org/web/20121213011003/minecraft.net/game).

In that quote, Notch was referring to two common—and commonly contrasted—paradigms of software development. His proclamation is worth a closer look given the discursive modes it activates: the implications change significantly between the standard and utterance readings (described in Subsection 3.6.3), and a clearer understanding of the statement puts us in a better position to evaluate Duncan's conclusion.

4.2.3.1 Standard reading

The *waterfall* model is a classic software-engineering approach that treats development as a linear process consisting of distinct phases to be completed in sequence. For instance, start by *determining the required specifications* for the software in consultation with the customer; next, *design* the software architecture, from high-concept to low-concept; then *implement* through coding; *test* the software to make sure it functions and performs according to specification; and finally, deploy software to the user and *support and maintain* it. Each phase is assumed to have been completed in its entirety, and is thereby "frozen," before the next phase commences (Weisert, 2003) (waterfalls are generally unidirectional). Royce (1970) is typically credited with providing the canonical description of this approach, although he does not use the term "waterfall."

Agile methodology arose from a meeting of industry leaders in 2001 as a rejection of the rigidity of waterfall, advocating adaptability to changing conditions over predictive planning. These executives, according to their "Manifesto for Agile Software Development," favour smaller, self-organizing development teams delivering working software to the customer early and often. Development is a rapid, iterative cycle in which programmers can respond to customer feedback or changing specifications, adapting each new cycle's deliverable to the shifting landscape.

Duncan (2011, p. 8) portrays waterfall as the exemplar of strategically-planned top-down management—rigidly structured and inflexible. In contrast, he notes that "*customer collaboration* is an explicit element" of Agile (emphasis original). Duncan argues that Minecraft's development

exemplifies the Agile approach, with its iterative addition of new features in response to player feedback, which was often tweeted directly at developers' personal Twitter accounts.

Waterfall versus Agile looks like another drama of strategy versus tactic, with Mojang rhetorically casting itself as the nimble tactician. The Agile developer does indeed *mimic* the situational opportunism that is characteristic of tactics, and thus finds itself in a good position to reap the benefits of player behaviour. At the same time, it creates favourable conditions for players to make more independent contributions to the dynamism of the product (i.e. mods). Of course, Mojang is not actually adopting a tactician's position—but let us return to this point in the utterance reading, below.

4.2.3.2 Utterance reading

Notch's statement is undoubtedly part of an ongoing dialogue about game development, but it is a highly diffuse one in which it is impossible to draw proximal links between this and specific other utterances, as one might do with posts in a forum thread. We can nevertheless use the utterance reading to reveal a second system of meanings in the statement, by identifying how it implicitly constructs and activates a prior dialogic context in which it can be understood.

Notch is positioning his development philosophy as "agile"—that is, responsive and flexible, with implicit connotations of independence—as opposed to traditional, rigid, and industrial. He engages in the dialogic practice that Holt (2004, p. 86) calls other meaning—when an interlocutor "enlists" concepts that originate outside of the immediate discussion. Specifically, he enlists existing rhetoric about the rigidity of waterfall, and the flexible "values" espoused by proponents of Agile. To understand how this other meaning plays out, consider what Notch writes immediately afterwards regarding his approach to game design:

> I've got a few plans and visions, but my only true design decision is to keep it fun and accessible. There's no design doc, but there are two lists; one for bugs, and one for features I want to add but think I might forget.

> I make sure to play the game a lot, and I've built my share of towers, and flooded my share of caves. If something ever doesn't feel fun, I'll remove it. I believe that I can combine enough fun, accessibility and building blocks for this game to be a huge melting pot of emergent gameplay.

The other meaning enlistment of the waterfall/Agile binary serves to elevate Notch's description of practice beyond a casual statement of personal preference. It situates it in a broader professional discourse about software development, and suggests intentionality—maybe even *strategy*—on his part.

Notch is also engaging in what Holt (2004, pp. 86-87) calls other conception, wherein the speaker asserts a certain perspective on a concept. The construction of the sentence suggests that what Notch most wants us to know about waterfall is that it is "dead"—outmoded, obsolete, belonging to an old and now-disfavoured paradigm. Meanwhile, Agile is introduced as the new replacement—fresh, superior, and expected to have a bright future (a long life). Mojang is shown to be a forward-thinking studio by embracing this regime-change.

However, in addition to using the waterfall/Agile binary to construct an understanding of Minecraft's development, Notch's statement is also taking part in (re)constructing the concepts of waterfall and Agile themselves. Weisert (2003) argues that the waterfall model is a straw-man caricature and does not correspond to any actual programming practice, calling for readers to "question vague claims from tools vendors and methodologists that tie the virtues of their product or methodology mainly to a rejection of the waterfall approach." He asserts the value of workflow models that have "phase-limited commitment" wherein stages are funded, approved, and carried out one at a time in order to maintain control and reduce wasting resources. Any real-world project, he claims, maintains the necessary flexibility to re-iterate prior stages of the cycle. Weisert is still describing a rationalizing, strategically managed, top-down practice, from which Agile's bottom-up rhetoric attempts to dissimilate. When Notch and other Agile advocates say "waterfall," they may be using it as a catch-all term for formally-planned life-cycles, rather than actually asserting the

existence of a specific and excessively rigid prescription. Either way, discourse about Agile constructs waterfall as an existing, dominant paradigm and then defines itself in opposition.

Yet Agile could hardly be described as a tactic: despite its nod towards customer collaboration and bottom-up innovation, it remains a calculated operational strategy, developed and endorsed by industry experts seeking to improve efficiency. However, it is a strategy-in-tactical-gear, one that *mimics* much of the situational opportunism of tactics, and as such is in an ideal position to reap the benefits of player behaviour. By embracing Agile, Mojang attempts to capture what Critical Art Ensemble (2003) classify as a "nomadic power." In response to scholarly enthusiasm regarding the possibilities for nomadic (tactical) resistance in the electronic age, CAE warned that entrenched institutions could themselves co-opt nomadism to expand their power bases. Agile software development, while relatively benign, is symptomatic of this embrace of nomadic thinking. Mojang, operating as it does from an established power-base, is *not* the agile tactician it purports to be in the standard reading: it is, rather, co-opting the rhetoric of tactics.

The attempt to enclose the proceeds of nomadic power in rationally-executed strategy gets to the heart of what Malaby identifies as a fundamental tension inherent in trying to design games to produce emergence. Having described games as "socially legitimate spaces for cultivating the unexpected" (2009, p. 14), he notes that the game developer as a modern corporation is in the tricky position of aspiring (strategically) to reduce unpredictability and exert rational control over the business of making games, while also seeking to leave a space for the user's playful activity. In addition to fulfilling Malaby's formal requirement that games have contingent outcomes (2007), this "room for play" is often a part of the developer's business model, especially in the case of software products (like Minecraft or—in Malaby's example—*Second Life*) that explicitly set out to create a platform for emergent play: the developer need not explore all creative possibilities itself, instead relying on the ingenuity of crowds to discover some of the most novel and enticing play possibilities.

Of course, as we have seen, designing for emergence means redirecting—not eliminating—unpredictable outcomes towards strategically-encapsulated goals.

4.2.4 Modders develop strategies of their own

While Mojang's strategies take on a decidedly tactical inflection, modder tactics solidify into strategies not through their uptake by Mojang, but through emergent self-regulation among the modder community. This has been discussed above in the account of how the Forge framework evolved. Especially remarkable is the passion with which some modders defend the new strategies of their fledgling institutional power bases, and how quickly and strongly they disavow once-favoured tactics. The status of "COREMODS" is illustrative. A coremod uses a code LIBRARY called ObjectWeb ASM to modify Minecraft's bytecode on the fly. It is essentially code that rewrites other code. FML necessarily makes extensive use of ASM bytecode manipulation in order to set up the environment into which other mods can be loaded. As noted previously, Forge could not provide hooks for every conceivable bit of vanilla Minecraft code that a modder might want to change. Around 2012 and 2013, whenever modders felt the need to make changes to the vanilla codebase, ASM was considered a far cleaner way to go about it than the jar-mod-style wholesale replacement of classes. The resulting mods were called "coremods." They had—and still have—a special installation folder in the Forge directory structure. Several tutorials spread throughout the Minecraft.net forums could provide detailed instructions on how to make one. Yet since 2016 or so, if one were to ask about coremods, one would quickly learn that they are now treated as anathema to everything that Forge stands for. In early 2017, a user asked for help on the Forge forums getting a coremod to work with the latest version of Minecraft. The only three responses, all from forum regulars (including the Forge founder, LexManos), are instructive:[32]

```
#1    Don't make a core mod.
      Alternative options, to be used in order:
      -  Use APIs
      -  Use Events
      -  Use Reflection
```

[32] See Glossary for REFLECTION and PULL REQUEST.

```
        - Ask on the forums, you probably missed how to do one of the
          above
        - Make a Forge pull request on GitHub
        - Still don't make a coremod

#2      Coremods are not officially supported by Forge. Why do you want
        one? Please explain your use case and we will try and find a
        solution without it.

#3      Do not create coremods. There is no reason to.
```

What was accepted and encouraged just a few years prior is now strongly discouraged because it is inconsistent with the new logics espoused by the consensus of Forge developers, in which invasive changes to the program's core behaviour by individual mods are to be avoided in favour of strategically-planned "safe" solutions relying on the Forge hooks that appear in later versions. Response #1, in particular, lists the "proper" sanctioned solutions. Some practices that were once tactical in nature have clearly become strategic prescriptions for modders, while others are cast as chaotic and threatening to the emerging order. Chapter 6 provides further analysis of this emergence of sanctioned best-practices as a rationalization of the operations of Forge modding.

4.3 The platform: Heartland and frontier

In light of the above, de Certeau's poachers and city-walkers do not seem to capture the complexity of player and developer practices. The poachers are too reliant on what others have created rather than what they can create for themselves, while the movement of walkers is too inwardly focused, towards the interstitial spaces of the city. Instead, the actions of players and modders are more akin to a rhetoric of frontier settlement and trans-Atlantic migration—an appropriate metaphor, given that Minecraft's open-world gameplay itself almost always falls into a narrative of hearth-and-hinter, and of taming a wild frontier (for an extensive critique of Minecraft's implicit colonial/frontier gameplay narrative, see Bull [2014]).

However, caution is warranted here: Minecraft is hardly the first game or digital space which has been described in pioneering/immigration terms, and all such comparisons are tangled with a dark real-world history. Gunkel and Gunkel (2009) note that terms like "new world" and "frontier"

are often used by scholars, journalists, and other commentators to frame our encounters with MMORPGs and other new digital domains as recapitulations of the discovery and colonization of the Americas. Acknowledging that these "hard to resist" comparisons "clearly have a way of articulating what is really interesting and compelling about this technology" (2009, p. 119), Gunkel and Gunkel are highly critical of their careless use:

> [These terms] have what Sardar calls a "darker side," specifically the forgetting of history, the imposition of colonial power and the exercise of ethnocentrism, and the unfortunate exclusion of others. Colonization, violent conquest, and bloody genocide necessarily haunt the use of this terminology and mitigate against its effectiveness and significance. To make matters worse, the current publications, marketing literature, and academic studies surrounding MMORPGs wilfully ignore, unconsciously suppress, or conveniently forget these important complications. (2009, p. 119)

I am not arguing that Minecraft play and modding reflects a history, but rather a *myth* of history. However, it is generally a schoolbook-sanitized myth that attempts to dodge guilt by simultaneously erasing both the presence of non-Euro-American others and the crimes committed against them. Yet the adaptation of this myth to the digital encounter is already deeply entrenched, "as much as a result of computer programming and game design practices as it is a product of the discursive decisions made by game developers, marketing firms, journalists, gamers, scholars, educators, bloggers, etc." (Gunkel & Gunkel, 2009, p. 120). In the case of Minecraft, we see it in the motifs of gameplay, and in the casual way that Notch invokes the figure of the "melting pot" in his writings on design philosophy. It is worth considering how modder practices fit into this narrative, problematic as it is. In doing so, I hope to go beyond simply acknowledging what the sanitized history leaves out, to demonstrate how the modder story can be understood in terms of some common *criticisms* of the ethnocentric colonial myth.[33]

[33] To be clear, my purpose in comparing modders to settlers is *not* to insinuate that modders perpetrate colonial atrocities.

4.3.1 Vanilla gameplay and the settlement of the frontier

Goetz (2012) argues that Minecraft articulates "tether" and "accretions" fantasies. A tether fantasy is characterized as "the pleasurable process of oscillating between feeling safe and feeling exposed, dwelling on the boundaries that separate the two," while accretions are about the "pleasurable process of correcting a weak or vulnerable body by accruing objects from the world of gameplay" (Goetz, 2012, p. 420). In Minecraft, the site of safety is a "home base" that the player has built out of harvested materials. It is a space that is known, mapped, bounded by artificial walls or fences, and—most important of all—well lit. Goetz writes,

> … home base comes to rely on artificial, internal lighting like torches or a burning oven. If players are caught outside after sunset, they might spot torchlight through those same home-base windows as they scurry back to safety. Players can install external lighting along a walkway, perhaps, so that ''home region'' is lit as well—but darkness inevitably encroaches dangerously around built structures, and along with it comes monsters that can be heard stalking the player through walls even when not visible. (2012, p. 428)

The Minecraft player's progress is thus gauged not only by the accretions of complex machinery and precious resources, but by the gradual claiming and taming of space from the unknown, converting it from wild-space into homestead-space through the addition of lighting. It is a game about bringing the literal light of civilization to a literally dark continent. In this, it exemplifies what Jenkins and Fuller identified as the "new world travel" narrative in games, a "shift in emphasis from narrativity to geography" (1995, p. 58).

To be more specific, at the heart of Minecraft's representational mechanics and ludic pressures lies a procedural rhetoric (i.e. Bogost, 2007) that re-figures the mythos of nineteenth-century settlement in the American west (a curious theme for a Swedish-developed game). The self-sufficient pioneer-settler enters a wide, wild, and (supposedly) uninhabited land full of promise and opportunity, constructs a home out of locally-available materials, and begins to impose order upon the territory. The new industrial infrastructure that solidified the European settler's grip on the dwindling frontier—the railroad, telegraph, electric lighting—reprises in Minecraft in the form of

minecarts, redstone wires, and lighting systems, the latter of which are most effective at keeping monsters away when they are laid out in a regular grid pattern (Wershler, 2015). There are even echoes—probably unintentional but nevertheless troubling—of representations of encounters with indigenous populations. Belligerent zombies, bow-wielding skeletons, and explosive creepers feature as wilderness-dwelling "hostile" entities, intent on sabotaging the project of civilization. There are also "docile" villagers that live in rustic agrarian settlements and will trade goods with the player—but can also be attacked, exploited, or managed like livestock for the player's advantage.

4.3.2 Platforms, possibilities, and pioneer rhetoric

Even as Minecraft players settle the in-game virtual world, they also colonize a more abstract space of possibilities and rules, expanding the reach of their cultural practices through modding and other emergent play activities that have been described in this chapter.

To model this dynamic, let us consider Minecraft in terms of a *platform* that enables (affords) a range of possible activities. These activities form a *possibility space*. The boundaries of this space are established by the strategically-designed affordances of the platform, but the edges are fuzzy: as we have seen, there are activities that are *possible* in an uneasy sort of way, things that seem to go against what the system was actually designed to support. Modders and players are constantly pushing on the boundaries of the possibility space, expanding and puncturing it (Figure 4-8, below).

Adding in the feedback mechanisms by which the developer incorporates modder innovations into the vanilla game, and by which modder practices become standardized and institutionalized, a picture emerges of a space that is shaped by inward and outward forces (Figure 4-9, below). In the heart of this space resides the center of civilization, the broad paved highways and the settled, highly-developed land—the set of conventional possibilities that are easily afforded by the vanilla code. Out on the frontier, modders use tactical practices to push deeper into the wilderness, beating wandering trails that, at the moment of their creation, cease to be true wilderness but are not settled land either. These same modders turn their attention inward, over the recently-

made trails, newly-ploughed fields, and frontier homesteads, and attempt to impose a strategic discipline upon them. Modder strategies are always *inward-facing*, oriented towards the centre, while the tactical practices are *outward-facing*.

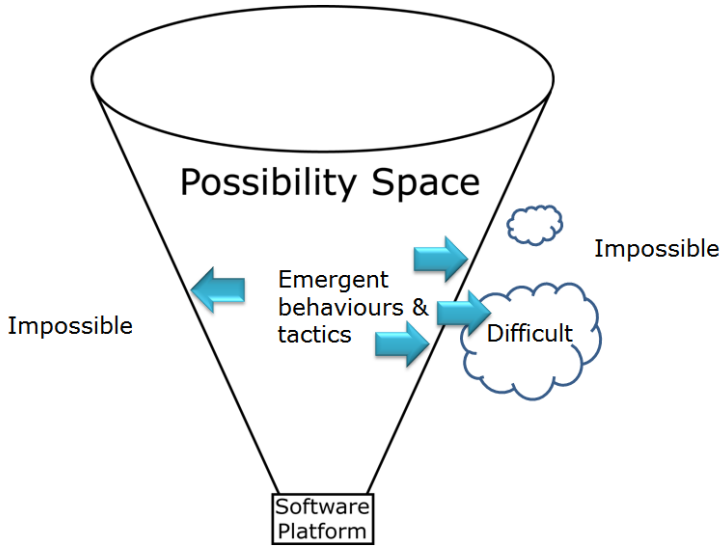


Figure 4-8. Software platform and possibility space. Modders and players are constantly pushing the boundaries of the platform-provided possibility space. (Diagram by N. Watson.)

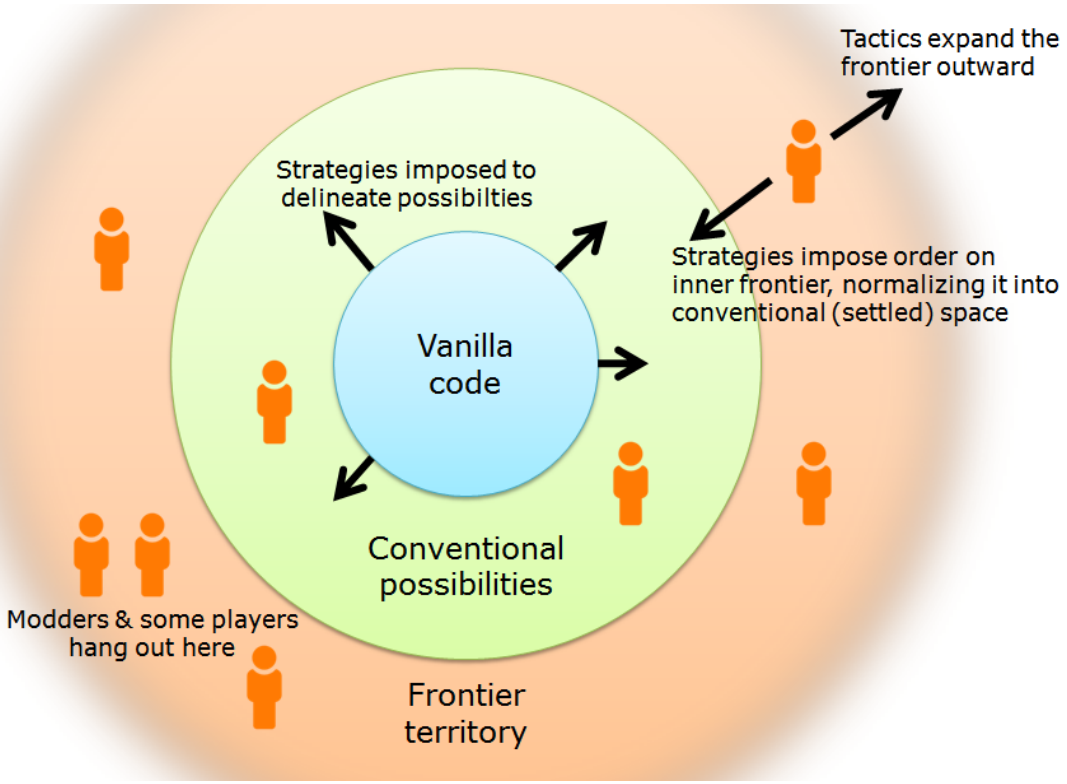


Figure 4-9. Modders are frontier settlers. The heartland and frontier are shaped by a combination of outward-facing tactics and inward-facing strategies. (Diagram by N. Watson.)

4.3.3 "A huge melting pot of emergent gameplay"

The mythical narrative of America's identity formation takes a turn in the late nineteenth century, from the settlement of the western frontier to ongoing trans-oceanic immigration into the new land of freedom and opportunity. When Frederick Jackson Turner expounded upon the significance of the frontier in 1893 (as reprinted in Turner, 1921), it was a retrospective on an era that had already come to a close: by that time, the Statue of Liberty had already been welcoming newcomers to the United States for seven years. These immigrants differed from the frontier settlers in that they were arriving not in a (supposedly, but of course not actually) "empty" land but in a nation that was already partly built, one with which they were expected to integrate.

McDonald (2007) suggests that, before the late twentieth century, historical commentary on the role of immigration in American nation-building was dominated by assimilation theory, of which there are two varieties: the hegemonic, Anglo-conformity model, and the *melting pot*. He explains:

> The melting-pot model depicts assimilation as a multidirectional blending of peoples and cultures, the end product being the creation of an entirely new people and culture. This egalitarian slant on the theme of assimilation has a long tradition in the United States, dating back to the late eighteenth century. (2007, p. 50)

Discourse on integration suggests issues of identity and ethnicity, terms which implicitly encapsulate the totality of a person's meaning-making systems and social context. While I maintain that modder practices should be understood as cultural practices, it is doubtful that they have such totalizing implications. Still, theories of cultural integration can provide insight here: because the cultural domain of Minecraft expands to encompass both Mojang's shifting design goals and modders' ideas, it becomes necessary to turn from metaphors of colonization to those of immigration and integration.

Notch himself actually takes the first step in re-framing Minecraft's particular version of the digital encounter drama. Recall how, in his explanation of Minecraft's design philosophy (see

Subsection 4.2.3), he hoped to create "a huge melting pot of emergent gameplay." Instead of repeating the dominant *terra nova* rhetoric, he explicitly advances the story to the next corresponding stage of the mythic history. Through this wording, Notch seems to express two assertions (or at least, aspirations):

1. Minecraft (the game itself), like America, shall have its own unique character and identity, differing from other games and genres.
2. This identity shall be forged through the blending of contributions made by both the developers and a diversity of players (including modders), with different goals and play styles.

While early modders were pioneers, those who arrive to the scene more recently find a system of practices already intact (don't distribute Mojang's code; eschew jar modding; use Forge or Bukkit; and definitely don't make a coremod!!) and, like immigrants in the melting pot, are expected to adapt their own contributions to this milieu.

Of course, the American melting pot was (and still is) itself a rhetorical strategy for shaping immigration policy and justifying what turned out to be a primarily Anglo-conformist hegemony. It is worth remembering, too, those groups that were systematically excluded from the largely Euro-centric melting pot history, such as African-Americans and nineteenth-century Chinese immigrants. If, following Kücklich (2005), we were to classify modding as exploitative labour dressed up as freedom and opportunity, we would see striking similarities to the way that "problematic" immigrants in American history were valued for their cheap labour but not for their contributions to cultural identity.

Yet the melting pot model fails to capture the diversity not just of play styles but of entire *game modes* in Minecraft. The "kaleidoscope" metaphor preferred by advocates of multiculturalism (Fuchs, 1990, p. 276) more closely describes the vibrant and multifaceted nature of the Minecraft

playground. After all, the lesson from the beginning of this chapter is that there is not just one Minecraft, but many.

In the next chapter, we will see a snapshot of what the world of "settled" Minecraft and Forge-based modding looked like in 2017, at an online convention to show off and celebrate mods.

V. BETTER THAN MINECAMP: AN UNCONVENTIONAL CONVENTION

ATTENDEE 1: Um, this is a mess!

ATTENDEE 2: No, this is BTM!

—Attendees at Better Than Minecamp, 2017

We can't have the least amount of crashes, but we can still try to go for the most amount of crashes.

—copygirl, co-organizer of Better Than Minecamp, 2017

Early on in the research for this project, I considered the possibility of shelling out to attend "Minecamp,"[34] Mojang's yearly promotional convention, to get an inside look at Minecraft fan culture and the developer's corporate messaging, and possibly to get in touch with prominent modders.

I never ended up going to Minecamp, but I went somewhere better—a fact that was proclaimed in the very name of the event, *Better Than Minecamp* (BTM). This virtual convention, held in-game on a heavily-modded multiplayer server, celebrates Minecraft modding and provides modders with an opportunity to showcase their work to each other and to other players. The 2017 iteration—the fifth since the idea was launched—was, like many of its predecessors, held concurrently with Minecamp.

Initially, I thought the name was intended as a humorous acknowledgment of the fact that many guests would prefer to actually be at Minecamp but had to settle for a less glamorous online, fan-organized event which defiantly claimed offer be the superior experience. However, it quickly became apparent that most attendees were much happier to be at BTM, and many regarded Minecamp with indifference or even scorn.

Minecamp 2017 was held on November 18 in Anaheim, California, but it was branded as a non-localized event, "Minecamp Earth," to be streamed live all over the world for what the

[34] "Minecamp" and "Better Than Minecamp" are pseudonyms.

advertising copy asserted was the most inclusive Minecamp yet. In response, BTM organizers playfully labeled the 2017 event "BTM Moon," discursively positioning it outside of Mojang's influence. Accordingly, the event website featured a sparkly pink background of stars and cartoon crescent-moons, and a MIDI file playing the theme from *Sailor Moon*. The in-game event space was not, however, a lunar landscape: copygirl, one of the event organizers, had a different vision, which she wrote about in an email to participants:

> Though called "Moon," this BTM does unfortunately not take place on the moon. No. We're going to build in SPAAAAAAAAACCCEE![35] (I like space.) The idea is that the convention center is going to be a space station, with the mod booths ideally being spaceships docked to it.

I initially planned to attend the convention as an ordinary guest, hoping to observe and solicit interviews with modders. However, when I contacted the organizers for permission, they invited me to set up a booth of my own.

This chapter provides an ethnographic account based on my participant-observation at BTM Moon, beginning with the initial convention setup, and ending with a rather explosive conclusion. In between, I tour the convention grounds, taking a look at spatial logistics; listen to a celebrity modder opine on the state of the art; and sit in on a lampooning-session of the Minecamp livestream. All throughout, I am likely to encounter playful chaos around every corner. While Chapter 4 described modding as a process of settling Minecraft, what follows here is a description of the ways in which it is still, in part, quite unsettled.

5.1 Preparing for the convention: Adventures in building and chaos

I started exploring and planning my booth about two weeks before the event. When I first arrived, I beheld a world that had already been extensively built, but nobody else was around. True to copygirl's description, a vast, metallic conglomeration hung amid a starry void (Figure 5-1,

[35] This is likely a reference to a line from the geek-classic computer game *Portal 2*.

below), with a sun, moon, and square-shaped Earth revolving slowly in the distance. This addition of an Earth-seen-from-space was the first indicator of the modded nature of the server.

Figure 5-1. A view of part of the unfinished space station. (Screenshot by N. Watson.)

Figure 5-2. BTM demo booth. This large ship, attached to the station via a walkway, is home to one mod's demo booth. The Earth is partially visible in the background. (Screenshot by N. Watson.)

It seems I had arrived during a construction lull. Most of the world's megastructures had been built at some previous time, including a bulky metal skeleton, two large glass spheres, and several attached spaceships of different visual themes (Figure 5-2, above); but many individual booths were yet in a half-finished state, to be completed in a building frenzy during the final week. I was, as expected, in Creative Mode, which meant I could build without constraints, and also that I could fly around like an astronaut with a jet pack.

I could not initially figure out where to build my own booth. Copygirl had spoken of a "white platform on top of the 'modules'" but I never found it. I looked for holes that might be interpreted as docking ports, but most of the main structure was either sealed shut or already had ships attached. There was a blocky white module with obvious docking ports hanging some distance above the rest of the station, but this seemed remote and undeveloped compared to the rest of the world. Afraid of breaking existing creations or intruding on someone else's booth space, I headed over to the secret IRC CHANNEL for BTM staff and presenters, seeking clarification. The channel topic was set to warn off the general public, establishing a clear difference between the organizer's room and the public discussion room #BTM, and implying that some BTM demonstrations were intended to be surprises:

> The Secret Channel | If you don't want leaks, STAY OUT! If you're not whitelisted on the server[36] and haven't gotten an OK from me, you probably shouldn't be here at this time. No hard feelings <3

Copygirl's introductory email to presenters had informed us all about the organizer's channel, so I was one of those who had legitimate business being there. Despite dozens of chatters present, nobody was speaking. After almost 15 minutes of silence, during which I continued to fly around the space station looking for a good building spot, I launched into my question:

[36] A Minecraft server can be configured to only allow logins from people who are listed in a "whitelist" file. In this way, access to the BTM server was restricted to organizers and booth-operators during the setup phase. When the convention officially began, the server setting was changed to ignore the whitelist and allow access to all comers.

"So… these docking ports I'm supposed to attach my booth to… are the black circle things on the white cube things?"

"Yeah," replied Tothor, copygirl's busy co-organizer, about two minutes later. Where copygirl handled most of the in-game building and general administration and organization, Tothor was responsible for the modpack: making sure that all of the presenters' various mods, which had to be installed server-side, were functioning correctly and playing nicely together. It seemed my question was well-timed, because Tothor had just been checking in with everyone before resetting the server program—a procedure that has to be carried out any time a mod is added, removed, or tweaked, and which unfortunately kicks everyone off the server for several minutes. Thus, as a courtesy, Tothor was checking to make sure that nobody was in the middle of something: "Is everyone okay if I update the server now?"

I explained that as the only person currently logged in, I was fine with a server reset, and followed up seeking further clarification on my question: "And I can just pick a circle anywhere I like that doesn't already have someone's stuff on it?"

"Yeah, that sounds about right."

I realized later that Tothor's response subtly highlighted the division of labour: copygirl was responsible for building policy, so Tothor was not so much instructing me as remarking that, based on his knowledge of what copygirl wanted, what I had proposed "sounds about right."

Having found no other open black circles, I began to build on the aforementioned blocky white frame that was detached from the rest of the station. I supposed that I would need to build a docked space ship—I did not realize until much later that many of the booths were simply rooms within the station's main body. Lacking artistic skills in original ship design, I settled on building a replica of the Spathi Discriminator from the video game *Star Control II* (Toys for Bob, 1992). Consisting of multiple coloured spheres connected by grey struts for a "molecule" appearance, this design would be simple to build, yet perhaps interesting enough to attract attention from convention-goers. Attention-grabbing was to be the primary function of my booth anyway, since I did not

actually need the interior space to demonstrate modded blocks or machines: the "content" of my display would of a series of text-based signs, posted at the entrance, which briefly described my research, invited readers to participate, and provided a URL for more information. Within a couple of hours, I had constructed a space-based field station (Figure 5-3, below) and populated it with an assortment of flashy consoles and whatever other futuristic widgets I could find, chosen from the wide selection of available technology-themed mod blocks based solely on appearance, and without regard for actual in-game function.

Figure 5-3. The Spathi Discriminator, my initial on-site ethnographic field station, was docked to an unfinished part of the convention grounds. (Screenshot by N. Watson.)

Turns out that I did it wrong.

Three days before the convention, I logged into the server to see how preparations were going. This time, half a dozen people were present, many of them putting frantic final touches on their booths, or troubleshooting unforeseen conflicts between mods. Copygirl greeted me right away: "Hi nwatson. I'm probably going to move your booth. The top part of the station it was built next to is out of commission, I didn't remove it completely because there was some stuff in it." But before she could say more, a bug in one of the mods caused a server-wide crash.

I followed up with copygirl on IRC:

> <copygirl> I want to ideally put it somewhere people will walk more often. Spawn[37] would be nice. But with a lot of stuff already built there is little place.
>
> <copygirl> Worst case, I know where to put it though. But it'll be right at the end of a possibly less used hallway (though close to my booth.. and the DJ Flamin' GO place! *[Figure 5-4**Error! Reference source not found.**]*

Figure 5-4. DJ Flamin' GO, I would later learn, had been BTM's de facto mascot since at least 2016. The devices around him can be used to play music tracks in-world. *Flamingo*, the mod that provides the ornamental ornithoids, was developed by copygirl. (Screenshot by copygirl, used with permission.)

When we finally got back in-game, copygirl flew over to my ship and, using a combination of commands offered through the WorldEdit mod, copy-pasted my ship to a new spot, which she described as a "prime location" right next to the room where panels would be held. WorldEdit requires that the region to be copied be defined by a rectilinear volume. The easiest way to make a selection is to click on two existing blocks that represent opposing corners of the region, but my ship was laid out in a way that its furthest extremities were not near the corners. Copygirl therefore had to add artificial extensions to get the correct positions of corner blocks. For this, she built struts off of

[37] "SPAWN" means the area where new players would appear upon "spawning," or joining the server for the first time.

the ship's hull using the block she happened to be holding—black steel-frame stairs (Figure 5-5, below).

Sadly, when pasted into its new location, my ship was too wide to fit. The green module ended up erasing a portion of a neighbouring booth. Fortunately, any WorldEdit change can be easily reverted. I offered at this point to redesign my booth, but copygirl was determined to make it fit. Using a combination of copy/paste operations, she shortened the connective struts and tried again, but it was still too wide.

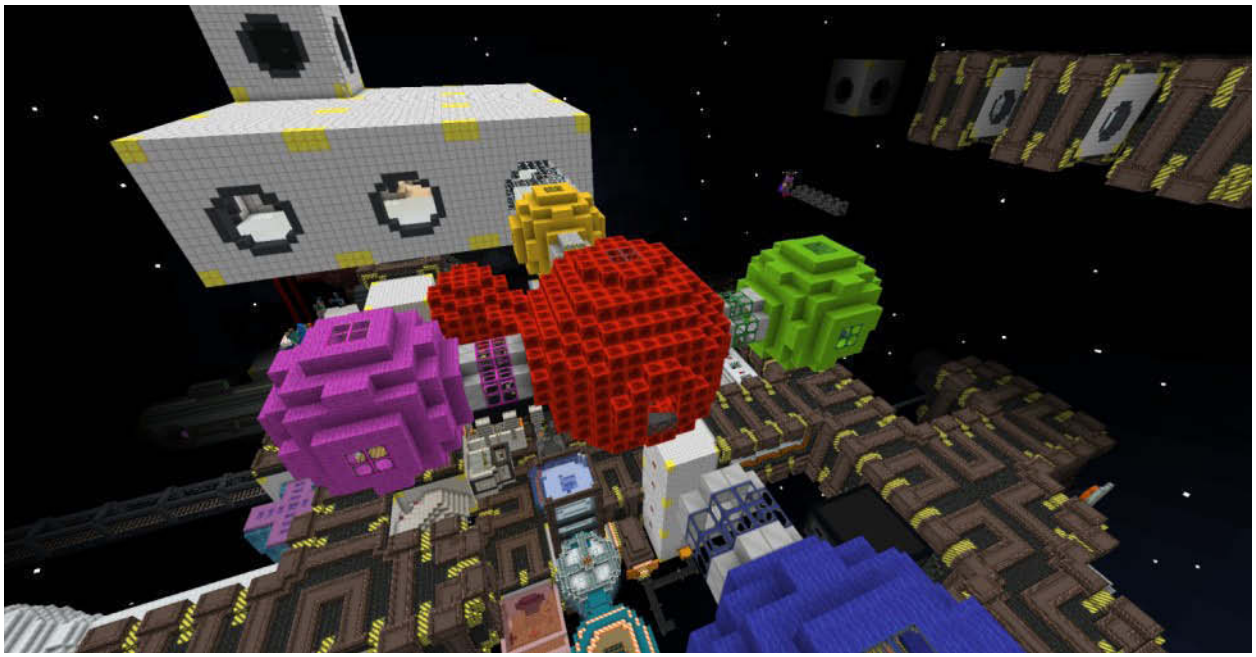


Figure 5-5. WorldEdit copy/paste operations at BTM. Copygirl, seen as a small figure just above the green pod, uses a temporary scaffold of blocks to select the correct volume for a copy/paste operation. In this image, the connectors running to the green and magenta pods have been shortened, while work on the other two struts is still in progress. The operation took about five minutes in total. (Screenshot by N. Watson.)

Copygirl probably would have kept trying but she was disconnected by another server restart and was prevented from rejoining due to some technical issues. I resolved to build a new booth in the spot copygirl had shown me – something that sprawled less. Copygirl suggested I could ask for help from the "talented" members of the FuturePack mod team, who had been hanging out on the server around that time.

When I did rebuild, I went with a rough approximation of the Hubble Space Telescope, hollowed out inside to create room for my signs and an "office," consisting of a desk, some tables

and benches, and a few bookcases. I used shiny metallic blocks for the exterior and mirrors, and for the solar arrays I selected one of the several solar panel variants offered by the many technology-themed mods. It was during construction that Velo, one of the FuturePack demonstrators, flew over to have a look. Velo had taken an early interest in my Spathi Discriminator, and was now offering to help out with my telescope design. For lighting, I had put in sea lanterns from vanilla Minecraft, which arguably clashed with the rest of the visual theme, but were easy to implement because they glowed without the need for a power source or wiring. Velo offered to put in futuristic-looking "neon lamps" as a more thematically-appropriate alternative, to which I readily agreed since I would not have to learn how to do the wiring myself on a tight schedule.

To my alarm, Velo began punching out a section of the telescope's wall to lay an electrical conduit (Figure 5-6, below). "Can we run the wires out one block?" I asked. Not sure if he understood what I meant, given that he was not a native English speaker, I flew up and replaced one of the blocks Velo had removed, then placed a small length of conduit on the exterior side. I figured I could tolerate holes in my telescope in a few spots, but not all along its length!

Figure 5-6. "Wait and drink a tea ^^ " —Velo punches out a section of my Space Telescope in order to lay power conduits for the neon lamps. (Screenshot by N. Watson)

Velo's response was to tell me to "wait and drink a tea ^^" (with the double-caret smiley suggesting amusement rather than annoyance). As I stood back, he removed the blocks I had just

placed, finished laying the wiring, and then replaced the shiny metal chassis blocks directly on the conduit, explaining that they were "holograms." The end result was that the outside of my telescope looked exactly as it had before, with the wires completely hidden from view. I had, however, committed a cardinal sin of the ethnographer: impatience.

Having finished my new booth, I figured that I ought to disassemble my old one. I started to break it up using a combination of punching and explosives, but here Velo volunteered to help again. He appeared eager to impress with FuturePack mod functionality. He asked if he could deploy a "little friend," a tiny drone that would be instructed to mine the blocks in the area one by one. In the end, the drone was probably slower than what I could have done by hand, but it was fully automated.

The "wait and drink a tea" incident primed me to respond with more bewilderment and fascination than irritation when, returning to my telescope a day later (and on the eve of the convention), I saw that it had been completely redecorated. The shiny metallic exterior had been retiled with a dull, modern grey, and sloped molding covered many of the corners (Figure 5-7, below). The interior mirror was replaced with a bank of control panels.

Figure 5-7. Original (left) and modified (right) telescope exteriors. (Screenshots by N. Watson.)

At first I supposed Velo might have taken extra initiative based on how I had been receptive to his previous assistance, but Velo had always asked me before doing anything. I found a brief explanation in the form of two signs posted on the wall near the entrance:

"Made the shuttle more pleasing to the eye. –Olstead"

While the new design arguably *was* more pleasing to the eye, it certainly no longer looked like a facsimile of the Hubble Telescope, though the note suggested that Olstead had not even realized my intent, since he referred to it as a "shuttle." It is just as well that I had no time to revert the design, since I was not sure if I ought to. I reasoned that Olstead's action may have been more than just an unsolicited favour: maybe I had simply run up against the local homeowner's covenant, so to speak. Perhaps my crude design had been updated to meet community standards or blend more smoothly with the theming of the rest of the neighbourhood, particularly given the importance that convention organizers and presenters might place on the appearance of booths occupying such prime real estate. Copygirl provided some additional insight, suggesting that this sort of thing is not out of the ordinary. "We didn't set any rules for what's okay to touch by builders and what isn't. Zoll[38] would probably say that chaos makes BTM what it is." She noted that Olstead "was one of the general builders helping out with everything." To this I would add that the location of my booth, where builders like Olstead would see it, made it a probable target for renovation. It is possible that Olstead did not even realize that it was an individual demo booth, and considered it part of the central station infrastructure.

In general, BTM builders are not shy about taking their own initiative to fix anything that they perceive as deficient. With a tight schedule and lots to do, there isn't always time for the general builders to consult with each other or with booth owners before making changes. The first BTM, according to copygirl, was "quite mad," being set up in less than 18 hours. This year there was more than a month of preparation time, but this "doesn't change the fact that every BTM still feels like all the difficult work is crammed into 18 hours. (It's a curse!)" The result is the kind of emergent bricolage that characterizes the BTM station. Copygirl related how one year, she built a panel room that "apparently wasn't big enough, so people built a bigger one next to it. In my opinion that was way too large. I did have some bad feelings over it but hopefully kept them mostly to myself."

[38] Zoll was the original founder of BTM, and chief organizer in previous years. He was not directly involved in the organization of BTM Moon, which was run by copygirl and Tothor. Zoll did deliver the keynote presentation, however (described later in this chapter).

In addition to being chaotic, BTM work is decidedly playful, with the madness of modification giving rise to intentional pranks. Copygirl explains:

> There's somehow an ongoing joke with the missing texture[39] within the modding communities that I don't quite get. Can probably be described as a meme nowadays. Like, a purple/black checkered texture. I had to defend one of my booths - was it the first DJ Flamin' GO! booth? - against... I believe it might've been Provo and Foss.. From being taken over by chiseled[40] (or real?) missing texture blocks. In the end the VIP section above had carpet that was checkered purple and black.

There were no major pranks of that kind this year, although some people (unintentionally) "built things in ways that made things a bit difficult." The intentions of booth owners are sometimes out of sync with the organizers' plans. For instance, flight was supposed to be limited to booth owners and staff, with guests being gravity-bound, but "one booth author made the mistake of giving out brooms" via an in-booth vending machine. These magical broomsticks would have allowed players to fly all over, but copygirl caught the issue before the convention started and "quickly put a piece of bedrock there so hell doesn't break loose."

These accounts of BTM setup suggest a sort of "working anarchy" that emulates the kind seen and celebrated in the rhetoric of the indie game studio (Dyer-Witheford and de Peuter, 2009, pp. 41-42). It is a kind of tactical approach to project work that supposedly trims the overhead costs of rationalized management, while simultaneously encouraging innovation through emergence of the unexpected in the coincidental interactions of many moving people and parts. BTM setup of course also exhibited another quality of game developers' labour dynamics, one which has practically become a cliché: "crunch time" (Dyer-Witheford and de Peuter, 2009, p. 56). A key difference is that BTM's participants are not being compelled to work under these conditions for a

[39] A missing texture means that the Minecraft program was unable to find the image file that is supposed to be displayed on the face of a block. When this happens, a magenta-and-black checker pattern is displayed as a placeholder.

[40] There are multiple "chisel" mods that allow one to change the appearance (texture) of a block without modifying its other properties or functions. If the pranksters chiseled in missing textures, it means they did not actually destroy the existing blocks. In the case of "real" missing texture blocks, the existing ones were actually replaced with some other block that did not have a proper texture mapping associated with it.

living. Instead, this creative chaos suggests a playful cultural production for its own sake—what Pearce (2006), in her deconstruction of the "magic circle"-based work/leisure dichotomy, calls "productive play." Dyer-Witheford and de Peuter insinuate that rhetorics about playful work can serve the interests of exploitative companies (2009, p. 55), and Kücklich (2005) has extended this critique of playbour to modding, but the productive play of BTM appears to be authentic. Anyway, it would be far-fetched to claim that any corporate entity is exploiting the BTM crowd. While Mojang may benefit *generally* from a vibrant Minecraft modding scene, BTM itself resists the Mojang branding narrative and evades enclosure into a capitalist logic. I will return to this matter, but first it is necessary to describe the setting in which all of this anarchic productive play takes place.

5.2 Spatial organization and logistics in "SPAAAAAAAAACCCEE!"

Given the "madness" inherent in the building process, BTM's station layout was chaotic—almost maze-like. Maintaining a sense of direction and finding one's way to specific booths was difficult enough that attendees remarked on multiple occasions that they needed maps. Although no official maps were available, booth placement was not actually arbitrary and some provision was made to facilitate navigation. The station—which attendees more often referred to as "the convention centre"—was divided into zones, with mods of similar type being grouped together. Copygirl says that she "kinda sorta decided on these [groupings] and nobody really said anything against it. (Yay madness.)" The zoning provides some insight into how modders—or at least copygirl—think about mods in terms of genres and categories.

The descriptions that follow use directional terms like north/south/east/west, and up/down. These are based on the three-dimensional Cartesian grid on which every Minecraft world's blocks are plotted (see Subsection 6.3.2), so they still have meaning despite the outer-space theme. The station's environs were accomplished by means of a mod that removes the normal terrain, renders stars in the lower half of the VOID, and adds an Earth that rotates around the player's position along with the sun and moon. East and west are defined by the direction in which the sun

and moon are moving, while up and down correspond to the direction of normal gravity (which applies throughout the station, except in certain spots that have their own local gravitational fields).

5.2.1 Industrial Hub

The Industrial Hub, occupying much of the southern half of the station, was home to technology-themed mods (with the exception of certain computing mods, which had their own zone). Copygirl described this as a place for "mechanical and industrial-feeling mods," emphasizing theme or "feel" over game mechanics. The themes of industrial tech mods run the gamut from high-tech, futuristic contraptions to low-tech, steam-age machinery, but these divergent motifs are grouped together, which is unsurprising given that their different themes do *not* map onto different game mechanics.

However, the "Industrial Hub" grouping does not account for the fact that there are at least two very different subcategories into which industrial mods can be grouped in terms of their impact on gameplay, irrespective of theme. I call these *gadget mods* and *tech tree mods*. Gadget mods add some technological devices, useful for performing tasks that are already part of the game—whether in vanilla or added by some other mod. An example from BTM's Industrial Hub is *Turret Mod Rebirth* (Figure 5-8, below), which adds automated gun turrets that can shoot monsters, protecting players and providing a means to automatically farm MOB (monster) loot. The *context* for turret use (i.e. the presence of hostile mobs) already exists in-game, as do the materials needed to build them. In contrast, tech tree mods add entirely new experiential threads of gameplay progression: in a typical implementation of the general "tech tree" concept in game design, one needs to start by building simple devices, which gradually enable one to bootstrap (or "tech up") to ever more complex and powerful machinery. *IndustrialCraft 2* (IC2) is an example of a popular tech tree mod found at BTM. It adds a variety of engines, generators, batteries, wires, transformers, and material processors, only the most basic of which can be built from vanilla components—everything else depends on some earlier tier of IC2 tech progression. In general, tech tree mods are focused on

"automating all the things"—they increase the volume and efficiency of a player's resource processing, while cutting down on repetitive labour. However, it takes a significant amount of labour to reach the highest tiers of automation and optimization in the first place. Tech tree mods approach what Scacchi (2011) calls *total conversions*, but fall short because most of them are additive rather than substitutive. In fact, a true total conversion that replaces existing game mechanics with its own, like Card1null's *Biolithic* mod (see Subsection 7.4.1), would likely not be compatible with the rest of the modpack.

Figure 5-8. The BTM demo booth for the gadget mod *Turret Mod Rebirth* displays several of the turrets that can be used with the mod, accompanied by explanatory labels. (Screenshot by N. Watson.)

5.2.2 Magical District

Southwest of the spawn plaza (that is, the station "lobby") lay the Magical District, home to mods that deal in magical and mystical forces. In terms of gameplay mechanics, these are often isomorphic to industrial mods, with both gadget and tech tree subtypes. In fact, since few industrial mods can make any credible claim to scientific realism anyway, the distinction between technology and magic is a matter of theming and lore. Arthur C. Clarke famously claimed that any sufficiently advanced technology is indistinguishable from magic, but in Minecraft mods at least, the difference

is that technology whirrs and beeps, while magic sparkles and chimes.[41] In fact, the official documentation for *Botania* (not present at BTM), a mod based around magical flora, insists that it is best understood as a tech mod.

Copygirl directly acknowledges the tech-leanings of many magic mods: "I think we had fewer 'true' magic-themed mods but there were some techy mods that had a bit of a magic style to them so I suggested for them to also go there." Here she is again emphasizing surface characteristics like "style" over underlying game design principles in determining mod organization.

5.2.3 Nature Dome

The Nature Dome, west of the spawn plaza, was a large glass sphere designed to house various expansive terrestrial environments. Significant mods in this zone included *Twilight Forest* (TF) and *Forestry* (the latter was not actually inside the sphere, but on an attached ship). TF is large, magic-themed mod based on the exploration (and exploitation) of a mystical alternate dimension, while Forestry is an industrial mod focusing on agriculture, arboriculture, apiculture, and mycology. Despite this difference, the two mods find themselves together in the Nature Dome. According to copygirl, this separate category arose out of the TF team's special request for a large, outdoor space. Although TF has some tech-tree aspects, its focus on atmosphere and adventure made it unsuitable for the Magical District, since its functions (including fifty-metre-tall trees and towers) could not be effectively demonstrated through the placement of a few selected devices in a room. Instead, the TF team wanted to replicate a facsimile of the twilight forest environment within the station. Other mods that dealt more with outdoor environments than with machines were "tacked on."

5.2.4 OpenComputers

Attached to the east side of the spawn plaza was the OpenComputers zone. This entire region was set aside for just one family of mods. Although vanilla Minecraft technically makes it

[41] Magic-themed mods are also more likely to add new explorable dungeons or alternate dimensions based around mystical forces, but even this is not magic-specific, as the tech mod GalactiCraft allows players to build rocket ships to travel to new planets.

possible to build fully-featured computers out of redstone electronics, problems of physical size and speed make such machines of limited practical use. OpenComputers fulfills the implicit promise of redstone by bringing fully-featured, programmable virtual computing systems into Minecraft worlds. These virtual machines can also interface with the file system and resources of the physical server running Minecraft. There are *many* add-on mods that contribute new gadgetry to the OpenComputers ecosystem, which accounts for why it was allocated an entire multi-booth zone. OpenComputers is arguably a meta-mod, because it provides a means by which new features can be added to Minecraft *without* further modifying Minecraft's codebase. Instead, the virtual computers themselves are programmed in-game. Many of the demos in the OpenComputers zone, therefore, were not Minecraft mods per se, but rather programs that players had developed for OpenComputers platforms—for instance, a functioning *Tetris*-like game which one attendee, Amadornes, played on his Twitch stream while casually conversing with a group of other modders (Figure 5-9).

Figure 5-9. Left: Amadornes plays an OpenComputers-based *Tetris*-clone on his stream. Right: My own view of the in-world room in which Amadornes is playing Tetris.

OpenComputers is not the only computer mod, its chief competitor being ComputerCraft (not present at BTM). However, OpenComputers' fans cite its open-source nature (hence, the name) as its key strength, and the likely reason why it boasts so many third-party add-ons. In fact, during the Tetris playing session, modders were speaking quite critically of ComputerCraft, trading stories about security exploits that could allow players in-game to use ComputerCraft devices to compromise the Minecraft server host itself. The conversation also suggests that many modders,

even those not directly involved in OpenComputers projects, are computer hobbyists who take an interest in computing and programming topics well beyond the scope of Minecraft itself.

5.2.5 The Fun Zone

The Fun Zone, also east of spawn, seems to have been conceived as an area for mods that were considered playfully frivolous, featured interactive mini-games, or otherwise did not seem to fit anywhere else—including mods offering miscellaneous accessories such as (functional) backpacks and (whimsical) hats. Dispensers offering freebie samples were characteristic of these booths (Figure 5-10).

Figure 5-10. The Fun Zone demo booth for copygirl's *WearableBackpacks* offers instructions, freebies, and a menagerie of backpack-wearing mobs. (Screenshot by N. Watson.)

Like the Nature Dome, the Fun Zone was another case of a single "big booth" request leading to the creation of a new zone with other mods tacked on. DJ Flamin' GO's music booth and accompanying "dance" floor could be found in this area, as well as a separate "Disco Booth" which offered its own music players, playlists, and flashy lights. A racetrack in this zone (the cause of the aforementioned big booth request) promised opportunities to race mod-contributed cars against other convention guests, but instructional signs explained that a qualified attendant was needed to

operate the track. I visited several times, hoping to try out the cars, but I always seemed to check in at times when no attendant was present.

One corner of the Fun Zone proclaimed itself to be the exact opposite, the "No Fun Zone" (with the subtitle, "Why are you even here?"). This was a playful means by which BTM contributor Vexatos marked off the non-interactive informational kiosk for the mod Conventional. A simple alcove featured a series of text signs explaining that Conventional allowed server administrators to designate regions of a world for restricted interactions, allowing different players to break or interact with blocks in different ways. This mod was made specifically as logistical support for BTM, so that booth owners could modify their own booths, while guests would be prevented from breaking things. As a behind-the-scenes logistical mod, Conventional seems to have been featured in the Fun Zone simply because it did not belong anywhere else.

5.2.6 Centrepieces

The spawn plaza, being the lobby-like entry point where newcomers would appear, was a wide, square-shaped room in the northern middle part of the station. The middle of the floor was decorated with a large red "jewel", and three connecting corridors led off to the various convention zones. Likely owing to its central location (despite being north of the geometric middle, it was a topological nexus between zones due to the corridor layout), it became a site of frequent social activity. At least a handful of people were hanging around chatting at most times during the weekend. On several occasions, I observed people exploring together in small groups, who would pause on their way through the spawn plaza to discuss where to head next.

A large digital clock display, supported by mod-supplied computing blocks, counted the days, hours, minutes, and seconds since the convention opened—previously it had counted down towards the opening moment, for the benefit (and anxiety) of exhibitors still working up to the last minute. Another screen displayed a running count of the number of bugs and server crashes. These informational displays were rearranged somewhat over time, being moved from one wall to another.

As of 10:20 PM on Day 2, this display reported eight server crashes, one complaint about the station layout, and 50 bug "reports"—stated as the number of "times people screamed: 'BUG!'," making it unclear whether this was an actual bug count or just a running gag about attendee complaints (Figure 5-11, below). Plentiful bugs could be counted on anyway—after all, some modders asserted in conversation that BTM actually stands for "Bug Testing Marathon." By Day 3, the clock was no longer reporting time elapsed since launch, but rather uptime since the last server crash. A new display appeared, enumerating "segmentation faults" (a kind of memory access error) and "death by train" (a reference to the Immersive Railroading trains—see Subsection 5.2.8, below). The final count I was able to observe, before the spawn plaza was destroyed in a celebratory post-convention bombing, was 14 crashes, 52 "BUG!" reports, 3 complaints about layout, 2 segmentation faults, and "5?" [*sic.*] deaths by train.

Figure 5-11. The BTM crash/bug-counter screen on Day 2. (Screenshot by N. Watson.)

One wall of the spawn plaza featured four dispensers for free "Fruit Phones" and related products. The phones were not fruit-themed: the name was a not-so-subtle allusion to a certain popular cell phone brand, and an old-style rainbow-coloured imitation of the Apple Computer logo was appropriated for the banner. The actual function of these phones—not explained on the banner—was to let people identify the blocks they were looking at, by displaying the block's name

and the mod to which it belongs on the user's HUD. This is an example of a type of infrastructure mod that has become necessary thanks to the proliferation (and concurrent use) of hundreds of mods adding thousands of new blocks. Increasingly, such strategic utilities are deployed in modpacks as a way of managing the explosion of features well beyond what Minecraft was originally designed to handle. In the context of BTM, these block-identifier tools also help attendees to discover new mods that they would like to try in their own Minecraft worlds.

WAILA (What Am I Looking At) seems to be the most commonly used mod for this purpose, but it is a client-side feature that requires no special in-game equipment. The Fruit Phones are different because they encapsulate this functionality in a game item which—importantly—becomes part of one's avatar appearance. Four gadgets are offered: the phone, with a portrait-oriented display, which needs to be held in the player's hand to work and displays text on the item proper; the tablet, which is like the phone but larger and landscape-oriented; the glasses, which are wearable, always active, displaying floating HUD text, and meant to evoke the appearance of Google Glass; and the contact lenses, which provide the functionality without being visible on the avatar.

A couple of miscellaneous booths were attached to the hallways close to the spawn plaza. One of these, the "Flamingo LOVE" booth (not to be confused with DJ Flamin' GO's music booth down in the fun zone), contained two waterfall fountains and about two dozen ornamental pink flamingos which bob back and forth when struck—an homage to copygirl's *Flamingo* mod and the ongoing BTM meme it created. (The BTM client loading screen also features a bouncing pink flamingo.)

A generic, square-shaped empty booth room was located just off of the spawn plaza along the corridor leading to the Nature Dome. This ended up being claimed by Amadornes, the official Twitch streamer of BTM, to demo his "Super Sekrit Mod" (SSM) which he had been working on for several months, but had not yet fully unveiled. The way in which Amadornes went about claiming the space is illustrative of the decentralized communication and just-in-time coordination of BTM

operations. With was no single all-knowing booth allocator or centralized database of booth ownership to consult, he had to resort to the tactical response of leaving a memo that other interested parties would hopefully see on time—although the fact that he did so while streaming to a substantial audience probably increased the chance that his message would reach the right person. He posted a pair of vanilla Minecraft signs that read:

> If this booth is taken, please let me know. Otherwise, I'll take it tomorrow for the SSM :)
> –ama

Amadornes' memo did not stop his booth from being appropriated as a chaotic playground for the rest of Day 1. It probably started with a small number of people using the space as a temporary sandbox to hang out and tinker with various blocks, but since it was on a high-traffic corridor, these activities apparently attracted the attention of more and more passers-by until the room was crowded with an eclectic array of crates, tanks, redstone devices, ornamental flamingos, and players flying around pelting each other with throwable tomatoes.

The Panel Room was directly south of the spawn plaza, but one floor up with gravity-shaft access, leaving a wide, empty room underneath. Despite its size and high volume of traffic, this space was mostly empty and featureless, and attendees seldom paused here for long—except to check out my field station, which was attached here.

5.2.7 The Panel Room

A great block of glass and metal protruding from the top of the station's central hub, the panel room merits a section to itself, particularly as a case study in the relationship between decoration and infrastructure at BTM (Figure 5-12, below).

This auditorium periodically became the locus of frenetic activity when panels and presentations were taking place, although these were few and far-between, and the room was sparsely populated the rest of the time. Access was achieved by means of two gravity shafts on opposite sides of the room. The floor was slightly inclined and lined with theatre-style seats—84 in

total. On the ceiling and three sides, large glass panes offered vistas of the station and the stars beyond, while the fourth side was dominated by the stage and display screen. The screen itself was a large OpenComputers monitor that could display images and videos. A curtain of wool blocks was used to cover or reveal the stage as necessary. Switches to operate the curtains and lights were tucked away stage-left.

Figure 5-12. The panel room gets crowded as people wait for the opening keynote to begin. (Screenshot by N. Watson.)

Some amenities were offered along the back wall: a kiosk proclaiming "FREE SPACE HATS", giving players a wearable astronaut helmet (cubic, naturally) at the touch of a button; some recessed dispensers providing free food; and trash cans.

This isn't all for decoration—nor is it all for utility: form and function chase each other in circles here. In offering the aesthetic trappings of a real convention, the theatre, with its seats oriented towards a common screen, provides a sense of presence, embodiment, and shared experience that a Twitch video stream could not accomplish. Food kiosks and trash cans act as enriching details. These features serve the *expectation of human embodiment* that, for better or worse, is pervasive in the design of multiplayer games and virtual worlds (Yee, Ellis, and Ducheneault, 2008)—and it becomes all the more important when one is explicitly trying to create a virtual

substitute for the convention-going experience. But these objects are not just for looking at, and the interactions they provide are important to convention logistics.

Thanks to a mod, players can actually sit in the seats via right-click. When seated, the player's location is fixed and their view is automatically rotated to face the screen (the view is not locked; they are still free to look around). Being in the audience means being able to see not only the images and video being displayed on the big screen, but also the avatars of the presenters on stage. In theory, a seated audience on an inclined floor optimizes everyone's view as people are not distributed haphazardly around the room. In practice, many people chose to stand, float, or fly anyway—which of course obscured the view of others behind them. The logistical aspect of seating is therefore not well-defined despite the appearance of some gestures in that direction, and sitting is mostly about embodying the role of convention-goer. This conclusion is reinforced by screenshots of the panel room from BTM 2015 (shown during the 2017 keynote), which reveal more of a classroom theme with rows of seats at long desks, and no incline.

The logistical function of food dispensers is much clearer. At its core, Minecraft remains a game, not a virtual conference engine. Game design assumptions integrated at the lowest level cause Minecraft to resist certain uses. As discussed in Chapter 8, neither Minecraft nor any other platform is truly universal, and they all afford some uses while constraining others. In this case, the structure of game modes means that a player can have either all (Creative Mode) or none (Survival and Adventure Mode) of three traits: immortality, flight, and unrestricted building capability. Since non-builder attendees should not be granted the ability to build or destroy parts of the station at will, it is necessary to keep them in Survival Mode, which means that they are also subject to hunger and need to eat periodically to avoid death. Infrastructure is needed to ensure that attendees always have access to food, and this is accomplished with dispensers in key locations that freely supply food items on demand. The fact that the food was in the form of mod-created mini-sandwiches, as opposed to some vanilla Minecraft food like pork chops, is a thematic reinforcement of the convention motif.

The trash cans also have an important logistical purpose. The organizers realized that people would probably accrue a lot of freebie items from touring the booths, and they would want to discard some of these items. The vanilla-Minecraft way to get rid of items is to throw them on the ground, as they will eventually disappear, but this causes considerable lag on an already-busy server, and in high-traffic areas those items are likely to end up in someone else's inventory (items are picked up automatically on contact, whether one wants them or not). The trash cans, found all over the station, provide an efficient and lag-free way to get rid of stuff. Players can interact with the bin as they would with other in-world inventory items such as storage chests, but anything that is moved into a trash can's inventory is summarily deleted. All of this also highlights BTM's deep dependence on mods, not just as content to show off, but for core functions.

5.2.8 Transportation and orienteering

Encircling the entire station, the designer of *Immersive Railroads* had lain a track with a ride-able passenger train making a regular circuit. The train passed through multiple labeled stations attached to the different zones, making it seem akin to tram circuits at theme parks: more than mere vehicles, these trams act as automated tour guides showing off different portions of the park, while also drawing attention to the experience of the ride itself. But the BTM rail loop did not seem to accomplish those functions. Attendees did not generally seem to be using it as a way of getting around, preferring to walk through the interior of the structure. The train was instead used largely for recreational purposes, as after the Immersive Railroading panel session a group of attendees decided they wanted to see if they could crash the trains into each other—a debacle that is described in Section 5.5.

The space station mostly sprawled along the horizontal axes, but did have three vertically stacked floors. Instead of elevators or ladders, of which copygirl admitted to being "not a huge fan," traffic between floors was accomplished by means of gravity-switching shafts. If I approached one of these shafts, I would find myself re-oriented so I was walking on the wall, which allowed me to

travel the length of the shaft. At the end, attempting to approach the now-vertical floor would cause me to revert to normal gravity. The shafts did not always work, however, or the gravity switch had a lag, and I observed several guests fall to their deaths! (Dying meant dropping all of one's inventory, and getting sent back to the spawn plaza.) I was fortunate to still be in Creative Mode, giving me immunity to fall damage.

The gravity shafts were used for their thematic consonance with the space setting, and to show off yet another mod feature, rather than for logistical expediency. In fact, they contributed to spatial confusion by causing temporary directional disorientation, and also caused lag because of the server sub-processes required to detect when players crossed a gravity boundary. On the second day of the convention, when the Minecamp stream was being displayed on a screen in-world, administrators had to disable the gravity shafts to reduce lag, temporarily replacing them with ordinary, low-tech ladders[42] from vanilla Minecraft.

As mentioned, there were no maps (although I have provided one in Figure 5-13 on page 140), and guests had some difficulty finding their way around the winding corridors. One guest wished aloud for shopping-mall-style map kiosks with "You are here" indicators, while another admonished the organizers on Reddit:

> So, um, a tip for the next convention; use a proper floor plan. I keep walking in circles, and seeing just a few mods over and over.
> The mod messing with gravity *really* does not help, nor does all the teleporters you accidentally walk into and end up in some random location.
> A proper floor plan / map of the world would help.[43]

These attendees wished for the unifying, top-down, strategically shaped experience that de Certeau (1984, loc. 1421) likened to viewing Manhattan from atop the World Trade Center. However, a different spatial strategy—"street level" but not tactical—had been implemented by

[42] One advantage that gravity shafts do have over ladders is that when they are working well, the gravity shafts allow for faster movement between floors and can accommodate greater simultaneous volume than ladders can.

[43] http://web.archive.org/web/20190401185223/https://www.reddit.com/r/feedthebeast/comments/7dmgwu/btm_moon_is_upon_us_hang_out_with_your_favorite/dpzggcj/

builders to help guests find their way. The hallways in each zone featured coloured trim specific to that zone, and the corridors between zones had coloured stripes running along the floor, highlighting the paths to the various zones, as is sometimes seen in real-world airports. The stripes were orange for the Industrial Hub, purple for the Magic Zone, dark green for the Nature Dome, lime green for OpenComputers, and magenta for the Fun Zone (a small section of drab brown flooring, and a very short pathfinding stripe, marked the No Fun Zone). An additional length of light blue stripe linked the spawn plaza to the panel room. All six stripe colours (not counting the brown "No Fun Zone") converged at the spawn plaza, orienting newcomers towards each of the zones. In addition to assisting with wayfinding, the trails also acted as expressways by increasing the movement speed of those walking along them.

Spatial logistics were so central to the fabric of the convention that they were even implicated in user communications. BTM used mods to implement *positional audio and text chat*, integrated with the separate Mumble voice chat software. When attendees were connected to the Mumble server while in-world, the volume and apparent direction of others' speech was calculated based on the relative positions of those players in-world. If a person was to my left, I would hear their voice from my left-hand headset earpiece. The further away they were, the quieter they would be, and if they were sufficiently far, I would be unable to hear them at all. Position—but not direction—also affected whether text chat messages were visible, with anything typed in the Minecraft chat prompt being visible only to people in the immediate area (unless a special "shout" command was used to broadcast to the entire server). In addition to creating a sense of realism and localized rapport, the positional system was a logistical strategy to reduce noise and cross-talk in a space that regularly contained between 40 and 60 people over the course of the weekend.

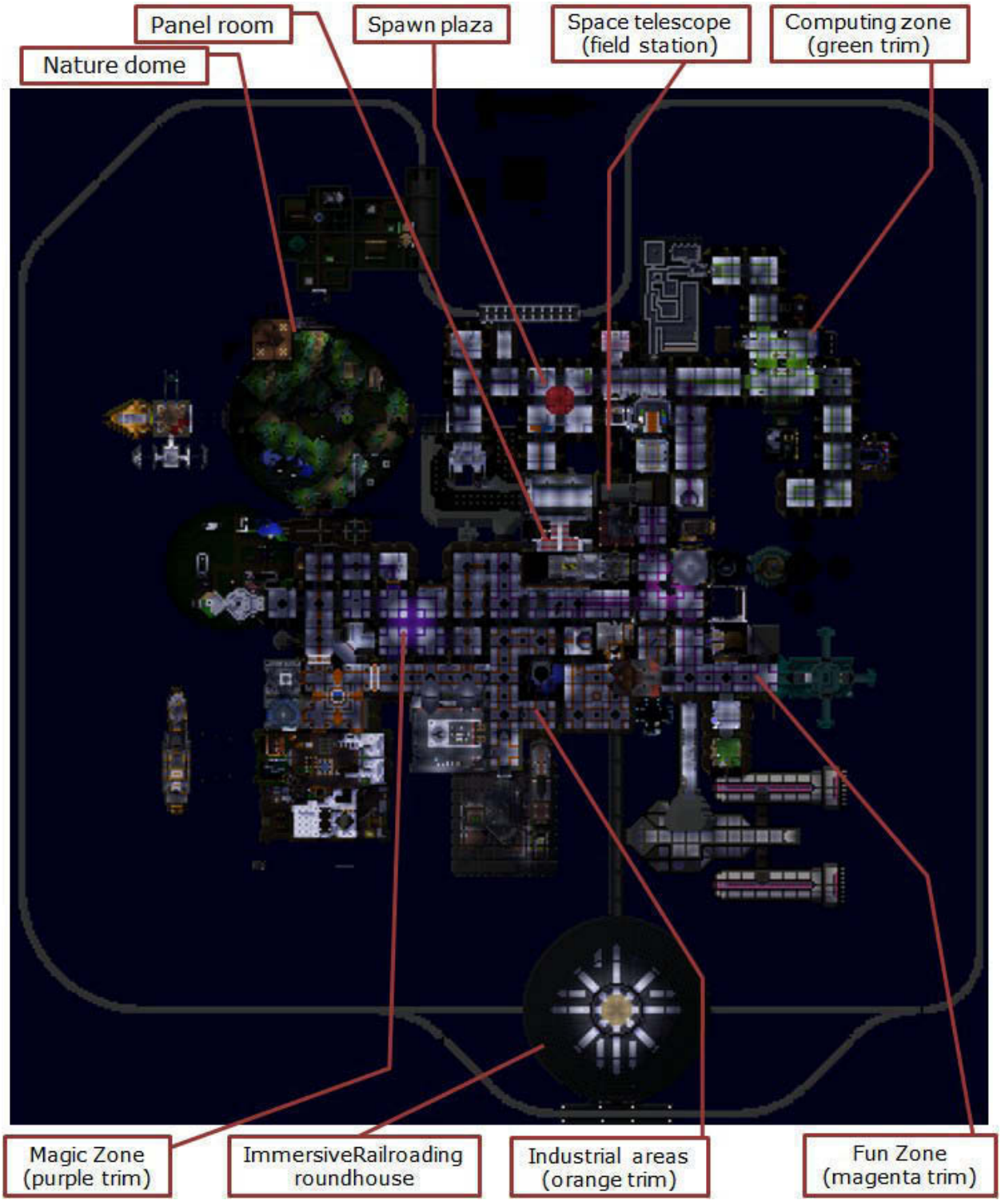


Figure 5-13. Map of BTM Moon station, showing zones of coloured trim, with key locations labeled. As some areas overlap on different floors, not all rooms are represented on this map. The grey loop around the perimeter is the railroad line. (Map compiled by N. Watson from BTM backup files, rendered using *JourneyMap* v.5.5.4 Minecraft mod by Techbrew Interactive.)

5.2.9 My field station

I intended for the space telescope to look vaguely attractive (something with which both Velo and Olstead certainly helped), but it was light on function. Aside from a comfortable-looking office/lounge space behind the main panel/mirror (Figure 5-14, below), the only feature of note was the bank of informational signs I put at the entrance, on which I explained that I was writing a PhD dissertation on Minecraft modding and was seeking interviewees, accompanied by a link to my project website and an email address.

Throughout the convention, I spent 15 minutes here and there on manning the booth, particularly during high-traffic periods when crowds were leaving the nearby panel room following a presentation—just in case anyone wanted to ask me a question in person. However, most of the time I was elsewhere, exploring or observing other attendees, and anyway it seems that the signs themselves were enough to accomplish my goals. I observed several visitors appear to notice the telescope entrance as they were walking through the vast and otherwise-featureless foyer under the panel room. They would walk over, pause at the entrance to read the signs, enter the office area to take a quick look around, and leave again. Evidently I succeeded in attracting attention, which was my objective, but there wasn't much need for people to linger once they had noted the key information.

The location was certainly a boon for getting noticed: the general emptiness of the central foyer likely worked to draw attention to my signs, although it probably also played a role in discouraging people from lingering, since there were other places with more going on—like the spawn plaza, just around the corner from me. Also helping to drive traffic to my telescope were announcements that copygirl, Cervator, and others graciously made in the panel room, encouraging attendees to check out my station and my website. The effort of establishing my own booth certainly paid off for the study: in the ten months prior to BTM, I had succeeded in convincing just one

modder-informant to participate in my research (a few others had seemed like promising leads, but ultimately backed out); within days of BTM Moon's launch, seven more interviewees had signed on.

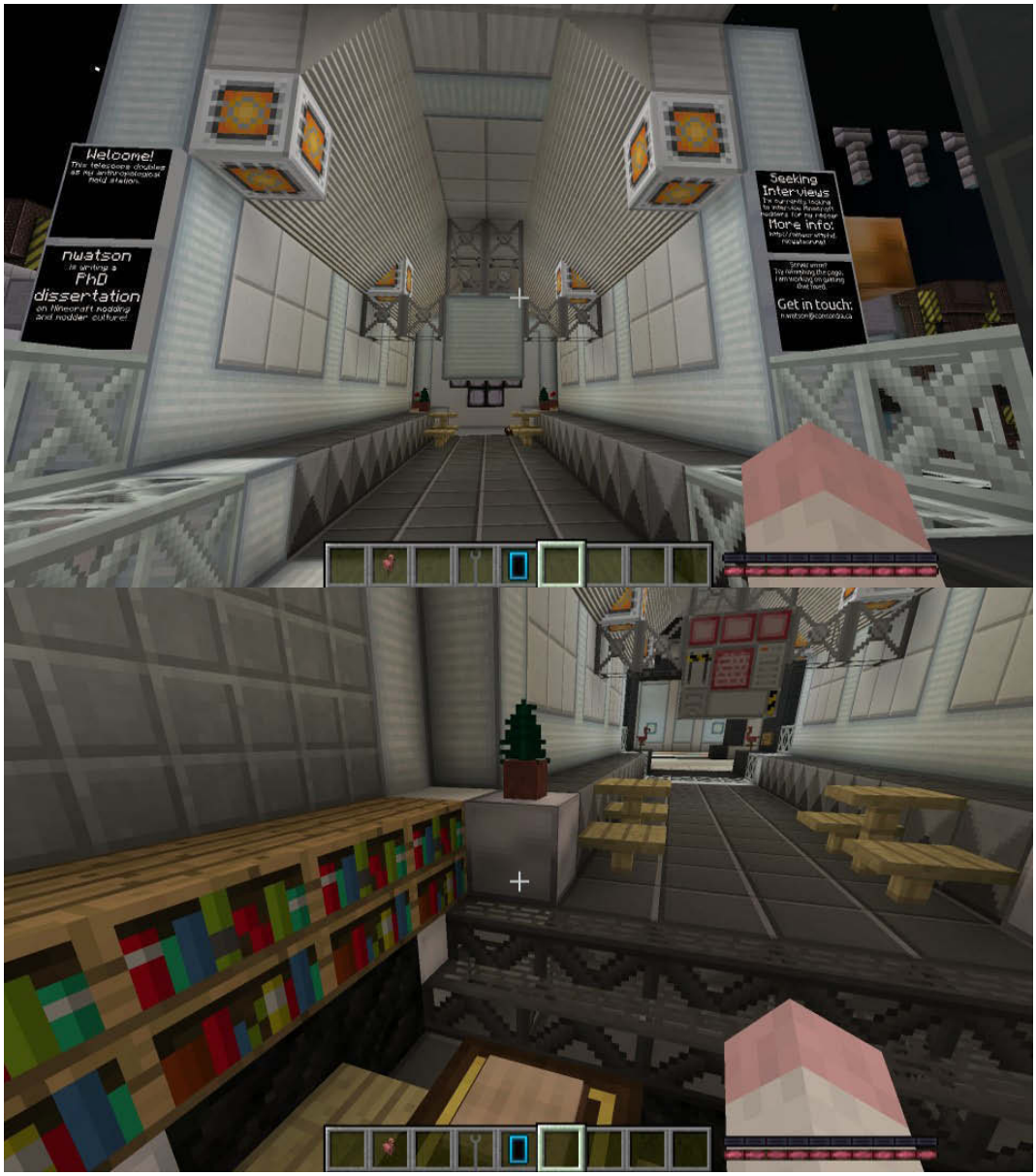


Figure 5-14. Views of the space telescope field station. Top: The telescope entrance, with recruitment signs. **Bottom:** The office/lounge area inside the telscope. (Screenshots by N. Watson.)

5.3 "Minecraft's got problems": The keynote presentation

The server opened to the public at 8 PM UTC on November 17, 2017, with the keynote scheduled for 9 PM. During the first hour, participants gradually filtered in and wandered around looking at the booths. Around 8:50, one of the organizers started announcing in global text chat

(using the mod-provided "shout" command to bypass positional audio) that the opening ceremony would commence soon, and encouraged attendees to make their way to the panel room. By 9:00, over thirty people had gathered there. Most were seated (actually standing, but positioned on a seat) but several were floating near the ceiling or standing around the edges of the room, and more were trickling in all the time. Several of them visited the hat kiosk and put on space helmets. (At all times during the opening ceremony, roughly half of the attendees seemed to be wearing helmets.)
The keynote did not start on time, which is apparently typical: an audience member remarked in the voice chat that "we are five minutes late, which means we are right on time." In actuality, the keynote would start about an hour late, as time was lost due to a server crash and waiting for the "special guest" to arrive—Zoll had technical complications regarding internet connectivity and had to make special arrangements.

While we waited, casual conversation—split between Mumble audio and in-game text chat—turned to a discussion of computing and technical matters. This was the most common topic of conversation I heard over the course of the weekend. As often happens in busy online spaces, there were actually multiple conversations crossing over each other, and positional audio meant that in some cases I could only hear one side of a discussion, because the other speaker was just out of range. This made it difficult to follow exactly what was being discussed, and nearly impossible to keep track of exactly who said what. However, the general impression was that the modders—many of them also employed or studying in IT fields—were spending a lot of their time at BTM "talking shop," debating and sharing knowledge (recall the discussion of the shortcomings of ComputerCraft, mentioned in the previous section).

As I arrived in the panel room, a mod's RAM usage and slow launch times was under discussion, and one modder was criticizing another's approach to the problem: "More cores and more threads are not helping you guys." Another modder was arguing that a big contributor to slow loading times (such as the 5-10 minutes it takes to start up the BTM modpack) was mods that added a lot of new image textures, which needed to be "stitched" together into a master image table on the

fly. Would some sort of texture caching help? "No, that's not going to make it faster. It's not." What about pre-stitching the textures and saving it to disk so that stitching doesn't have to occur every time? "Yeah, that is doable. It's actually been done before."

As time wore on with no sign of the keynote, some attendees became restless and sought ways to pass the time. They began to conjure lightning bolts, shoot arrows, and throw tomatoes at each other and at the staff on-stage. People had already remarked that their framerates were poor because of the number of players present, so a staffer pointed out that "if you're chucking stuff around, that's not helping." Other exclamations of annoyance followed: "Who the hell is throwing tomatoes?" and "Can we *not* hit me with lighting?!" A participant observed, "Um, this is a mess!" to which another replied, "No, this is BTM." At one point, administrators threatened to clear everyone's inventory in order to prevent further tomato-chucking antics, but attendees protested that they needed the items in their inventories, and the threat itself was apparently enough to get the agents of chaos to settle down.

The server crashed again, of course. Some mod was causing fatal errors, and multiple people chimed in on how to make it stop:

> "As expected."
> "So, the funny thing is, I heard someone press ENTER and then the server went down."
> "Oh my god… the null pointer exceptions!"
> "Whoever's using the block that crashes the server, stop using it."
> "What block?"
> "The server-crashing block."
> "Can we destroy it?"
> "Can I turn the server-crashing block into nine server-crashing ingots?"[44]
> "Tothor, can you send me the log from that crash?"
> "I was just running up one of those gravity things."
> "[That's a] coincidence."

One of the staffers put up an image of the well-known "This is fine" meme, featuring an anthropomorphic dog sitting in a burning room and insisting that nothing is amiss. Someone wondered aloud: "Has there ever been a BTM that hasn't begun with that image?"

[44] A Minecraft joke referencing the fact that metal ingots such as iron and gold can be converted into blocks at a 9:1 ratio, and vice-versa.

5.3.1 The keynote address finally begins

Finally, Zoll arrived, introduced himself to audible applause, and began his talk, with accompanying slides displayed on the screen behind him. This address would set the tone for much of the convention, and provide insight into modder attitudes towards Minecamp and Mojang. Several sections of Zoll's speech are elided, but key moments are highlighted here.

Zoll reminisced about how, at the last major BTM, the community of modders and mod-players was split across multiple Minecraft versions. "I am pleased to announce," he said, "that we have finally all moved to Minecraft 1.12.2." The ensuing applause was cut short when he continued, "Or is it 1.11? Or 1.10?" Apparently the community has not united around a common version after all.

"Minecraft's got problems," Zoll continued. "Minecraft is in a bit of a messy state. Some people call that 'meme versions.' We seem to be stuck between three versions. Many people are still playing on 1.7.10." "Woohoo!" interjected an audience member.

For context, version 1.7.10 came out in June 2014 and is long obsolete, but has remained popular among modders and mod fans. According to Tothor, this is because many modders took a long time—months to years—to update their mods to 1.8 and beyond, because changes in the Minecraft and Forge codebases made this a more labour-intensive process than usual. Some popular mods were even abandoned after 1.7.10.

Zoll went on: "The state of the art does not look as good as I would have hoped at this point in time, but we have bigger problems coming up—much bigger problems." With this, the slideshow switched to a photograph of the Earth. One audience member interjected, "Undead!" while another, seeing the image, asked, "Global warming?"

"That is also a problem," Zoll agreed, "But I was referring to an event which we are going to witness in about 20 hours." The Earth picture was an allusion to Minecamp Earth. "But that's… not what I want to talk about now. I wanted to give you some hope, instead of just being dark and

gloomy. So let's look at a short illustrated history of how we got here, and maybe that can give us inspiration to keep going."

Zoll then spoke of past BTMs, showing a screenshot of the first stones being laid in the first panel room for the first convention, which came together from scratch in two days. The countdown-to-BTM timer originated in this context, as the screenshots revealed. Some remarked on how impressive it was that they managed to get the convention going so quickly, so copygirl replied that having more time does not mean it actually gets used: "The funny thing [about this BTM] is everything happened in the last two days. Nobody really bothered, seven weeks ago."

"We had a great convention [in 2015]," Zoll told us, "Even due to all its shortcomings... the server only crashed twice, a record which we have to this day not beaten. All of these weeks of preparation… and the BTM that crashed least was the one we did in 18 hours!" Audience-member CR2032 noted in text chat that this makes some sense—the more prep time you have, the more mods you add, so the more things go wrong.

Zoll then showed some slides from BTM 2016, recalling a few bugs that were so unexpectedly catastrophic, they have become part of BTM lore. Audience members commented on the screenshots: "That's my booth!" "That was me!"

"The message I want to share," Zoll announced, "is that, if my hunches are right, then for the Minecraft community some tougher times might be coming—you know, with the universal plugin API,[45] we don't know what it'll be. It could very well be a replacement for Bukkit, or it could try to rain on our parade . . . But I would like to welcome you all to BTM Moon. For just today, let's put all of the gloom aside and let's just have fun together. Let's just seal them off, let's go to the panels—we still have panels, right? . . . The server has already technically crashed twice, so we can't have the least crashes."

[45] Mojang has long promised that they would eventually release an official modding toolkit. It was widely expected that some announcement would be made on the subject at Minecamp 2017, but this did not happen.

Here, copygirl interjected to much mirth, "We can't have the least amount of crashes, but we can still try to go for the most amount of crashes."

Wrapping it up, Zoll wished "all the best for the community, because even if we're stuck complaining about versions, and complaining that the community's stagnating as *some others* like to do a lot… in the end we stagnated for six years now. I think we can get a few more going. I mean, let's face it: most of the popular mods still use the same power system invented in 2011 with IndustrialCraft and BuildCraft. Ultimately, we're still here! And we're still making new mods. We're still building new content and as a community I think that's what matters."

A flurry of voice chat rose almost immediately after the speech's conclusion. Amadornes took the stage and announced upcoming events, including the official Minecamp Earth stream the following day, which would be streamed in the BTM panel room itself. A portion of post-keynote chatter is worth recounting, because it further elaborates on the modders' attitudes towards Mojang. One attendee remarked that they now understood why the event was called BTM Moon—as a contrast to Minecamp Earth. "Either that, or BTM is a satellite orbiting it, doing occasional orbital strikes!" Another person replied, "This year, Bug Testing Marathon actually *is* better than Minecamp," echoing the general opinion that Minecamp's emphasis on the global streaming gimmick fell flat, missed the point of having conventions, and would detract from the overall quality of the event by turning it into little more than an extended video advertisement. Zoll joked that he thought BTM's moon was a *Sailor Moon* reference, while copygirl remarked, "I thought it was about how we'd be 'mooning' the Mojang guys by doing something cooler than they were." As conversation turned to whether the potential release of an official modding API would affect the legality of Forge modding, one modder in attendance asserted, "I think it's pretty clear that Mojang doesn't deserve to have their terms of use taken seriously at this point."

More broadly, it is Mojang itself that is not being taken entirely seriously here. In denigrating Mojang's policies and its public relations machine, a subset of BTM attendees are engaging in a kind

of "carnivalization" which, naturally, falls on the tactical side of the de Certeau-binary that I have been carrying through this study. The concept of carnivalization, which originated with literary theorist Mikhail Bakhtin, is used by Holt as an interpretive frame for understanding certain modes of discourse on the Internet, in which participants simultaneously lower the status of those in positions of power and authority while elevating their own statuses (2004, pp. 104-105). Holt explains:

> Whatever its form, carnivalization is a reversal of the positions between those in authority and those they control. Moreover, one key feature of carnivalization is humor, occurring as parody, exaggeration, and so on. In other words, status is reversed by subordinates making fun of superiors. (2004, p. 105)

The carnivalized nature of the BTM proceedings were perhaps most apparent during the in-game Day 2 Minecamp livestreaming event.

5.4 Minecamp Earth: The meme/cringe generator

Some modders didn't bother to watch the Minecamp Earth stream, either in BTM or on their web browsers. Qwil told me later that he avoided it because of its "cringey" nature, while Icoso, who did watch, echoed in an interview that there was "plenty of cringe." This neologistic noun-form of "cringe" generally means "content that makes one cringe," and in this context it can refer to a wide range of corporate content that viewers feel does not effectively interpellate them as audience-members: marketing personalities that seem rehearsed and disingenuous in their bubbly excitement; song and dance interludes intended to entertain much younger viewers; the perceived inauthenticity of celebrity personalities claiming to be avid players and making rehearsed jokes (Minecamp host Will Arnett's wisecracks were not especially popular with the BTM crowd); and anything that is seen as a poor excuse for an unpopular design decision. Not everyone had a negative view: one interviewee remarked that the stream wasn't bad, though "mostly for kids."

The most troublesome aspect of the stream for those actually viewing it in the panel room was purely technical. The server could not keep up with the demands of all the blocks and players, *and* process and display a video stream in real time. Despite efforts by the staff to optimize TPS

(TICKS PER SECOND, a measure of how fast the server is able to think) by removing gravity shafts and despawning mobs, the in-world stream ran several minutes behind, was severely pixelated, and kept skipping and repeating segments like a broken record. Not to be deterred, viewers made light of the situation. When an enthusiastic on-stream announcer talked up Minecamp Earth's global/online nature by asking if viewers were "confused by the technicalities of international travel," the BTM audience laughed much over the glitchy repetition: "Confused by the technicalities of international travel? – Travel? – ational travel? Technicalities of international travel?" An audience member claimed that BTM had inadvertently created a "meme generator," and the "international travel" meme was repeated many times over the course of the following 24 hours, as participants exaggerated and parodized the audio distortion in service of carnivalization.

The more vocal audience members were not above making sarcastically critical commentary, largely pertaining to their disapproval of Mojang and Microsoft's stewardship of the game. When a Minecamp panelist talked up the "Better Together" update that enabled cross-platform multiplayer, an attendee remarked in text-chat:

> "we can play crossplatform now" – "but fuck linux and mac"

And another added:

> Crossplatform that is not at all crossplatform

One of the big (and much-discussed at BTM) Minecamp headline-grabbers was to be a "Vote a new Mob into Minecraft!" event. Mojang had published design proposals for four new monsters, and during the livestream, fans would be able to vote online for their favourite. The winning mob would be implemented in a future release. In snippets of conversation I heard on Day 1 and 2, I got the impression that multiple BTMers favoured a sinister monster that would appear and attack players only if they had gone a long time without sleeping in-game. During the stream, trash cans labeled A, B, C, and D were set up at the front of the room, and BTMers could vote by depositing items into them.

The general interest in the mob designs and the outcome of the vote did not prevent people from being sharply critical of Mojang's handling of the affair, either. Why couldn't they just include all four mobs? From the modders' point of view, it didn't seem that hard: solo modders had been adding colourful and complex new creatures to the game for years, so the Mojang spokesperson's claim that they needed to focus on just one to ensure quality and creativity fell flat with this audience. In exemplary carnival fashion, Icoso quipped, "It might be a pale mockery of mods that have existed for almost a decade but they're trying!" Several times, people remarked in chat that modders would end up implementing all four ideas within a couple of months anyway—including a version of the winning mob that would be better than the official offering.

Although the long-awaited modding API was not unveiled at Minecamp, the speakers reinforced the expectation that the eventual offering would facilitate plugins written in the C# programming language, targeting the newer Bedrock editions for Windows 10 and console, rather than the original Java edition that BTM was using. This was both disappointing and a relief: disappointing because Mojang had been promising the modding toolkit since well before the Microsoft purchase and Bedrock editions, but modders felt that they had instead done all the heavy lifting themselves by building systems like Forge and Bukkit; a relief because the proposed add-ons framework would not, as Zoll put it in his keynote address, "rain on our parade." There were even cautiously optimistic attitudes towards the proposed system. In interviews, both Mezz and Tothor told me that they were intrigued and might be interested in trying it out, though it likely would not offer the same expansive freedom that Forge modding enables.

5.5 Curiosity and chaos: crashing and exploding all the things

The *constructive chaos* of BTM organization has been described, but another kind of free-wheeling and whimsically destructive experimental play made itself apparent in the latter half of the event.

It began, from my perspective, with cam72cam's Q&A panel on Immersive Railroading (IR). Several mods, including RailCraft, TrainCraft, and Steve's Carts, have expanded Minecraft's minecart system to support complex and detailed trains and railroading equipment, but IR is a separate system with its own modular track segments (about 2.5 times wider than minecart tracks) and a whole fleet of locomotives and cars patterned after real-world rolling stock. In the gigantic roundhouse that was IR's demo "booth," players learned to tweak track placement and use gigantic wrenches to assemble and disassemble entire locomotives from their key components. There was nothing mischievous about the questions during the panel: How do you run these tracks up hills? Is it hard to make these train models? Why are these locomotive assembly wrenches comically huge? (The answer to this last question is that seven-foot-long wrenches are actually used in real-world locomotive work.)

After the Q&A, cam invited people to go with him to the train station, where he would demonstrate how to drive the trains. We would have to wait for the active train to come around so we could hop in. As several people spilled out onto the tracks, cam warned them that getting hit by the train would be quite lethal. This, naturally, prompted some folks to want to play chicken, and not everybody got off the tracks in time. This is the probable origin of the "death by train" count on the spawn plaza sign.

Once on board the train, the questions took on a different character. How fast can these trains go? Can we take that train in the siding there and put it on the track too? If we make it go really fast, can we catch up to the first train and crash into it? cam readily accommodated these requests, but the train collision failed to produce a spectacular wreck: instead, the trains were kind of superimposed on each other for a time.

Then the server crashed again.

5.5.1 Ending BTM

The closing ceremony was a blast.

Actually it was a series of blasts, starting with small detonations but culminating in the massive, world-annihilating explosion of an aptly-named "chaos crystal." None of this was officially planned. Rather, acts of chaos and celebratory destruction gradually amped up in the final hours of the convention.

The beginning of the end came after the final panel presentation by Cervator, who talked about his Minecraft-inspired game project *Terasology*, which was explicitly (strategically) designed to be extensible and support modding from the start. One of the staffers wondered aloud what to do to keep people entertained for the next hour or so until the closing ceremony. Copygirl invited everyone to go to her booth to acquire BTM Moon exclusive floating decals for their avatars, if they had not done so already. As a dozen of us crowded into the booth, one individual detoured to DJ Flamin' GO's station nearby to queue up some music. After playing around for several minutes with the elevators and sliding doors in copygirl's ship, people filed out onto the dance floor. It did not take long for a population of autonomous curiosities to appear: a wolf, wandering penguins wearing backpacks, knee-high miniature horses wearing the Minecraft cake texture, a swarm of tiny clay soldiers hopping all over the place. Multiple people started trying to control the DJ's music computer at once. Someone remarked, "This has the BTM feeling of utter chaos!"

Sensing that I might not have much more time, I took the opportunity to run around the now-empty parts of the station, taking note of booths I'd missed. Then I flew out to get exterior views so that I could analyze the layout later. By the time I returned to the fun zone, music was playing and people were hopping and gliding all over the place in what I assume was the best facsimile of dancing possible, given Minecraft's limited expressive affordances. Fires had been set in random locations: I had nothing to worry about as I was in Creative Mode, but at least one less-fortunate attendee burned up.

Then the lightning strikes started, a rapid-fire of explosive bolts all around the room. Although these did not actually damage the structures, the sounds apparently signaled that destruction had begun. One attendee who was hoping to do more booth exploration after the closing

ceremony (or perhaps just wanted to ensure that the convention centre was preserved for posterity), wanted people to stop: "Please don't destroy stuff!"

Zoll, however, responded by saying, "Please actually destroy stuff!"

The closing ceremony was to take place in a different Minecraft dimension, in a special room designed for the purpose. Organizers mused for several minutes on the best way to get the 30+ remaining players to that location. There was some discussion of a "rocket" but they decided it would be more practical to use a giant portal instead. Copygirl asked people to head to the panel room, and headed there herself with a crowd of about half a dozen in tow—getting from the Fun Zone to the panel room was not trivial, so it was helpful to be able to follow the person most familiar with the station layout.

We gathered on stage and waited as stragglers filtered in. As a prank, someone started throwing potions that made everyone float up to the ceiling. Copygirl asked us all to line up, face forward, and use the mod-provided "wave" emote at the same time so that she could take a group photograph. I was surprised at how well the group managed to synchronize the use of the emote. Then, a strange noise welled up all around, and suddenly we were somewhere else.

We had finally made it to the moon.

We were in a hemispherical glass dome with a flat, white floor and a raised stage protruding from one wall towards the centre. Outside was a tumbled, pale lunar landscape, and in case there was any doubt, signs just outside the dome in the four cardinal directions informed us, "This is the moon." That didn't stop several guests from joking that we'd been "lied to" when the normal Minecraft moon passed overhead—this dimension was using the same sky configuration as the convention centre, making the moon visible from the moon. "That's actually just a really bright star," someone claimed.

Technical discussion of mods and other favourite games, combined with plans and jokes pertaining to the next BTM ("next time there will be microtransactions"), formed the backdrop of ongoing voice chat for the next half hour. These seemed like fall-back topics to which conversation

reverted whenever it was not actively being pulled in another direction—as it was periodically, to attend to the business of wrapping up the convention.

First, copygirl took the stage to thank everybody for attending, and also to thank those who helped with organization. She recounted how she had taken over responsibility for the convention from Zoll in September, and had not been too concerned about the task, but about three weeks later, "panic set in," and in practice most of the heavy lifting happened in the last few days. Zoll reiterated that this is how it has always been with BTM preparations ("Thirty-two hours? That's a lot of time! What are you talking about?").

Zoll then delivered his closing remarks:

> As you know I've organized a few BTMs, and I have to say I am pleased with how this one turned out, even though the main attraction was not what it was supposed to be. Thankfully nothing of value was lost. Except the mobs of course! I'm really sad the mobs were lost.

The "main attraction" in this case was the ill-fated attempt to display the Minecamp stream in-world. As discussed previously, all mobs were deleted as part of an ultimately fruitless effort to improve TPS. In saying "nothing of value was lost," Zoll is implying an overall low opinion of Minecamp.

After thanking the organizing staff, Zoll asked the audience to think about which mod we thought was the buggiest. Then, on his signal, we would all shout out our answer. Most people said, "Screens!" referring to the OpenComputers monitors, probably because of the streaming difficulties. The jokes and teasing quickly followed, with one person saying "Stagelock!"—Zoll's optimization and bugfixing mod, prompting Zoll and others to burst into laughter. Apparently the celebrities of the modding world were not to be spared the barbs of carnivalization either, although they were certainly gentler and were received accordingly, in playfully self-deprecating spirit.

Zoll also noted that another convention that "shall not be named" (i.e. Minecamp) had occurred that same weekend, and asked us which one was better. The overwhelming response was

"BTM", with several chatters claiming (falsely, no doubt) that they were unaware that any other event had taken place. At least, Zoll noted to much collective mirth, we were now all better informed about the "technicalities of international travel"—the most prominent meme from Day 2 thus making a reappearance at the closing ceremony.

Zoll, Tothor, copygirl, and Amadornes began to discuss plans for a "last man standing" contest of sorts. Zoll asked if Tothor could write a mod to make it so that anyone who disconnected would not be able to reconnect. Then, there would be some kind of "participation award" for whoever managed to stay on the glitchy and unstable server the longest. Copygirl suggested simply enabling whitelisting, but in that case server ops would still be able to connect. This led to a steady stream of attendees shouting in text chat, "CAN I GET OP"—a meme in which they were pretending to be annoying young Minecraft players who (supposedly) show up on servers and ask for administrator powers.

Figure 5-15. End-of-convention lunar dance party. (Screenshot by N. Watson.)

Meanwhile, the partying started (Figure 5-15, above). Turrets began to appear and shoot projectiles at each other, creating explosions that were colourful but harmless to the terrain. Someone turned on loud disco music and activated a laser lightshow. Tomatoes began to fly. Someone asked, "Can we please nuke something?" Some device from the Draconic Evolution mod

generated deep warbling sounds and made the star field whirl past at breakneck speed. Of course the server crashed, again.

This crash did not kick everyone off immediately like most do. Instead, everyone froze and the world stopped responding to block interactions, but it was still possible to move around—only, one would be moving around in a client-side representation of the world, because it had stopped listening to network traffic. The Mumble chat, being run on a separate server, persisted. Zoll asked Tothor if he had remembered to turn off the "snooper," which is Mojang's anonymous analytics tool embedded in Minecraft. Tothor said no, so Zoll joked that this explained what was happening: "Mojang has finally shut us down!"

Zoll said that it looked like the server had decided to end BTM for us, but Tothor managed to get it restarted and we all rejoined. Lunar partying resumed. Tothor placed a COMMAND BLOCK with a pressure plate to teleport people back to the convention centre—I observed people jumping on the plate before he had even finished programming the block. Because of the way command blocks work, they do not actually know who pressed the button. Instead, this block just targeted the closest player, and that is how I ended up back at the spawn plaza unintentionally.

The remaining people were now split into two groups—one at spawn and one still on the moon. At spawn, tomato chucking continued, and people started rapidly building all manner of curiosities, such as a row of lamps that lit up like a strobe. The conversation, which at times was very much unrelated to the in-world goings-on, discussed which staffers had volunteered at every BTM to date. Zoll was trying to remember the first BTM at which copygirl had played an organizer role, but then decided, "You get into the BTM organizers club just because you made Flamingos." Copygirl commented that *Flamingo* is the one mod she would least want to have as the reason for such an honour. Zoll's response is indicative of how modders perceive mod types to exist in a kind of hierarchy:

> After I made Stagelock, I was always most sad that I was best known for making an optimization mod and not a content mod. Then I made Pinventory and that was my first mod to end up on 9-minecraft. Remember that wishes can backfire!

9-minecraft.net is a site that re-posts mod downloads without the authors' permission, and generates advertising revenue in doing so. Having one's mod end up there is a dubious—or at best ambivalent—honour: certainly for a mod to get stolen in this way, it must be popular enough for the profiteering poachers to take notice.

Judging from the chat, interesting things were happening back on the moon. People were shouting, "NUKE NUKE" and "BOOM BOOM BOOM!" Someone said, "Did you say something about blowing up the moon? We can blow up the moon…." Then a voice said, "What's that? Fuckfuckfuckfuck, it's a nuke, it's a nuke, it's a nuke, it's a nuke." The chat reported several deaths by radiation, meaning that the folks from the moon reappeared at spawn.

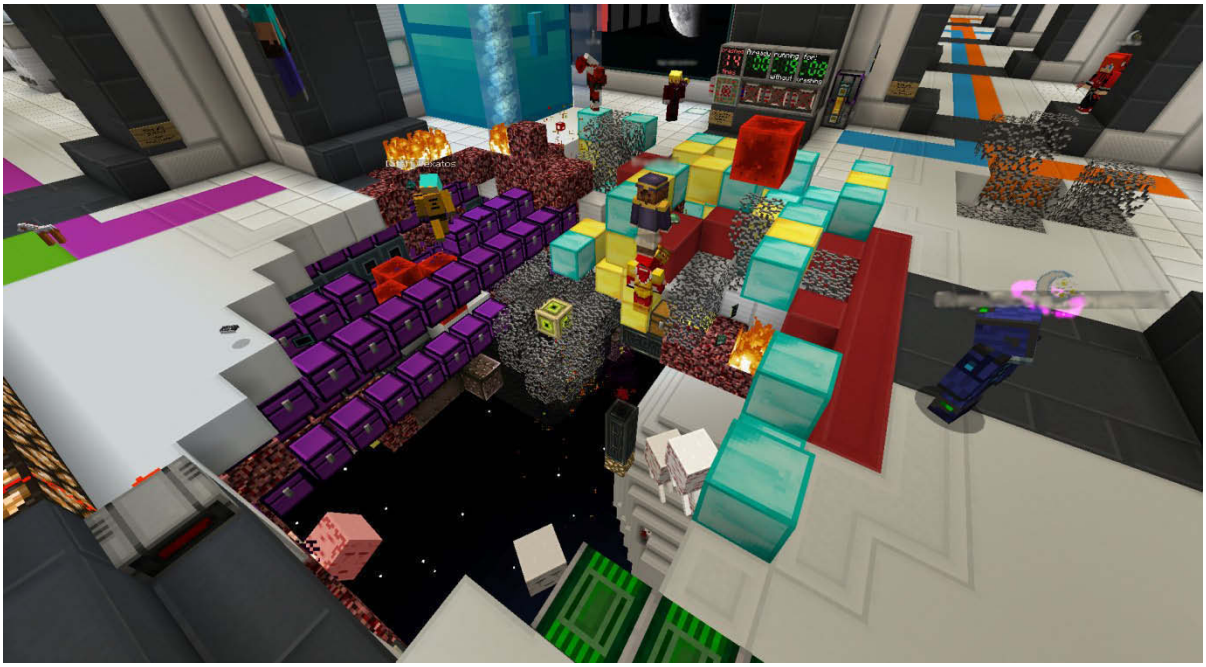


Figure 5-16. Carminite reactor detonations take their toll on the spawn plaza. (Screenshot by N. Watson.)

Copygirl commenced her own destructive fun at this stage, placing "Carminite reactors" from TwilightForest. When detonated, these blocks create small holes and generate flying, fireball-shooting monsters called ghasts (Figure 5-16, above). As the spawn plaza turned into Swiss cheese,

someone remarked, "I guess spawn is going." Another exclaimed, "Who the fuck did this?" Copygirl asserted that "kind of the point of the end of BTM is to blow up spawn." Someone else protested: "No, don't, please," but copygirl reassured them that the entire world had already been safely backed up: "We have a mod called Back The Fuck Up which backs up everything."

"But BTM has been able to run without backups. It was not able to run without flamingos," someone said.

Zoll: "Yeah, we actually ported the *Flamingo* mod back to 1.4.7 just so that we could have flamingos at RetroBTM!"

The celebration of "BTM madness" peaked. People started playing with adding gravitational fields, causing everyone to go upside-down. Some mod-added nukes, which look like vanilla Minecraft TNT with a radiation symbol on the side, appeared at spawn, but they were not successfully detonated: many of the bomb effects were disabled within the convention centre (although not, apparently, on the moon) to prevent griefing.

Then came the chaos crystal.

The entire spawn plaza instantly dissolved beneath my feet, and I fell through the void to my death. When I respawned, I beheld a station that was missing most of its middle. Where the spawn plaza once was, there was only a gaping circular emptiness (though there was some debate in chat over whether it was a circle or a rhombus). The nothingness was expanding outwards, eating away at the convention center, and as I watched my telescope disintegrate, someone remarked, "It's a bit late to stop blowing stuff up" (Figure 5-17, below).

Exclamations of amazement followed:

"That's the thing exploding. Oh my goodness."

"How big is that explosion?"

Again there was a note of protest against the destruction, and again copygirl asserted, "This is like BTM tradition that we completely obliterate the server." If, as Schneier and Taylor (2018) suggest, Minecraft play tends towards either the monumentary or the momentary, then BTM is a

contradiction: a grand, intricate monument to modder culture that lasts for three days and gets destroyed in spectacular fashion.

Noting that an annihilated station would reduce lag significantly, someone joked that "at least we'll have 20 TPS[46] when we're done."

At first, several people said that the cause was the detonation of a "Draconic Reactor," but the perpetrator corrected them:

"Actually it wasn't the Draconic Reactor. That wasn't enabled. It was the chaos crystal… which isn't disabled."

Figure 5-17. Effects of the chaos crystal detonation. Top: My field station disintegrates in the chaos vortex. **Bottom:** The majority of the station has been consumed. (Screenshots by N. Watson.)

[46] 20 ticks per second is considered the standard speed at which a Minecraft server ought to run under good conditions.

"Oh, thanks for letting us know *now*," replied copygirl, who had gone to great lengths to disable features that could cause "hell to break loose" (recall the broom dispenser—Section 5.1). Somehow the enormous liability of a station-obliterating bomb had gone unnoticed by all of the organizers. It could have been very bad, copygirl noted, if someone had discovered earlier on that chaos crystals were functional.

As crystal-related deaths and client crashes prompted an increasing number of attendees to call it a night, conversation turned back to modding matters. Zoll said, "I'm still disappointed that there was no modding API news at Minecamp." Copygirl sang a few notes of "Where's the modding API," a music video created by the YouTube channel Yogscast as a humourous lament about the instability of Minecraft modpacks and the lack of official mod support. To drive the point home, someone said, "Can I quickly mention that the *Where's the modding API* song is from 2013?"

It seemed as good a note as any to end on: me hanging in space above a chaos vortex, while the modding community hung in limbo over the question of official modding support—yet surviving, even thriving. I flew to the zenith, stood back, and watched the world end.

5.6 The lessons of BTM Moon

BTM is not representative of all Forge modders—while reviewing this document, some BTM organizers stressed that BTM attendees were something of a niche subset of Minecraft modders in general. Nevertheless, it is a vibrant and dynamic subset, and includes many creators of well-known mods. BTM proves that Minecraft is settled by modders, in the sense that they have moved in and made it their own. It also proves that it is still unsettled, as in turbulent and chaotic—"in a bit of a messy state" as Zoll puts it. The march towards rationalized, strategic modding is enveloped in a rich mosaic of enduring tactical practices. In the following chapter we will be taking a closer look at that rationalization process itself, but first I offer my four key takeaways from this weekend of productive play.

1. **Chaos is celebrated**

From the frantic last-minute building activity, to the taking-in-stride of catastrophic failure, to the gleeful post-convention destruction, it is clear that BTM thrives on chaos as a generator for creativity. While staff employs organized, strategic approaches to specific tasks, like the Stagelock optimizations or the careful design of capitalthree's Back The Fuck Up (BTFU), it is a mess of tactical (and sometimes panicky) improvisation that ties it all together. What's more, nobody seems to think that it should be any other way.

2. **Modders have a complicated relationship with Mojang**

The Minecamp stream was the subject of much mockery and derision, while Mojang's design decisions regarding cross-platform play and the new mob vote were sharply criticized. But even some of the most critical modders did not completely demonize Mojang. Zoll noted *Stardew Valley* shared two things in common with Minecraft: "A nice developer and no proper modding API." It is something of a testament to the company's "niceness" that they have generally taken a permissive and celebratory attitude towards modding. Modder freedom and the lack of official support may, however, go hand in hand—a topic to which I return in Chapter 8.

3. **Logistical mods are the unsung heroes**

Some of the most important mods at BTM did not even have booths of their own. Stagelock and BTFU did their work behind-the-scenes, and most guests probably were not even aware of their existence. Trash cans—easy to take for granted—ensured that the server did not buckle under the load of item-tracking lag. Vexatos's Conventional, relegated to a few explanatory signs in the "No Fun Zone," is what made it possible to keep the convention on the *productive* side of pure chaos. And beyond having an entire wing of the station to showing off its add-ons, OpenComputers was woven into every part of the infrastructure, from the DJ booth to the panel room screen and the crash counter. Even the informational signs in my own booth relied on these features. Logistical mods

may lack glamour and prestige, as Zoll suggests in his contrast between Stagelock and "content mods," but they are central to social life and creative production in the modding community.

4. Modding and play have a two-way relationship

Scholars have framed modding as an extension of play (see e.g. Sihvonen, 2011, p. 88). However, BTM makes it clear that playing is also an extension of modding. BTM attendees were neither playing a mod-enriched canonical Minecraft game, nor actively writing mod code, but what they were doing was clearly both a form of play and a mod testing/workshopping activity. It is thus not possible to maintain the notion that there is a core "play" activity associated with the game, with an attached "play mode" that consists of designing and programming mods. On the surface, BTM activity looks more like vanilla play than it looks like coding, but the coding activity stands between and separates these two contexts of play.

VI. Operational Logics and the Rationalization of Modding

Sadly, abstractions are lies at their core.

—Minecraft Forge Documentation, https://mcforge.readthedocs.io

Chapter 4, Settling Minecraft, discussed how fan activity, including modding, intervenes in the articulation of Minecraft as a cultural object; it also described—briefly and in general terms—how modding has evolved from a loose collection of ad-hoc tactical practices, to a set of rationalizing strategies based on a combination of community consensus and the assertions of influential modders.

Two areas bear further examination: the technical details of how modding has been increasingly rationalized and regulated; and the discursive actions by which prominent voices in the community work to legitimize or de-legitimize modding practices. This chapter deals with the former, looking at how *process* is subjected to increasing strategic rationalization, in turn imposing constraints and prescriptions on what modders do and how they do it. A *standard reading* of the relative merits of these prescriptions will be applied. The next chapter will reconsider these reconfigurations of modding practice as *discursive acts* (i.e. an utterance reading), and will describe the social arenas in which they are developed, deployed, and contested.

6.1 Domains of rationalization

As noted in Chapter 3, I am working from Weber's (1930) definition of rationalization as the tendency in modernity for traditional values and emotional motivations to be replaced by rationally planned, calculated, results-focused prescriptions. Rationalization strives to establish efficiency, calculability, predictability, and control (Ritzer, 1983).

In games, rationalization appears in three domains, which I will call the *computational processes*, the *practice processes*, and the *player processes*. Computational processes are the activities carried out by the machine in order to realize an interactive, rule-bound simulation game-world,

while practice processes are carried out by human game developers (and modders) in designing and packaging those computational processes. Finally, player processes are those carried out by the players as they interact with the game.

That a game's underlying computational processes are rationalized is unsurprising: the very notion of a game implies the existence of rules, which need to be consistently defined and applied even in the most open-ended game worlds. Further, the rules of the game must be operationalized through logic states of a machine. I take the constituent units of rationalized process to be what Wardrip-Fruin calls "operational logics"—identifiable, abstract logical elements of a computational system such as windowing, collision detection, or virtual dice-rolling (2009, p. 13, 16).

Operational logics don't stay inside the machine, however. These computational elements are entangled with the practices of institutions that provide a backdrop for software design activities, such that programming logics like data encapsulation[47] are simultaneously constellations of operational units *and* disciplinary "best practices"—that is, normative cultural codes. In the following pages, I explore how sets of closely-linked operational logics are exposed and implicated in all three process domains under discussion (namely, the aforementioned player, computational, and practice processes), and what this means for modders.

6.2 Exposing operational logics across domains

Wardrip-Fruin identifies *quest flags* and *dialogue trees* (2009, p. 46) as examples of operational logics commonly seen in role-playing games. Quest flags are integer or Boolean variables that store vital information about a player's progress through a quest subplot, while dialogue trees are data structures that provide snippets of in-game dialogue and transition rules for determining the order in which they are encountered in conversation. While illustrating these concepts with mock examples implemented in BioWare's Aurora Toolset, Wardrip-Fruin somewhat understates the significance of

[47] Data encapsulation refers to the practice of bundling pieces of related data together in a program module (an "object") along with the procedures that need to work on those data, while keeping them hidden and protected from tampering by other parts of the program.

how computation-level operational logics are implicated and reflected in the authoring tools themselves: the logics of the game designer's ongoing practice processes are shaped by the structure of the computational processes.

Awareness of computational processes bleeds through to the player level too. Game wikis and forum posts suggest that RPG players have an awareness that the game program tracks certain facts about the game-world in order to decide which quest activities should be available. Cheating to skip an unpopular quest or using console commands to rescue the game from a bugged state may even require a direct knowledge of the relevant quest flags.[48]

With this in mind, a multi-layer (player/computation/practice) analysis of operational logics is warranted. Similar models have been used in the past to analyze games, such as Konzack's (2002) seven-layer (hardware, code, functionality, gameplay, meaning, referentiality, and socio-cultural context) or Montfort's (2006) five-layer (platform, game code, game form, interface, and reception/operation) models. In collapsing from seven or five to three levels (one of which does not even correspond to anything discussed by Konzack or Montfort), I am no doubt sacrificing some nuance, but my focus is narrowly on operational logics and my ultimate goal is to arrive at an exploration of modders' coding practices, and I want to avoid getting hung up in the subtle differences between, say, functionality and gameplay. I am not arguing for a neat delineation between game mechanics, computational processes, and source code idioms either—on the contrary, we shall see how the same "family" of operational logics (e.g. Minecraft's block-based metaphysics) takes on a different "flavour" (a different specific operationalization) in each of the three domains. If, as is one of the central claims of this dissertation, the practices of developers (including modders) are an important component in understanding a digital game's "context," then this element's absence from Konzack's or Montfort's definition of context is conspicuous.

[48] For example, numerous posts at https://web.archive.org/web/20170320133055/http://forums.bethsoft.com/topic/1546438-fallout-4-crash-in-specific-area/ on the *Fallout 4* official forums provide user-submitted solutions for quest-breaking bugs. In one quest, the player must kill an NPC, but the game fails to register the action and advance the quest. The thread recommends that players use the arcane console command "player.setstage MS04 900" to force the quest flag to the proper state.

Finally, I use "level" and "domain" interchangeably in my model, but am inclined to avoid "layer," because the domains are not ordered by the extent of their abstraction. The only priority that practice processes have over computational processes over player processes is temporal: source code is first written, then compiled, then run/played.

I will occasionally make reference to common object-oriented programming (OOP) concepts in order to better illustrate why some aspects of computation and developer practice have been operationalized in the way that they have. In particular, the concepts of *class*, *object*, *instantiation*, *inheritance*, *overriding*, and *polymorphism* are used. Readers not familiar with these terms are invited to consult Appendix A for something of a crash-course.

6.3 Operational logic families

My analysis begins with an exploration across all three domains of two closely-linked operational logic "families" of Minecraft: block-based metaphysics and chunk-based geography. I will also touch briefly upon some other closely-related operational logic families, such as item-manipulation, crafting recipes, and biomes. (The information presented herein is valid for Minecraft versions up to 1.12, and is likely to be inconsistent with subsequent versions released after 2017. Some minor technical details are elided or simplified for clarity.)

6.3.1 Logic families in summary

- Chunk-based geography

  This pertains to the macro-scale structure of a Minecraft world. As columns of blocks (see below) 16 by 16 metres wide and 256 metres tall, chunks are basic units of both player-level geographic orientation and computation-level data storage. Their structure has implications for both the top-level organization of the world and block-level activities, across all three process domains.

- Block-based metaphysics

The primary constituents of a Minecraft world are *blocks*. Each block is a one-metre cube, and a three-dimensional grid of these cubes encompasses all that exists (excluding certain dynamic "ENTITIES," such as living creatures, flying projectiles, and items that have been dropped from a player's inventory). Block coordinates in this grid are Cartesian triples: x (east/west), y (up/down), and z (south/north). Much of the gameplay involves reshaping the world by placing and removing blocks, which accounts for the frequent comparisons to Lego (e.g. Duncan, 2011, p. 6).

6.3.2 Chunk-based geography

6.3.2.1 Preamble: How a Minecraft world is put together

A Minecraft world is a wafer of practically infinite diameter, and a thickness (height) of 256 metres. It is, of course, not truly infinite in either the "potential" or "actual" senses of the word. For an infinite world to actually exist would require infinite disk space. Instead, the world is at any given time finite, but Minecraft generates new terrain at the edges as the player moves around in order to ensure that there is always more world wherever they go. However, the seeming promise of this potential infinity breaks down at extreme distances, with versions since Beta 1.8 (September 15, 2011) establishing a hard boundary around 30 million blocks away from the spawn point.[49] This leaves each Minecraft player with a theoretical demesne of 3.6 billion square kilometres, over seven times the surface area of the real Earth. That is more than enough room for most players—only a tiny fraction is usable on modern hardware anyway (if terrain were generated out to the limit in all directions, it would take over 80,000 terabytes of disk space).[50]

[49] Specifically, the world border lies 29,999,984 blocks away from the world origin (i.e. coordinates x=0, z=0), which is *near* but not necessarily precisely collocated with the spawn point. Although there are tricks for passing through this border, no terrain generation occurs beyond 30,000,496 blocks. Prior to Beta 1.8, no hard limit existed, but due to math-related glitches, reality would break down rather spectacularly at extreme distances, with different effects corresponding to different distance milestones within these "Far Lands."

[50] This calculation is based on pre-generating a square-shaped Minecraft world 200 chunks (3200 blocks) on a side (a total of 40,000 chunks), measuring its disk footprint (about 232 megabytes), and then extrapolating to the maximum possible size of a Minecraft world (14,062,485,000,004 chunks).

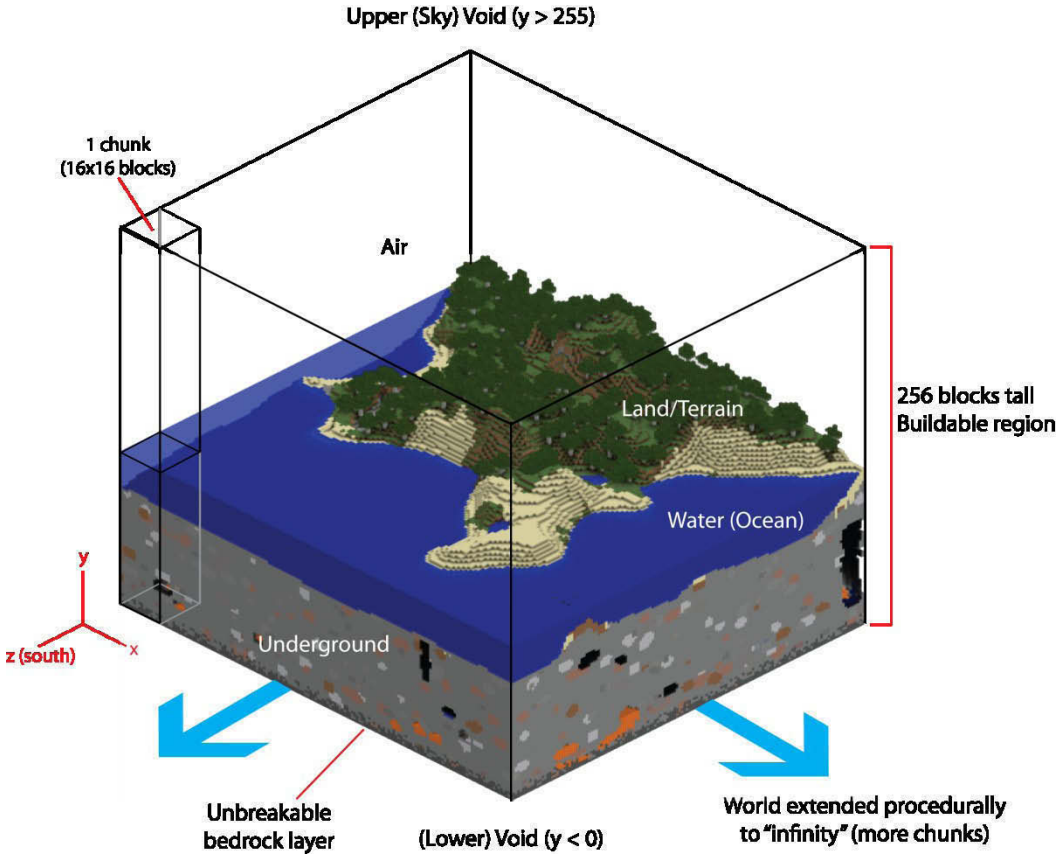


Figure 6-1. This **diagram of a portion of a Minecraft world** shows the naturally-generated terrain occupying the bottom quarter of the 256-block-tall buildable region, with the boundaries of a single chunk marked off. All of the space within this volume is segmented into a three-dimensional grid, with each grid-cube corresponding to one block (empty cubes are said to be air "blocks"). Blocks cannot exist above or below this region. (Diagram by N. Watson. Isometric projection rendered from a Minecraft save file using Mapcrafter software: https://mapcrafter.org/)

The more significant boundaries for the player are those that exist above and below. At zero on the y-axis of the world coordinate system lies a layer of "bedrock," which cannot be broken in ordinary Survival gameplay (it *can* be broken in Creative Mode, or using commands). Blocks can only exist between y=0 and y=255, beyond which are featureless VOIDS. It is possible for the player (and other creatures) to travel through these voids if they have flight abilities, but anyone or anything that falls too deep into the lower void will be annihilated. Figure 6-1 (above) shows the high-level structure of a Minecraft world, distilling the above information into a single diagram.

6.3.2.2 Chunk geography on the player level

At first glance, the *biome* would seem to be the most salient unit of Minecraft geography on the player level. Biomes, which are blob-shaped and hundreds of blocks wide, feature their own

terrain, flora, and fauna based on biome type. Experienced players are aware that some resources—plant, animal, and mineral—can only be found in certain biomes, and that some biomes offer special boons or hazards. While it is true that most of a player's geographically-sensitive activity is organized around biomes, a different unit called a *chunk* intrudes in player processes in a way that Mojang likely never intended. To understand why, we need to look at the role that chunks play in organizing and operationalizing space.

In my prior research on fan creation in *Myst Online* and *Second Life*, I argued that large virtual spaces were generally organized according to one of two spatial logics: a *grid* or a *lattice*, each of which divides up the world into smaller, more manageable units (Watson, 2012, p. 110). The reason this division is necessary at all is because it is generally impractical to have the entirety of a vast virtual space loaded into memory all at once, much less to run active processes in each one.

Lattices solve the problem by using self-contained, isolated regions with a limited number of transitional connections between them. When the player passes through such a connection, a new area is loaded and the old one is simultaneously unloaded. These transitions typically cover the player's "hard" displacement to a new area by means of some diversion, diegetic or otherwise—a massive doorway, an elevator ride, a teleporter, a linking book, a stats-and-score report, or even just an unapologetic loading screen. Lattices are hyperspatial, with no global geometry, since the different regions do not actually have to fit together geometrically. Many games use a simplified lattice—a chain, really—in which the player moves through self-contained levels in sequence. Of the two topologies, lattices are the simplest to implement (computation domain), and the hardest to map out visually (player domain).

Grids, on the other hand, present the illusion of a vast contiguous space, one that is actually too big to be loaded all at once. Instead, at any given time, some number of "cells" in the vicinity of the player are loaded, out to the limit of visibility. As the player moves through the world, the cells they leave behind are unloaded while those ahead are loaded. Displacement is continuous and

seamless, with no discernible boundaries or transitions. Open-world games like *Skyrim* make use of such a grid topology for outdoor spaces (Skyrim is actually a hybrid, with interior spaces and dungeons acting as sub-lattices, attached to various points of a master "exterior" grid of tessellated "cells").

Minecraft is squarely in the grid camp.[51] The world is divided into chunks, each of which is 16 blocks long, 16 blocks wide, and 256 blocks tall. Chunks are loaded in the player's vicinity as needed, allowing for seamless travel. The just-in-time loading of chunks on the horizon, which supports the fantasy of infinite expansion and continuous traversal, allows Minecraft worlds to be "a perfect allegory for the desire for that space of infinite colonialism, enabled by the techniques of the grid" (Simon and Wershler, 2018).[52]

If chunk boundaries are invisible and transitions are seamless, it would seem that the operational logic of chunks has no bearing on the domain of player processes (indeed, I am encroaching slightly on the computation-level discussion here). However, it does because the promise of seamlessness fails in important ways.

The biggest problem for most players is that time effectively stops in a chunk when the player is far away; being unloaded from memory, the chunk cannot participate in any continuous processes.[53] This means that anything the player has to wait for, such as crop growth or furnace smelting, may not actually be progressing at home while the player is adventuring abroad. With tech mods that emphasize massive automation systems, this becomes even more troubling, which is why

[51] There are some mild lattice elements in Minecraft, since it has the alternate "dimensions" of the Nether and the End, reachable only through portals. Some mods add more dimensions and means of transitioning between them—Mystcraft, for instance, allows using magical books to "link" to arbitrary locations within countless dimensions, as an homage to the fiction surrounding the 1993 CD-ROM adventure game *Myst*.
[52] For Simon and Wershler, as for Siegert, the grid—in the abstract—is a "technique" rather than a strategy. In my analysis, specific given *applications* of grid techniques (e.g. chunk geography) can be considered strategies.
[53] Games typically don't bother to simulate ongoing non-player processes in areas that are far away from the player. However, in some games when a player re-enters an area, the program checks timestamps to see how much time has elapsed since an area was last loaded, and simulates the end result of a corresponding passage of time. So, for example, if your iron ore is supposed to take an hour to smelt in the furnace, and you go somewhere else in-game and return an hour later, the game may decide that your iron ought to be smelted by now. (This is how smelting works in Chucklefish's indie farming game *Stardew Valley* [2016].) Vanilla Minecraft does *not* do this: if you put your ore in the furnace and then go out on an expedition far from home, you are likely to find, upon returning, that it has made very little progress.

the popular industrial tech mod RailCraft allows players to build "world anchor" machines that force a chunk to stay loaded even when the player is far off.

Worse problems can occur if an automated system crosses chunk boundaries, because there will be a critical distance at which *part* of the apparatus will be loaded. When a machine tries to operate without all of its parts, the result can be unpredictable and even disastrous. Take redstone electronics, which sometimes rely on "clocks" that pulse a signal at regular intervals by cycling it through a loop. If any part of that loop crosses into an unloaded chunk, the pulse stops, and might not restart again when the chunk is reloaded.

Industrial mods, once again, up the stakes: RailCraft and other mods feature steam engines that require a constant supply of water in order to run. Allowing a boiler to run dry is not immediately fatal, but attempting to put water in a boiler that is *dry and hot* will cause a deadly and destructive explosion (if a boiler ever does run dry, the player needs to cut off the fuel and let it cool down before adding more water). A nice way to set up a steamworks is to pump water from a large tank or pool into the boilers by means of pipes. As long as both boiler and water source are loaded, all is well—and if both are unloaded, nothing can go wrong anyway. But what happens when the chunk containing the water source is unloaded, but the boiler is still running? With no incoming water, the boiler will soon dry up. Later, when the player moves back towards home, the water source is loaded again, and… boom! (Recall the destruction shown in Figure 1-4, page 14).

I am far from the only player to return home to find a big crater in the middle of my factory floor. Mezz, now a prolific modder, links his impetus for modding back to such an incident:

> I actually stopped playing for a long time. I set up a bunch of Buildcraft engines and because of some quirk in chunk loading, my cooling station was not in the same chunk as the engines, and when I went far away and came back, everything had exploded. And when I figured out that it was because of another quirk in Minecraft, I just kind of raged out and said, "Forget this game, I'm not going to play for a long time."

When Mezz returned to Minecraft a couple years later, he realized that he could put his programming skills to use in fixing infuriating quirks such as this one.

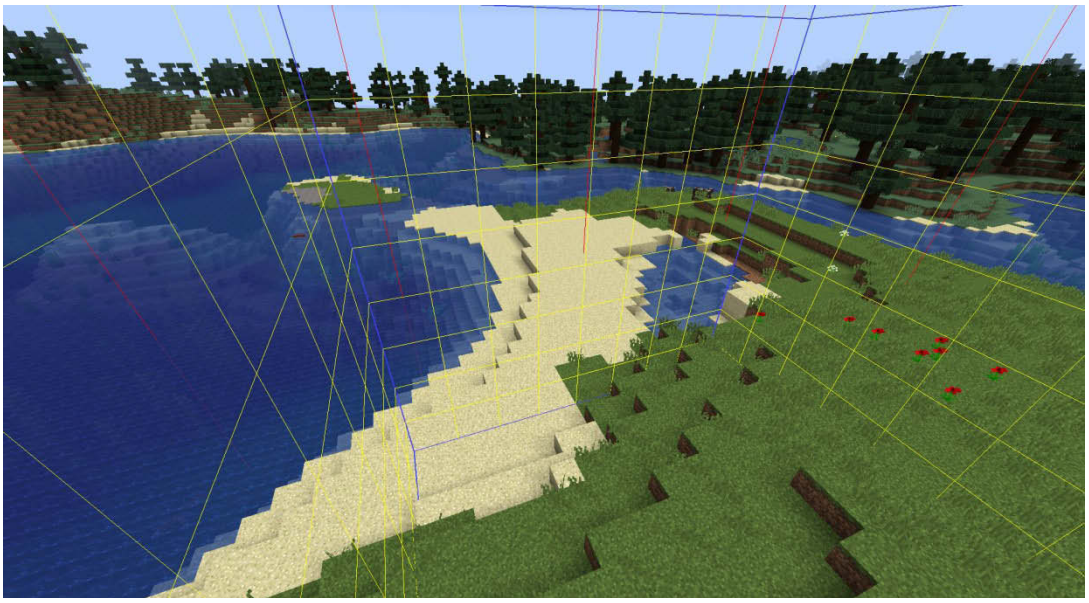


Figure 6-2. Chunk boundary visualization in Minecraft 1.13 (vanilla). (Screenshot by N. Watson.)

While never intended, chunk logics have become more than just a computational strategy: they are an actual game mechanic. Recognition of their importance to players is seen in the inclusion of features for visualizing chunk boundaries, previously offered in ubiquitous utility mods like NotEnoughItems and FTBUtilities, and now a vanilla client feature as of version 1.10 (Figure 6-2, above).

6.3.2.3 Chunks and computation

The previous section already stole some thunder here by describing the chunk-grid as a means of managing memory usage and processor load. However, a few more words can be said.

On the data level, a chunk is for the most part a collection of arrays (sequential lists) that provide information about the blocks contained therein, Each chunk is split into 16 vertically-stacked "sections" (each 16 blocks tall) which contain lists enumerating the properties of the blocks found therein. Chunks are the organizing principle for Minecraft save files. On disk, the world consists of one or more *region files* with the .mca extension. Each such region corresponds to a square of 32x32 chunks. When studying Minecraft save files using a tool such as NBTExplorer (Figure 6-3, below), we can see how data are organized according to region, then chunk, then section, and finally variety of data (block IDs, lighting, etc.).

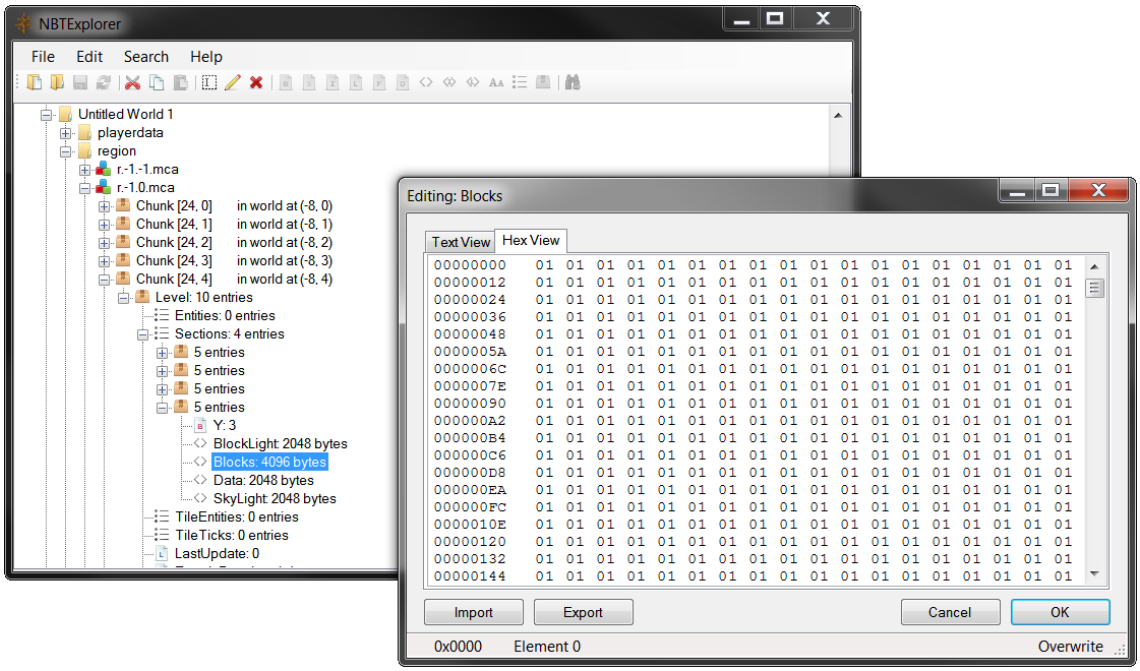


Figure 6-3. Viewing chunk data for a Minecraft world using NBTExplorer 2.8.0 by Justin Aquadro. (Screenshot by N. Watson.)

6.3.2.4 Chunks on the practice level

Although chunk logics are absolutely crucial to Minecraft's block-related computations, they recede somewhat from view on the level of the source code. Instead of requiring the modder to manage chunks as individual units with local block coordinates, the code provides a single `World` interface with which the modder can address any arbitrary block by its global coordinates. Modders can therefore accomplish much without having to worry about chunks at all. However, mods that deal with world generation, special lighting, or the transportation of resources and signals across distances all have to take a chunk-view of the world at some point. It turns out that there are, from a strategically rationalized perspective, right and wrong ways to do this. Mezz explains:

> I sometimes go on kind of like a crazy mission. Like, I'll be playing a [mod]pack and I'll find out that mods in general are doing a particular thing incorrectly. So, recently I was looking at world generation, and mods were creating stuff in such a way that would load more chunks, which would do more world generation, which would load more chunks, and I actually got into a pack where as soon as you loaded the game, it took, you know, more than a minute to settle down all this crazy extra chunk generation. So I started going around to a lot of the open-source mods, either making a pull request or talking to

> the author about how they might fix it up, so that we could, you know, fix this problem as a community.

The way Mezz frames his efforts here is notable: he does not merely request patches for problems on his favourite mods, but goes on a "mission" to "fix this problem as a community." This is strategic thinking—albeit an open-ended, non-dictatorial kind that invites collaboration from other stakeholders. The implication is that there is one best, most-rational way to address the issue, and Mezz wants modders to work together to find it and establish it as a new standard. This co-constructive effort corresponds to the early developmental stage of a new strategic *co-regulative* element of practice and its associated operational logics, which may come to be canonized in the future as a modding "best practice," operating from a "proper" position. It provides a glimpse of what other now well-established computational and practice logics, such as hook-based extensible mod-loading, may have looked like as nascent ideas.

6.3.3 Block-based metaphysics

6.3.3.1 Blocks and player process

The Minecraft player learns almost immediately that nearly all interactions with the game world are mediated by blocks, which are ubiquitous. Players know that most blocks can be struck until broken, and then picked up and carried around. Furthermore, carried blocks can be placed back into the world in different locations. Every block has a type, which can be readily ascertained from its visual appearance, but has deeper implications: different types of block can be easier or harder to break, be more or less flammable, allow or prevent monster spawning, require different tools to successfully harvest, emit more or less light, and so on. When broken, some blocks, like wood, drop portable versions of themselves. Others are transformed in the harvesting process, becoming a different type of block (stone becomes cobblestone) or even becoming an "item" which cannot be placed back into the world-grid, like coal (see Figure 6-4, below). Blocks, along with other non-block inventory items, can be transformed into entirely new types of items or blocks via crafting recipes (another operational logic unto itself).

In general terms, then, Minecraft blocks afford the player-process of *reconfiguring a Minecraft world* through the operational logic of block addition and removal, with possible block-type transmutations intervening due to harvesting rules or crafting recipes. More specifically, through this simple operational logic, blocks can become agents of protection (shelters to keep the monsters out), obstruction (stones that one has to dig through to get to the precious gems beneath), mobility (bridges and stairs), manufacture (using blocks as raw materials to make advanced items), architectural customization, and artistic expression.

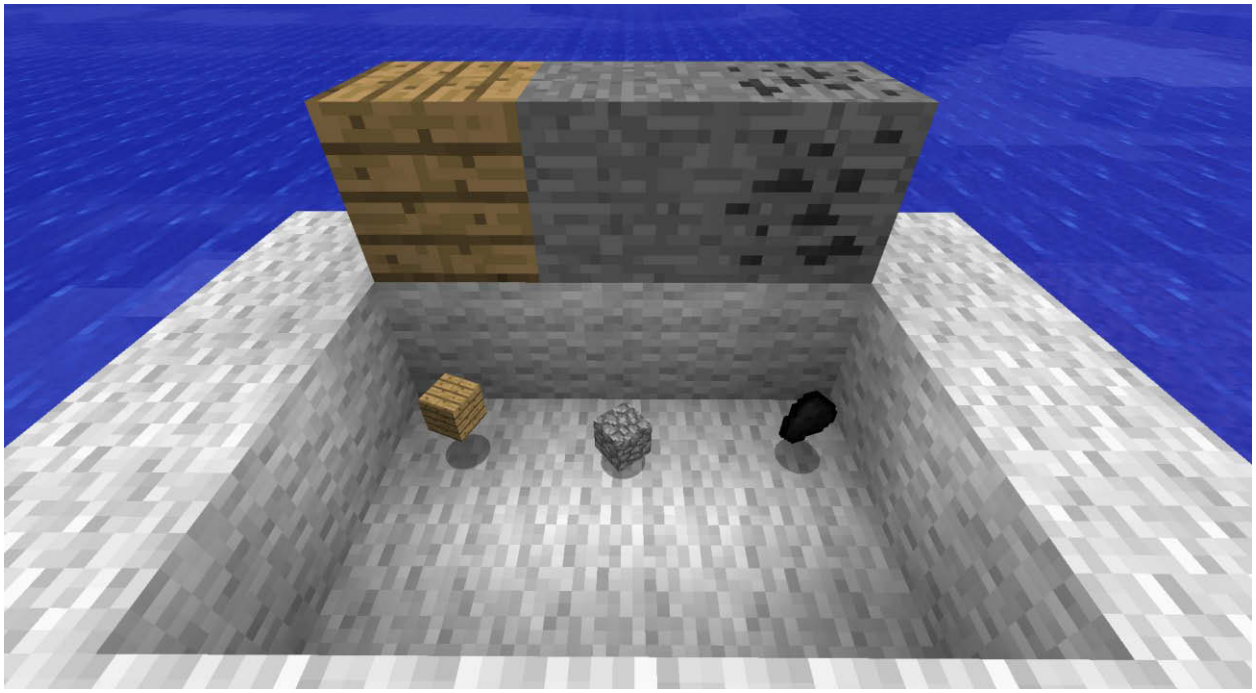


Figure 6-4. The relationship between blocks and items. (From left to right) When broken, a wood plank block drops itself as a item. A stone block drops a *cobblestone* block as an item (which will remain cobblestone if placed again as a block). A coal ore block drops a (non-block) lump of coal item. The block-items can be picked up and placed again as blocks (wood planks and cobblestone respectively); the lump of coal cannot. (Screenshot by N. Watson.)

6.3.3.2 Blocks and computation

Under the hood (as represented in a world save file), a block consists of two pieces of numerical data:

- A unique BLOCK ID, between 0 and 4095, that determines the block's basic type.[54]

[54] This is a minor oversimplification of the way that block IDs are actually calculated and stored.

- A "METADATA" or "BLOCK DATA" value, between 0 and 15, that can be used differently for different block types to account for variants, such as different textures of wooden planks, or different orientations of stairways.

There are actually two additional numbers that keep track of how brightly a block is lit by the environment, but these are beyond the scope of this chapter.

It may seem remarkable that a piece of data consisting of two small integers is expected to be able to do all of the things described above on the player level. However, there is a good reason for limiting the complexity of block data: a Minecraft world contains *a lot* of blocks! A new world I generated with default settings on Minecraft 1.9.4 starts out with a staggering 11,608,679 blocks (44,302,336 if counting empty air "blocks"). Java makes it easy to implement every individual thing in the world as its own Object, but making each individual block its own unique Object with a wealth of encapsulated data fields and attached methods would lead to slow runtimes and excessive memory usage.

Instead, the blocks in a world save file represented as a collection of arrays (which I will call "chunk data arrays," for reasons that will become apparent), containing nothing more than ID numbers and metadata values.[55] All of the other block properties and behaviours (appearance, hardness, intrinsic luminosity, physical shape, etc.) do not properly belong to the blocks themselves, but to a small set of master tables into which the block IDs serve as lookup keys.

6.3.3.3 Blocks from the coder's perspective

This all makes the source code itself rather dizzying, even to those familiar with object-oriented programming. The neat distinction between classes, representing abstract types of things, and objects, representing the specific, real instances of those things, gets severely distorted. If we allowed every block to be its own object, then we might expect Minecraft to have a general `Block`

[55] This statement is accurate for the way that blocks are represented in save files, up to Minecraft 1.12.2. The way that blocks are represented in runtime memory was quite similar until various changes were introduced 1.7.2 and 1.8 (see Subsection 6.3.3.4, below).

class, with the various block types being subclasses thereof, and actual blocks being instances of those subclasses. Instead, object instances of `Block` and its subclasses are the *abstract block types* themselves—one object not for every block, but for every general *type* of block (i.e. every in-use block ID). To complicate matters, there are several subclasses of `Block` to handle some of the more complex block behaviours, and these have their own instances. Figure 6-5 illustrates the data relationships between the `Block` class, two subclasses, two example block type objects, and several individual block entries in the world grid.

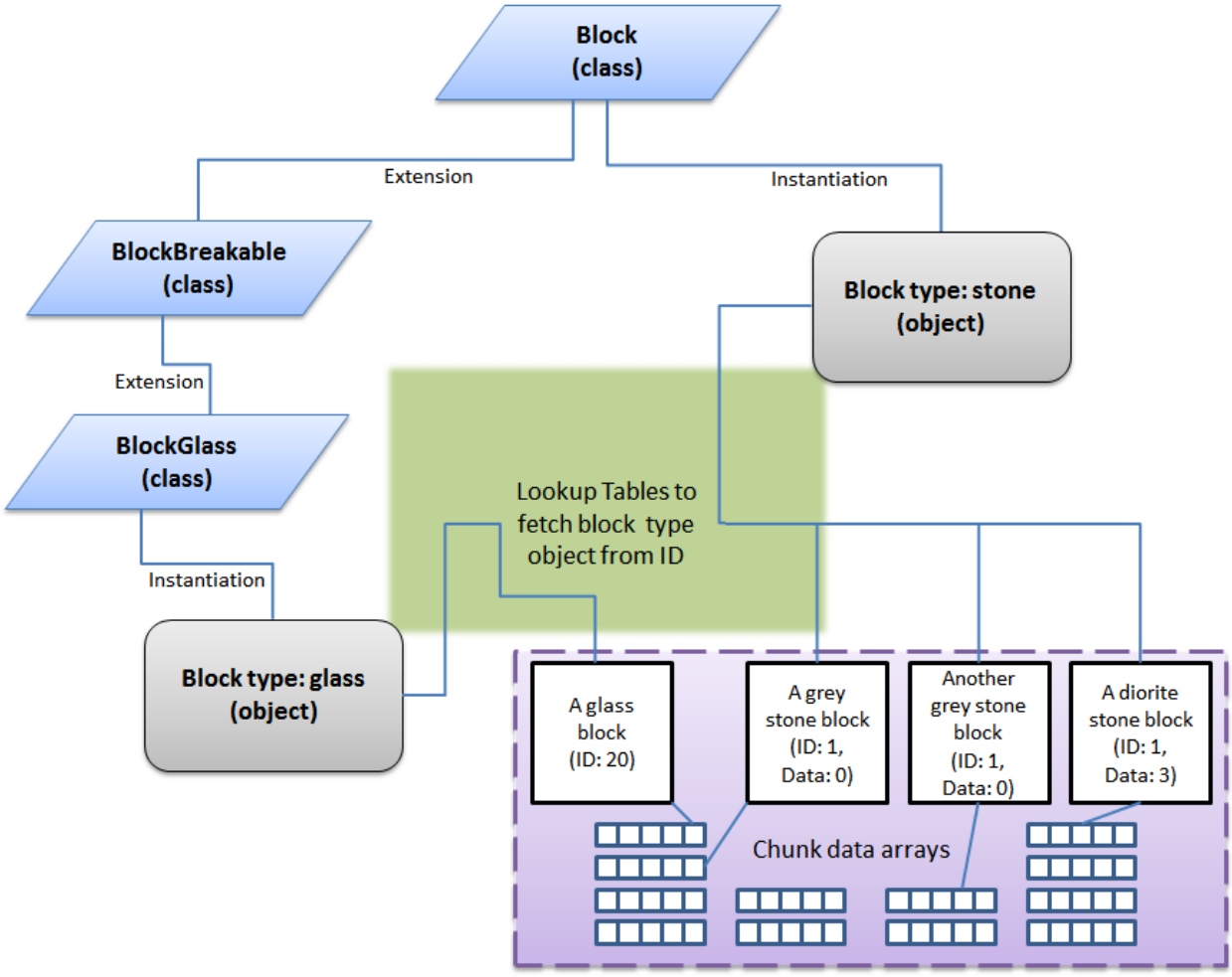


Figure 6-5. An illustration of the data relationships between the Block class, two subclasses, two example block type objects, and several individual block entries in the chunk data arrays housed within the world save files. The labels "glass" and "stone" are the actual internal names used in Forge source code for the objects representing these block-types. For clarity, this diagram uses pre-1.7.2 block ID numbers for its examples, and omits depiction of the abstractions added to the runtime representation of blocks in versions 1.7.2 (text-based block IDs) and 1.8 (block states). (Diagram by N. Watson.)

In my modding workshops (introduced in Subsection 3.6.4), students had to work consistently across all three levels of abstraction, just to define a new type of custom block that does nothing especially interesting. Some ran into trouble with the line of code from the main mod file in which an object is instantiated to represent the new block type:

```
public static final CustomBlock myCustomBlock = new CustomBlock(Material.ROCK,
        "customblock");
```

Here, `CustomBlock` is a subclass of Block. The name of the instance, which will represent a whole new *type* of block, is `myCustomBlock`. The text string "`customblock`" is a unique block ID, which brings up another issue: although block IDs are all in the form of integers in the chunk arrays and save files, they are (as of Minecraft 1.7.2) text strings from the perspective of the modder. The Forge Mod Loader maps text IDs to arbitrarily-chosen numerical ones at runtime, maintaining a "registry" to keep track of these mappings. On the one hand, a set of self-explanatory text strings is easier to work with than a bunch of arbitrarily-chosen ID numbers, but on the other hand, it becomes yet another signifier that the modder has to distinguish from other signifiers corresponding to different levels of abstraction. That is, the modder has to remember that `CustomBlock` (a class) is *a category of block-type*, that `myCustomBlock` (an object) is a *block-type*, and that "`customblock`" (the value of a data field on the object) is the *block ID* corresponding to any specific block of the `myCustomBlock` type.

The student errors I observed were the result of confusing these three similar-looking signifiers—for instance, the proper way to register the custom block with FML, as seen in Appendix B,[56] "File: ModBasic.java", line 57, is…

```
e.getRegistry().register(myCustomBlock);
```

…but students would sometimes attempt…

```
e.getRegistry().register(CustomBlock);
```

…or

[56] Complete source code for the basic custom-block mod I used, with explanatory comments, can be found in Appendix B.

```
e.getRegistry().register(customblock);
```

The latter two examples will result in compiler errors, because the name of an object is expected in the parentheses. The compiler knows that `CustomBlock` is a class name, not an object name, and it does not know what `customblock` is at all (because that was never a compiler signifier, but the hard-coded value of a piece of text data). Adherence to conventional Java and Forge nomenclature rules—which, according to my definition, are themselves practice-level operational logics—can reduce confusion. Java class names are always supposed have initial capital letters, while object instance names do not, but may contain internal capitals for readability. Forge documentation specifies that block ID strings are supposed to be entirely lower-case. In my workshop, most (though not all) students who ran into trouble with this code were not adhering to these conventional practices,[57] making it easier for them to get mixed up.

6.3.3.4 Scrambling to get a hold on block logics

The operational logics of blocks from the modder's perspective have changed significantly since Minecraft's official release, forcing modders to adapt by radically changing the way they write mods. Having surveyed the block logics family across all three domains, we are now in a position to consider the implications of these changes for modding practice. Far from being capricious, the trend of these changes has been towards increasing rationalization, causing what were once ad hoc modding tactics to be replaced by planned and sanctioned strategies.

Prior to version 1.2 (March 1, 2012)[58], only 256 block IDs were possible. This put a tangible limit on how many block-adding mods a player could install, and available block IDs were at a premium, as a 2012 conversation on MinecraftForum.net demonstrates:[59]

[57] The apparent similarity of all three signifiers is also the result of unimaginative naming in the example code I wrote for students. Taking extra measures to differentiate the signifiers could reduce confusion, which is why the version of this code provided in Appendix B uses the class name BlockNifty instead of CustomBlock.
[58] With version 1.2, the world save format was changed to accommodate larger block IDs. However, the Java code was not updated to make use of this change, so the 256-ID limit effectively remained until Forge patched it over in 1.3.

Fad:

> It has to not interfere with already existing block/item ids. 255 is the max number and I am [not] sure how to change that but you really dont need to make it bigger if you are dealing with blocks unless your adding like 100 new blocks into the game lol.

taa1taa:

> I really DONT AGREE with that! i have 9 mods installed at once and with already 4 mods i had to change configs! (and i have like 1 free id left) so yea. i am looking for a way to increase that in minecraft.jar files

taa1taa's problems go beyond running out of IDs. When they say that they already "had to change configs," they mean that some of their mods were trying to lay claim to the same block IDs, and they had to edit the mods' configuration files (using a text editor) to reassign block IDs in order to eliminate those conflicts. Even if a player's combined mod-added blocks do not exceed the limit, more blocks from more mods means a greater likelihood of conflicts. Pre-1.2 modders were under pressure to limit their use of IDs as much as possible, relying instead on the metadata fields or the use of potentially memory-hogging TILE ENTITY objects to implement different blocks as if they were variants of the same block, sharing the same ID number. Players also expected that, rather than hard-coding their ID choices, modders would provide configuration files, so that players could resolve any conflicts that arose from their particular combination of mods—which, of course, involved significant effort on the player's part. Importantly, any change to configured IDs would tend to invalidate prior world saves, because block IDs that previously meant one thing would suddenly mean something else, or nothing at all. These workarounds had a decidedly tactical flavour, as did the several "ID resolver" mods that some players used at the time to automatically choose non-conflicting IDs—although these latter measures provided a glimpse of the embedded strategy that would eventually be used to solve the problem in later versions.

[59] http://web.archive.org/web/20190116131118/https://www.minecraftforum.net/forums/mapping-and-modding-java-edition/minecraft-mods/modification-development/1423112-whats-the-highest-block-id-a-block-can-have

Boosting the quantity of block IDs to 4096 meant that modders and players could have approximately 30 times as many custom blocks,[60] which took significant pressure off of modders and players, and made some of these tactical workarounds less crucial. ID conflicts were still a problem, however, as there was no central authority allocating ID numbers to each mod—nor would that have even worked, if for no other reason than that there were enough mods out there to collectively exceed the 4096 limit, even if nobody would actually be trying to use them all at the same time.

A change in Minecraft 1.7.2 (October 25, 2013) eliminated ID conflicts (and obviated the need for ID resolvers) by replacing arbitrary numbers with descriptive ID text strings as the modder's (and server administrator's) point-of-contact with block types. As discussed above, they are still numbers under-the-hood, but Minecraft is able to dynamically assign these numbers to all of a mod's custom block types at runtime. More significantly, this change was the vanguard of a new *strategic* model that would completely reshape how modders approached block logic: the IBlockState abstraction.

The official Forge documentation, itself an open-source project, rather enthusiastically speaks for this complete re-rationalization of block-related coding.[61] I have reprinted these paragraphs to show how the rhetoric works to legitimize the new strategy:

> In Minecraft 1.8 and above, direct manipulation of blocks and metadata values have been abstracted away into what is known as blockstates. The premise of the system is to remove the usage and manipulation of raw metadata numbers, which are nondescript and carry no meaning.
>
> . . .
>
> How about, instead of having to munge around with numbers everywhere, we instead use some system that abstracts out the details of saving from the semantics of the block itself? This is where IProperty<?> comes in. Each Block has a set of zero or more of these objects, that describe, unsurprisingly, *properties* that the block have. Examples of this include color (IProperty<EnumDyeColor>), facing (IProperty<EnumFacing>), integer and boolean values, etc. Each property can have a *value* of the type parametrized by IProperty. For example, for the respective example properties, we can have values EnumDyeColor.WHITE, EnumFacing.EAST, 1, or false.

[60] Around half of the original 256 block IDs were already being used by vanilla Minecraft, meaning that the number of *available* IDs went from ~128 to ~3950.
[61] http://web.archive.org/web/20180917123140/https://mcforge.readthedocs.io/en/latest/blocks/states/

> Then, following from this, we see that every unique triple (Block, set of properties, set of values for those properties) is a suitable abstracted replacement for Block and metadata. Now, instead of "minecraft:stone_button meta 9" we have "minecraft:stone_button[facing=east,powered=true]". Guess which is more meaningful?

Seen in conjunction with the earlier transition to text-based block ID wrappers, this move appears to be part of a trend to position the modder as far from the raw data—and as close to the semantic surface-layer—as possible. The effort is not completely successful, as the documentation admits later on the same page: "Sadly, abstractions are lies at their core" because the metadata integer has not truly gone away. A modder who makes a block with its own custom blockstate properties must still implement methods to translate semantically-meaningful block states into integers from 0 to 15 for use in the save file, and vice-versa.

The new blockstate logic also goes hand-in-hand with 1.8's new way of handling block and item rendering logic, by separating these functions from the source code. The separation of code from graphical assets is nothing new. In fact, some scholars trace commercial game modding back to id Software's *DOOM* (1993), which facilitated player tinkering by separating content from process (Kelland, 2011). Minecraft has long made it easy to modify in-game textures by opening their corresponding PNG image files in an image editor like Photoshop or Microsoft Paint, but before 1.8, there was no way—short of hacking the hard-coded RenderBlocks module with a coremod—to actually change the *process* for rendering a block model. Thus, you could turn a rock pink, but you could not make it look like a pretzel.

Mods for 1.8 must include special asset files called "blockstate JSONs" and "model JSONs." JSON stands for JavaScript Object Notation, a syntax for storing information about complex objects containing data fields of arbitrary types as plain text. The way these files are used by Minecraft, they are somewhere between being process and data: they certainly contain no executable instructions, but they provide information about how a graphical model is put together, which the game interprets

as instructions on how to render it. This allows all manner of custom three-dimensional forms to be defined separately from the game code proper.

Such strategic changes on the part of Mojang are not just to please modders: they help Mojang's own developers as they prototype new blocks and models for future releases. It is much easier to filter and integrate contributions from different developers if they are modularized, as opposed to the previous "RenderBlocks" situation in which everyone had to tack code onto the end what was essentially a long chain of "if-then" structures, with a hard-coded case for every block type. That older method, like so much of Minecraft's code, originated out of a tactical practice of "quick and dirty" coding, perhaps going back to Notch's earliest game prototypes: the idea at the time was to create a fun proof-of-concept game that people would get excited about, not an extensible platform for years of ongoing commercial development and fan modding. The trajectory of renovations to the structure of Minecraft's source code seems to be away from those original tactics (some of which still persist in the codebase) and towards strategically planned, modular, extensible systems that happen—or are perhaps directly intended—to make modding easier. This can even be seen in operational logics at the computational level, such as when the logic for handling rideable entities (horses, vehicles) was generalized so that entities could ride each other recursively (Leavitt, 2013; a zombie on a horse in a minecart, perhaps). Leavitt (2013) identifies this as Minecraft increasingly taking on a "role as a platform for cultural production" (p. 27).

Rationalizing moves are not without their detractors, though. For instance, referring to the new 1.8 rendering system, modder MrCrayfish wrote on his website:[62]

> MrCrayfish has not been fond of the new JSON rendering system in Minecraft and when trying to convert his Furniture Mod to 1.8, it was a big task. Writing block models in JSON is annoying because you can't see what it looks like.

MrCrayfish's solution was to write a "Model Creator" tool for building models visually and automatically generating the relevant JSON. Many other modders seem to have found updating to

[62] http://web.archive.org/web/20171030131947/https://mrcrayfish.com/tools.php?id=mc

1.8 to be "a big task," as evidenced by the fact that it took years for some of the most popular mods to advance past version 1.7.10. Consequently, despite having been released in June 2014, version 1.7.10 remained popular among players who wanted mods well into 2017—one of the causes of the fragmentation across versions discussed by Zoll at BTM (see Chapter 5). Installing old versions is one of the tactics that modders and players have used to keep from getting swept under by the relentlessly rationalizing current of Minecraft updates. If strategic rationalization entails the elimination of unpredictable outcomes, as suggested at the beginning of this chapter, then it is ironically the unpredictability—from the modder's point of view—of Mojang's updates that drives modders to (usually temporarily) reject and resist the new, strategically-refined offerings.

6.4 Other operational logics of computation and practice

Some additional operational logics that are only visible within the computation and developer practice domains bear witness to the ongoing strategic rationalization of Minecraft modding.

6.4.1 Forge infrastructure: events, coremods, and public utilities

Chapter 4 described how Forge/FML facilitates modding, and reduces the need for modders to change original Minecraft files directly, by adding "hooks" at key locations that can trigger customizable behaviour. One of the key ways in which these hooks make themselves available is through the *Forge event model*. Forge maintainers identify points in the code where modders might like to intervene, such as whenever a new mob spawns or a block is broken. These pieces of code are rewritten so that they will dispatch *event* objects, containing information about what is happening, to an object called the *event bus*. Modders can direct their own custom methods to "subscribe" to receive notifications of specific types of events from the bus. This means that every time an event of that type occurs, the method—called an *event handler*—will be executed, contributing whatever extra procedures the modder has designed. Some events are even "cancelable," meaning that the event

handler can veto whatever is happening—for instance, nullifying any combat damage received by the player under certain circumstances.

The event model is presented as a carefully designed and sanctioned way of intervening in Minecraft's processes—recall the "don't make a core mod" forum post (Chapter 4) in which "use events" was one of the top suggested alternatives. Rather than hacking bytecode to add their own messy and potentially "unsafe" hooks, the author of that post wants modders to use one of the hooks that Forge has already implemented, via an approved, strategically centralized code-hook infrastructure (the event bus).

The range of things one can do with the event model has steadily expanded over time, reducing the need for coremodding. However, one modder I spoke with (capitalthree) explained that the temptation to resort to bytecode hacking (also called binary patching) remains when it comes to having to work with other modders' packages, because unlike Forge, they do not have much incentive to make their *own* code hookable by dispatching events of their own[63]—even though Forge is set up to allow this:

> That's the problem you run into with Minecraft and events. The event model is great, but mods are very rarely good at actually using the event model as a producer, generally only as a consumer. So mods tend to be doing a bunch of their own things that then don't have good ways to hook into them.

What this quote suggests is that modders have, for the most part, gotten on board with Forge's rationalizing strategy for a "proper" way to alter vanilla code, but (with some exceptions) they have not internalized this strategy to the point of designing their own mods around it for the benefit of others; this implies that on an individual level, modding work maintains a tactical valence. Naturally, mods that are not intended for publication—those that, to echo Eric Raymond from Section 2.3, "scratch a developer's personal itch," but do not subsequently get released—need not provide a rational interface for other mods to use. It is not clear just how much modding activity is

[63] For a more comprehensive explanation of the Forge event model, how it differs from other types of hooks, and what capitalthree means by using it as a producer versus a consumer, see Appendix C.

personal and unpublished, since it is, by definition, invisible. Some published mods, such as the total conversions of *Better Than Wolves* and *BioLithic* (discussed in Chapter 7), can also afford a more tactical, non-standardized approach, since cross-compatibility with other mods is not among their design goals. However, when a mod is nominally intended for public use in conjunction with other mods, but does not provide an interface for other mods to smooth out the wrinkles that pop up in mod interactions, further tactical wrangling in the form of bytecode hacking may be the only solution.

Mezz, for his part, sheds a little more light on the philosophical aversion to bytecode hacking and coremodding seen on the Forge forums. Since Minecraft version 1.8, he has been producing a mod called Just Enough Items (JEI), which provides players with a directory of all possible blocks and items, instructions for how to acquire or make them, and information on what they are good for. A reference (or "recipe manager") like this becomes essential when one is dealing with thousands of mod-added blocks, especially if the modpack curator or server administrator has tweaked recipes so that they no longer match the canonical versions found on popular wiki pages.[64] In fact, recipe managers have become so ubiquitous that modders sometimes just assume that players will have one, and thus don't worry about providing online documentation for all their mod's recipes.

JEI was itself a successor to a different author's defunct mod Not Enough Items (NEI), which itself had taken up the mantle of Too Many Items (TMI). NEI was somewhat notorious for being a hugely popular mod that required a coremod library. Consider this quote from Mezz (emphasis added):

> One of the things I wanted with JEI was no dependencies and no coremods, because I wanted it to be very compatible, and I wanted it to be something you could have in your developer environment without messing up anything. One of the troubles I had with NEI

[64] Mods like JEI thus accomplish goals similar to those of WAILA and the "fruit phone" mods seen at BTM, in that they help players to manage the ever-growing bodies of information associated with modded Minecraft play.

> was as soon as I had to develop stuff with it, I had to bring all that code to be running alongside mine, in my developer environment, and often it required a whole bunch of hacking stuff together to try to get it to start. I mean, it was just not very friendly. So having something that follows best practices was kind of one of the main goals that I had. And to get around . . . the coremod stuff that NEI had—a lot of it was extraneous features that I just didn't add to JEI. I wanted JEI to be super-small, like as small as it could be. . . . Instead of coremodding where I directly added to the Minecraft code, I basically made Forge my coremod, and that's kind of the accepted way to do it. (emphasis added)

It is significant that making something "that follows best practices" was one of Mezz's *main* goals. It suggests a desire to participate in an emerging modding ecosystem in which rationalized best practices exist. Where those best practices come from is another matter. Were they developed decades ago by software development gurus? Are they "just there" already, and not worth worrying about? Are they dictated by authoritative voices in the modding community? Based on our conversation, Mezz believes that best practices can be built by community consensus, but not everyone seems to hold compatible views or even agrees that best practices need to exist, as we shall see in the next chapter.

These questions notwithstanding, relying on Forge itself to do all of the binary patching work is the "accepted way to do it" because, according to Mezz:

> Then everyone else can benefit and no one's kind of stepping on each other, because they're using Forge as shared codebase. So other mods can use the same hooks and we're not going to blow up or draw over each other, or anything like that.

Mezz has been helping out on Forge since mid-2017 as the Pull Request (PR) manager, reviewing suggested code changes that others have contributed to the project and deciding whether to accept, reject, or follow up with the author. When introducing Mezz on the Minecraft Forge forums, Forge founding member LexManos wrote:

> Basically, he's the guy you yell at if your PR/Issue is rotting on github. He's the guy who is tasked with reminding me that they exist. He will be the one responsible for filtering all the PRs/Issues before

```
they get to me. So I don't have to deal with telling you guys to follow
the standards like making a test mod, posting logs, etc..
```

But Mezz sees his role not just as a gatekeeper, but as a facilitator of effective collaboration and provider of much-needed guidance:

> [LexManos] was pretty much the only one reviewing [pull requests], and often the feedback he gave was, like, pretty light, so people didn't really know how to fix it. So since I had a good sense of that, I asked him if I could help out with the pull requests and try to be like a front line on those, so I could help people fix up their pull requests before he would do a final review.

And later:

> I do communicate a lot with other modders, especially through the pull request stuff. I mean, we'll often take it to a side channel, and they'll just ask me, you know, what's going on? Why isn't this being accepted? What do I need to do? What does this feature need to look like?

For Mezz, the strategic rationalization of Forge modding is cast in terms of the creation of public goods. In fact, issues that arise due to multiple modders writing and submitting incompatible code can serve as the impetus for building new Forge infrastructure for public use, though the process can take a while because it needs to be done right:

> Sometimes with big features it'll take a long time. . . . Something that a lot of mods have been doing is adding extra tabs onto the player's inventory so they have additional screens they can look at. So if you have a backpack, you'll have a backpack tab and things like that. The trouble is when you have three different mods doing it, you have tabs overlapping and going crazy. So, they wanted to make it in forge. But if we put it in forge, it needs to look really good—it needs to be perfect. (Laughs) So it's been going through a lot of review process and that kind of thing.

Mezz's quote emphasizes that the rationalization of modding can bring with it the professionalization of modding—the normative expectation that the fruits of unpaid fan labour need to meet the same quality criteria as media content that is produced and sold commercially. Gallagher et al. (2017) noted a similar trend in the emulation of the commercial game studio's

professional development practices by Skyrim modders. For Minecraft modding, professionalization is also driven partly by the fact that many of the modders—particularly from the sample I interviewed—were studying, or held jobs, in information technology, computer engineering, or a related field.

6.4.2 Source code: using the right tools

The modders I spoke with also embraced the use of comprehensive, strategically-designed packages of coding aids in the form of modern Integrated Development Environments (IDEs), though they noted that not everyone had bought in. Capitalthree explained to me that modders seemed to be averse to his preferred programming language, Kotlin,[65] because of "type inference." When declaring a variable to store some value in Java, one has to say outright what type of data it will be (e.g. an integer, a decimal fraction, some text), but Kotlin will infer the proper type based on the assigned value, which some people find confusing. Capitalthree pointed to IDEs as a means of resolving such confusion:

> This point I will admit is probably a matter of opinion, but my answer to that is people should just always work in an IDE. And then if you want to know what something is, you mouse over it and your IDE will tell you. And I, to this day, am baffled by how many people go well out of their way to avoid using modern tools.

The most popular IDEs among Minecraft modders are Eclipse and IntelliJ IDEA, and Forge provides documentation for setting up a workspace in either one. The majority seem to prefer IntelliJ, based on responses from my informants as well as remarks I observed in public IRC channels like #minecraftforge. For Mezz, what sells IntelliJ is its quality-of-life features that save programmers from their own mistakes:

> I really like the static analysis tools. I use null annotations, like, religiously in JEI. That was one of the things that was part of the core design of it. I want to . . . construct it in a way that it's so safe that I barely have to maintain it. So it was a lot of work up front, but

[65] Kotlin is different from Java on the source code level, but it compiles into the same sort of Java bytecode, and runs on the same Java Virtual Machine. Kotlin, Java, Scala, and other languages with these properties are collectively called "JVM languages."

> now it's at a point where I'll get a compiler warning when I'm doing it wrong. And that really is helpful, a lot, so I'm a big IntelliJ fan. But, I mean, you can't force everyone to work in an IDE if they don't want to.

Code annotations are an IDE-based strategy through which a programmer tags parts of the source code in such a way that the compiler will throw a warning if the programmer makes certain common errors. By "null annotations," Mezz is referring to IntelliJ's `@Nullable` and `@NotNull`, but a full explanation of these is beyond the scope of this chapter. Instead, I will illustrate the principle with another, similar annotation, `@Override`, since we are already equipped with the tools to see its utility. Appendix A provides an explanation of the concept of method overriding in Java, but briefly, it is when a subclass implements a method with the same name as one on its superclass. If the method is activated on an object of the subclass type, the subclass's version will override the behaviour of the original method. Any time a programmer writes a method that is supposed to override the superclass, they can tag a method with the `@Override` annotation. This means that if for some reason overriding is *not* actually happening—perhaps because of a typo in the method name, or because its name in the superclass has changed—a compiler warning will alert the programmer to the problem. Without the annotation, a typo in the method name would make it look just like any other ordinary method to the compiler. The use of annotations like `@Override` and `@Nullable` are part of a strategic apparatus for quashing unpredictable outcomes on large, collaborative coding projects—one that some modders enthusiastically embrace.

6.5 Counterpoint: tactical modding is alive and well

While the modders I spoke to were generally happy to reap some of the benefits of increasing rationalization—especially when it comes to IDE tools—that does not mean they are all necessarily on board with the project to impose a strategic order on modding practice. Some of them consider their practices non-strategic, despite adopting some of the emerging rationalized practice. Part of the problem is feeling that they are unrecognized and alienated from the centres of strategic planning.

One modder opined that the Forge project lead "doesn't let anyone in on the planning as far as I've seen. He has some trusted people that help him with Forge, obviously. But… pretty much a dictatorship." Furthermore, the Forge web forums and IRC channel are sometimes perceived as hostile to novice modders. According to another modder, this "is exactly why people do not really always follow [their] 'design guidelines'" such as not making coremods.

Modder LShen rejected the notion that most of their modding activity is strategic at all (emphasis added):

> What Mojang does is strategy at player experience, etc…. [LexManos] does strategy for modding. And what we do is ad-hoc solutions that get reused and adapted. And to be honest, any code in Minecraft mods that is not easy to extract and generalize will need tweaking when used in even a slightly different context… so I cannot call this strategic or planning.

Mezz and LShen appear to offer two contrasting visions of the role of top-down strategies in modding. In Mezz's view, they are about collaboration, consensus, and the creation of public goods; but for LShen they imply exclusion and elitism, and are often not as nimble as the shareable, re-usable, ad-hoc creations of tactical modding. Both want to see modding solutions being shared for the good of the community, but endorse different approaches to issues of standardization and adaptability.

This is far from being the only contested terrain in the modding world. Differing perspectives on the role of Minecraft modders, the right way to make mods, and who has the authority to decide these issues, are as old as Minecraft itself.

While this chapter has taken a "standard reading" approach to a narrative of increasingly-rationalized modding practices coalescing out of a mixed bag of ad-hoc tactics, we now turn (in the following chapter) to a dialogic look at how that coalescing is far from straightforward, and what strategies and tactics members of the modding community use to try to claim a piece of the process.

VII. MODDING DISCOURSES

> *When I started modding, we didn't have Minecraft Forge. When I got back into modding [later on], it was basically political suicide* not *to use it.*
>
> —Omira, Minecraft modder

Modders are entangled in a web of social interactions that tug at their practice, pulling it this way and that. It would be easy, as a player and mod consumer, to perceive celebrated modders as virtuoso auteurs, or as inventors who create new possibilities out of thin air, but this obscures the processes and social relations involved. Equating them to "hackers" is not informative either: the hacker mystique is an image of radical autonomy, of someone alone with the code, free of other distractions. These are Mackenzie Wark's hackers, who "use their knowledge and their wits to maintain their autonomy," who are "not joiners" (2004, para. 005-006). Some modders might fit this description, but on the whole, an accurate description of Minecraft modding cannot escape the need to account for the social construction of the practice.

This chapter explores how modding practice is constructed, refined, and reinforced through discursive paratexts, which serve as vehicles for circulating gaming capital (Consalvo, 2007; see Subsection 3.3.3). I take an expansive view of what constitutes discourse, however: obviously forum posts and Discord chats are discourse, but so are technical references, tutorials, the sociotechnical architecture of websites, and even Java code itself—although these things can more easily get away with presenting themselves as monologic.

Although I speak of "paratexts," there is no usefully distinguishable primary "text" in this case. One might suggest that mods are the primary text, since these objects are the focal points of this tangled social-cultural web—but in what way are they "read"? A mod's source code is certainly a text but not *the* text, because modders don't make mods in the hopes that thousands of people will read and enjoy their source code. The gameplay elements added by a mod, in the specific (a narrative of a player's encounter with those elements) or the abstract (a formal encapsulation of the

mod's rules and mechanics), constitute a text, but one that can exist only when attached to Minecraft itself (not to mention the fact that most of them also must attach to Forge), so these are already adjunct texts or paratexts. What about Minecraft itself? No such luck—we saw in Chapter 4 that there are many Minecrafts: a horizontal branching of editions and a vertical proliferation of versions. Then there is the co-creative, forged-in-discourse nature of Minecraft's original design, as described by Redmond (2014; see Subsection 2.3.2), and the feedback loop by which modders' ideas have been incorporated into the base game. Genette (1997) saw paratexts as objects that shaped one's *reading* of the text, but Minecraft is *objectively transformed* through interaction with its paratexts. Lunenfeld (1999) provides a view that more comfortably fits the Minecraft moment, arguing that media convergence has made it "impossible to distinguish between [paratext] and the text" and that "the backstory—the information about how a narrative object comes into being—is fast becoming almost as important as that object itself" (p. 14). In a nutshell, all of the different types of texts considered in this chapter function as mutual paratexts to one another.

Before getting to the dissection of all this discourse, I provide a brief account of the internet venues in which much of it takes place, including—where relevant—a discussion of how the formal architecture of these virtual spaces has been adapted to govern and facilitate modder discourse, and the roles and positions that modders adopt within these spaces. Following that, I discuss specific examples of how modder discourse circulates gaming capital, with an eye to how discursive acts embody, advocate, or lay claim to tactical versus strategic positions. This part of the analysis consists of three themes: Forge software infrastructure as discourse; the dissemination and co-regulation of modding expertise; and issues of authorship and intellectual property.

One final note of caution: throughout this chapter I tend to label discursive actions as having a strategic or tactical character, but these proclamations should be taken with a grain of salt. As I will explain at the end of the chapter, apparently-strategic discourses might more accurately be described as expressions of a *fantasy about strategies*.

7.1 Venues of discourse

7.1.1 Asynchronous message boards

7.1.1.1 MinecraftForum.net

Probably the largest and most active discussion board for all things Minecraft, but not necessarily for modding, this forum was established in June 2009 during Minecraft's Classic version phase. The founding member, citricsquid, was an early adopter who was motivated to create a new space for the community following the original forum on Mojang's website "succumbing to masses of spam".[66] Following rapid growth coinciding with Minecraft's Alpha release—over 1,000,000 page views per day and more than 50,000 registered members by the end of 2010—the site was purchased by emerging web giant Curse, which runs a network of message boards, fan portals, and informational wikis for popular computer games (Curse was itself acquired by Twitch.tv in 2016). By December 31, 2017, the site reported 4,954,330 registered members, with a staggering 29,583,883 messages in 2,510,265 threads, and over 4.6 billion page views.

MinecraftForum.net was affiliated from the outset with MinecraftWiki.net (now Minecraft Gamepedia), described in Subsection 7.1.3. The use of the *.net* top-level-domain for both sites may have been an intentional choice to establish some mnemonic congruence with the game's official site Minecraft.net—or it may simply be because *.nets* are cheaper than *.coms*.

The organization of boards on MinecraftForum.net is itself a discourse reinforcing a particular way of thinking about play and modding activities. There are 69 forums in 11 top-level categories (used to group together boards with similar themes within the main page listing), and an additional 72 boards when counting nested subforums (some, but not all, boards have one or more sub-boards attached to them).[67] The categories are *Minecraft: Java Edition*, *Mapping and Modding: Java Edition*, *Servers: Java Edition*, *Minecraft* (Bedrock-based versions), *Minecraft: Editions* (primarily legacy

[66] https://web.archive.org/web/20160407065746/http://rmbls.net/post/16577257153/history-of-minecraftforumnet-and

[67] These and other MinecraftForum.net statistics in this chapter are accurate up to 31 December, 2017, unless otherwise specified.

console editions), *Minecraft: Pocket Edition*, *Support*, *Show Your Creation*, *Forums*, and *Off-Topic*. These are aspects of the forum's *architecture*, acting much like their physical antecedents—and, like those antecedents (e.g. the values expressed in the acoustic properties of an opera house, the social relations imprinted on its provision of seating and sight-lines, the management of crowds through the layout of its hallways and doors), they are not value-neutral.

Case in point: the forum's organization has changed many times as categories and boards have come and gone, been consolidated or split up, but the most revealing changes are seen in late 2017, when citricsquid reorganized and renamed many boards to reflect Mojang's rebranding of Minecraft editions following the Better Together update (his explanation for this move was quoted in Subsection 1.3.2). Previously, the Java Edition category was called simply "Minecraft" (and "Servers: Java Edition" was "Servers"), there was no category for Bedrock-based versions separate from Pocket Edition, and top-level categories existed for each of the console editions. The forum reorganization is an endorsement of Microsoft's branding, discursively demoting Java Edition (and its mods) to a position of lower significance. However, only a dialogic reading, aware of the prior structure and of citricsquid's justification, makes that effort visible: the average (and especially the new) user encounters the forum structure in a monologic mode, concealing the social processes that go into site design and making the organization appear natural and objective. This tendency for forum architecture to be seen as a fixed given is accentuated by the fact that it sits as "background" to the more obviously dialogic activity that is the message board's *raison-d'être*.

Yet there is a contradiction, an apparent unwillingness to fully commit the forum architecture to Microsoft's preferred branding, or an acknowledgment of how the majority of visitors still use the site. Three of the eleven top-level categories are still dedicated to Java edition, and they all appear at the *top of the page*, with the purported flagship edition showing up nearly one-third of the way down. Forum activity also reflects the continued dominance of Java Edition. The site indicates how many users are currently viewing each message board, and on an evening in November 2018 I

counted 1,110 active view-counts in the various Java Edition boards—536 in the Minecraft Mods area alone—compared to 29 in the entire Bedrock section.[68]

If we zoom in on *Mapping and Modding: Java Edition*, we find that it contains six boards with an additional 10 sub-boards (only sub-boards to the Minecraft Mods board are named in the list below; quotations are taken from the description lines for each board; square brackets indicate additional explanatory remarks that I have provided):

- Minecraft Mods (5 sub-boards)– "Post and discuss your Minecraft mods here!"
    - Mod Discussion – "Information about specific Minecraft mods and technical assistance for using and installing modifications."
    - WIP Mods – "Incomplete mods only. Everything before version 1.0"
    - Requests / Ideas for Mods – "Post your requests and Ideas for mods here!"
    - Modification Development – "Assistance in the creation and development of Minecraft modifications."
    - Mod Packs – "Find and download user-created mod packs here"
- Maps (2 sub-boards) – "Share and discuss your Minecraft maps here!"
- Resource Packs (3 sub-boards) – "Changing the look and feel of Minecraft"
- Skins – "Show others your skins, request skins and share information on how to create skins" [This refers to customizing the look of one's Minecraft avatar]
- Minecraft Tools – "All 3rd party tools and editors for Minecraft belong here." [This would include things like MCEdit and NBTExplorer]
- Mapping and Modding Tutorials – "Post your mapping and modding tutorials here"

7.1.1.2 MinecraftForge.net

The official forums for discussion and support requests regarding the Forge API are frequented by members of the Forge development team and other prominent modders. The site dates back to April 2012, and as of September 31, 2017 there were 103,639 registered members (the number of active accounts is likely much lower). The Forge forums are much smaller than MinecraftForum.net, with the following categories and boards:

CATEGORY – Minecraft Forge

- Releases – [This board only contains announcements of new Forge releases, with download links. Posting is limited to site staff, with most of the announcements written by the current Forge project lead, LexManos.]

[68] It is possible that the same user is counted multiple times if they have multiple browser tabs open in different boards, so these numbers should not be regarded as the actual total quantity of readers in each section. However, even differences in browsing behaviour cannot account for the large discrepancy.

- Support & Bug Reports – "Help with Forge goes here. Refer to Modder Support for help with Forge modding. You MUST read the EAQ before posting."
- Suggestions – "Suggestions for new Forge hooks, interfaces, etc…"
- General Discussion – "Feel free to talk about anything and everything related to Minecraft Forge in this board."

CATEGORY – Mod Developer Central
- Modder Support – "This is the support section for those modding with Forge. Help with modding goes in here, however, please keep in mind that this is not a Java school. You are expected to have a basic knowledge of Java before posting here."
    - ForgeGradle – "Help and support for the new ForgeGradle system of building mods."
- User Submitted Tutorials – "Tutorials for creating Forge mods by users, for users!"

CATEGORY – Non-Forge
- Site News (non-forge)
- Minecraft General – "Anything related to Minecraft, but doesn't have things to do with Forge, goes here."
    - Let's Plays and other Videos!
    - Texture Packs
    - In-game Creations
    - Servers
- Off-topic

CATEGORY – Forge Mods
- Mods – "Various mods with boards that are not so busy. Ones that get busy may be promoted to their own multi-board section, or dead ones will be removed."
    - Outdated or WIP Mods
    - [7 other sub-boards for specific mods or modders]

The site reports the number of threads and posts in an idiosyncratic way, so some statistical information was not captured by my semi-automated survey method. A count on October 14, 2017 yielded 252,588 unique posts, but the total number of *threads* is unknown as this would have required pulling data from the landing page for each individual board. The overwhelming majority of activity is confined to three boards: Modder Support (including the ForgeGradle sub-forum), Support & Bug Reports, and Forge General. Collectively, these contained 244,461 posts in 40,659 threads.

In an inversion of the organization at MinecraftForum.net, the Forge boards are *all* about mods, with a smaller section for non-mod-related Minecraft play. The Forge site also trades in Forge mods only, while MinecraftForum.net has its share of jar mods or those that use another platform, like LiteLoader and Bukkit.

The architecture of the Forge boards makes a clearer distinction than MinecraftForum.net does between spaces for modders and those for players. The use of a dedicated "Mod developer central" top-level category, as well as maintaining separate forums for mod-player versus mod-maker support—with descriptive text imploring the potential poster to use the right one—reinforces this separation. "Soft" gatekeeping is also incorporated into the design, as in the warning that the modder support forum "is not a Java school."

7.1.1.3 MCreator Forums

MCreator's website (https://mcreator.pylo.net before December 2017, https://mcreator.net afterwards) features a robust set of forums for supporting modders and promoting MCreator-made mods.[69] They are robust in that they provide a high degree of semantic granularity for organizing topics. They are also focused entirely on MCreator-related discussion, with no sections for general Minecraft or off-topic discussion. As of September 16, 2017, the MCreator forums reported 27,132 posts across 8671 threads, and 303,118 registered users. These figures underscore the problem of reading too much significance into the number of registered users, given that the Forge forums showed a third as many users but ten times as many posts.

MCreator's forum organization is flat (no category groupings or sub-boards), with nine boards total:

- Help with MCreator's application – "If you don't know where a button for *that thing* is or how to make something in MCreator, ask other MCreator users here and they will help you!"
- Feature requests and ideas for MCreator – "Would you like to see something new implemented in MCreator. Suggest it here and if it gets enough support, we will do our best to implement it!"
- Mod showcase and discussion- "If you would like to showcase your mod or ask other users on opinion on your mod, make a forum topic here. This is a place to discuss about the mods you post on community pages."
- Advanced modding – "Discussion about advanced modding belongs here. Custom events, variables, global events and such things."
- Help with MCreator modding – "Do you need some help with understanding how modding works? Look for an answer here."
- User side tutorials – "Would you like to contribute a tutorial for MCreator? Post it here and promote your YouTube channel if you have videos."

[69] MCreator was introduced in Section 1.5.

- Bugs and solutions - "Have you found a bug and can't wait for it to be fixed? Search for the workarounds here."
- Mod ideas – "Do you have an idea for a mod and are looking for a team to make a mod using MCreator together? Ask here!"
- General discussion – "General discussion related to MCreator and/or Minecraft modding which does not fall into categories above belongs here."

One striking pattern in the board descriptions is that most of them address the user as a potential poster—"This is where you can make a post about X." Only the Bugs and solutions appears to interpellate the user as a thread reader, encouraging them to "search" for a solution among existing posts—by implication, *before* making their own topic. This is a common refrain on support boards, and another example of a soft gatekeeping function.

As with MinecraftForge.net, support boards for MCreator are partitioned, with separate forums for help with the MCreator program, help with making a mod, and "advanced" modding. All of these, unlike the Forge boards, target modders exclusively—there is not much for players to see here, because MCreator is really just a tool to facilitate Forge modding. The end-user player has to install Forge locally to use the mods, but never needs to touch MCreator. The "advanced" category is interesting because it pertains to techniques that MCreator allows but does not try to scaffold with its collection of "elements"—the sorts of things that require doing some work "under the hood," entailing unfiltered exposure to core Java concepts. This allows us to read into MCreator's design goals and how its developers imagine their audience as being divided between a majority that are satisfied with the basic features, and a significant minority that want to go beyond those limits.

7.1.1.4 r/feedthebeast (and the rest of Reddit)

Reddit, like a web forum, affords asynchronous, text-based communication with optionally embedded multimedia content. However, Reddit differs in important ways from the other sorts of web forums discussed above. As in forums, posts are arranged into threads, but Reddit presents replies in a branching tree format rather than the flat, linear presentation used in most forums. A key aspect of Reddit's social affordances is its karma system, whereby users may "upvote" or

"downvote" others' posts based on their perception of how worthwhile those posts are to the community. Instead of a strict chronological ordering, the most-upvoted posts and comments may be promoted to the top of the page, while comments that are sufficiently unpopular are hidden from view unless the reader clicks to reveal them.

The most obvious thing that distinguishes Reddit from the web forums discussed above is that it is much bigger and its topics are general in scope, not Minecraft-specific. It lacks the kind of multi-level semantic organization seen on MinecraftForum.net, and the number of individual "boards" (called *subreddits*) is uncapped, with each board being a largely autonomous community created and maintained by whichever user had the will to start it up, and very little in the way of centralized control or curation. Thus there is no "Minecraft section" on Reddit, but rather a loose collection of Minecraft boards, some of which are more active and relevant than others, and none of which contain any subforums for further organizing discussion.

Reddit does not seem to pull a significant portion of discussion about Minecraft mod *development*. Much of the conversation on Reddit is consumer-oriented rather than developer-oriented: users discuss the relative merits of available mods, ask for recommendations, promote their own curated modpacks, and assist each other with technical issues they have installing or playing with mods. The verb "modding" is used in these subreddits to mean *installing and using existing mods with Minecraft* rather than *creating mods*. Topics have names like "I made a cobblestone generator using only Quark and vanilla," "What are the best mods in your guys' opinions?" and "Did I misunderstand how Applied Energistics works?" This is not to say that modders don't participate. They promote their own mod releases, provide development updates demonstrating amusing bugs, and weigh in on players' questions, suggestions, and creation showcases. The provision of "flair"—a tag attached to one's username—for modders on the r/feedthebeast subreddit suggests that the administrators want to promote this kind of modder/player interaction. However, discussion about

how to actually make mods, and especially technical details, make up a minority of the topics and are haphazardly mixed in with everything else.

r/feedthebeast is the most prominent venue for discussion of Minecraft mods on Reddit. It was originally a discussion board for topics relating to a collection of modpacks and their associated custom LAUNCHER program (see Glossary), called *Feed The Beast*[70] (FTB). Its focus has since expanded, as explained in their description:

> This sub-reddit was originally created for discussion around FTB launcher. It has later grown to be the main subreddit for all things related to modded Minecraft. Mod developers will be given a personal flair when confirmed.[71]

A more succinct description, displayed since the rollout of Reddit's new page layout in mid-2018, bills r/feedthebeast as "The official subreddit of Modded Minecraft." This actually refers to the relatively new FTB-affiliated community site ModdedMinecraft.net, but may as well be read as a general declaration that r/feedthebeast is *the* place to talk about Minecraft mods on Reddit. Capitalthree, who is not generally active in Reddit discussions, told me that he "posted a couple times on the FTB subreddit because I think it's just kind of a *standard way* to promote mods" (emphasis mine)—even those that bear no direct relation to FTB. Other subreddits with more accurate names exist, but lack the kind of buzz seen on the FTB subreddit. In October 2017, r/feedthebeast had approximately 40,000 subscribers, and by October 2018, that number had grown to over 60,000. Compare this to r/ModdingMC and r/MinecraftMod, both of which had around 1000 subscribers in October 2018—less than the number of *active users* seen in an October 21, 2018 snapshot[72] of r/feedthebeast. The description line for r/MinecraftMod even dryly notes that "/r/feedthebeast is active."

---

[70] *Feed The Beast* was originally distributed on the Minecraft Forums as a challenge map and associated pack of technology mods, but has evolved into a larger set of community websites, downloadable maps, and dozens of enormously popular modpacks curated by the "FTB Team."

[71] This description was captured on 8 October 2017. It was still available as of October 31, 2018 if using the "old" Reddit layout: http://web.archive.org/web/20181128055117/https://old.reddit.com/r/feedthebeast/

[72] https://web.archive.org/web/20181021175319/http://www.reddit.com/r/feedthebeast/

7.1.2 Real-time chat

7.1.2.1 Internet Relay Chat (IRC)

Briefly, IRC is a decades-old protocol for group and one-to-one text chats in real-time. Numerous public IRC *server networks* exist, each with some number of chat rooms or *channels*—most networks allow users to create and register their own channels. IRC messages are ephemeral and it is not possible to see what happened when one was logged out (unless one receives the information from someone else). However, it's quite common (especially in the channels mentioned below) for users to remain logged in around the clock, meaning they would have a record of everything that was said while they were away from the computer.

EsperNet appears to be the most popular network for discussion related to Minecraft and its mods—as well as gameplay and modding for Factorio, Kerbal Space Program, and several other games. Its status as the de facto IRC network for the Minecraft community almost certainly traces back to the fact that Notch himself established its #Minecraft[73] channel during early development.

The most immediately remarkable thing about the Minecraft-modding related channels on EsperNet is how little happens there. Of 18 publicly-listed channels, including #mcp (Minecraft Coder Pack), #Mystcraft, and #FTB , only two (#MinecraftForge and #BTM) featured actual *conversation* during over 50 hours of observation between September 2017 and March 2018. Despite having anywhere from 20 to 100 logged-in users, the other channels were almost completely silent. Occasionally a new user would appear and ask a question, but it would be greeted only with the IRC equivalent of cricket noises: the slow but inexorable scroll of join and quit messages (nor does joining the channel necessarily indicate intentional activity, thanks to the auto-rejoin functions supported by most IRC clients). The only activity seen on #buildcraft (an extremely popular industrial tech mod) was entirely piped over from the Discord bridge (see Subsection 7.1.2.3, below)—which, of course, points to why these channels are so dead. Discord appears to have

[73] IRC channel names are always prefixed with a pound sign ('#'). Discord also follows this tradition.

supplanted IRC as the chat platform of choice for these communities, and most channels now include a Discord link in their topic headers.

7.1.2.2 Discord

Discord is a synchronous text-based chat service that takes many of its cues from IRC, but incorporates new features—many of them apparently inspired by social practices on web-based message boards—that give it a wider appeal for modern users. While the presentation is synchronous and unthreaded, it is always possible to view a scroll-back of events that occurred while one was logged out—unlike IRC. User icons/avatars, roles/reputations, multimedia content, Facebook-style "reactions" to messages, voice and video chat, and private messages with three or more participants are also supported. Like IRC, the world of Discord is partitioned into "servers" with their own administrators and channel lists. However, with Discord, "server" refers to a social domain without implying anything about the form of network infrastructure. Any user may create their own "server" with a button-click in the Discord client, but none of the back-end computing will take place on the server-owner's machine: the true server, in the computational sense, is run out of Discord's data centres. Discord user accounts are global in scope (i.e. consistent across all Discord servers, rather than server-specific like IRC), as are private message capabilities (called "direct messages" or DMs).

Discord offers a more flexible way than IRC for communities to organize activity into separate channels. A single community, such as Minecraft Forge, will typically have a single channel on EsperNet IRC, but an entire "server" with multiple channels (ten in Forge's case) on Discord (see Figure 7-1, below). Channels can be created and removed by a server's administrator as needed.

7.1.2.3 Bridges and bots

Many of the online communities encountered in this research are spread across multiple platforms, due to the entrenched preferences of participants or a desire to increase outreach and

exposure by providing as many ways to connect as possible. In particular, Minecraft modding communities may be split between IRC users and those who prefer Discord. A chat bridge is a technical solution that allows communication across this divide, by monitoring both platforms and relaying each one's messages to the other. Minecraft communities often establish chat bridges between IRC and Discord by means of a bot—a computer-controlled dummy chatter that is logged into both platforms. In Figure 7-1, a channel called #ircbridge is visible on the Forge Discord server. Only messages within that channel will get passed to IRC, and vice-versa. While this does allow for cross-platform communication, it means that those who are only on IRC may be missing out on the main action in the other Discord channels.

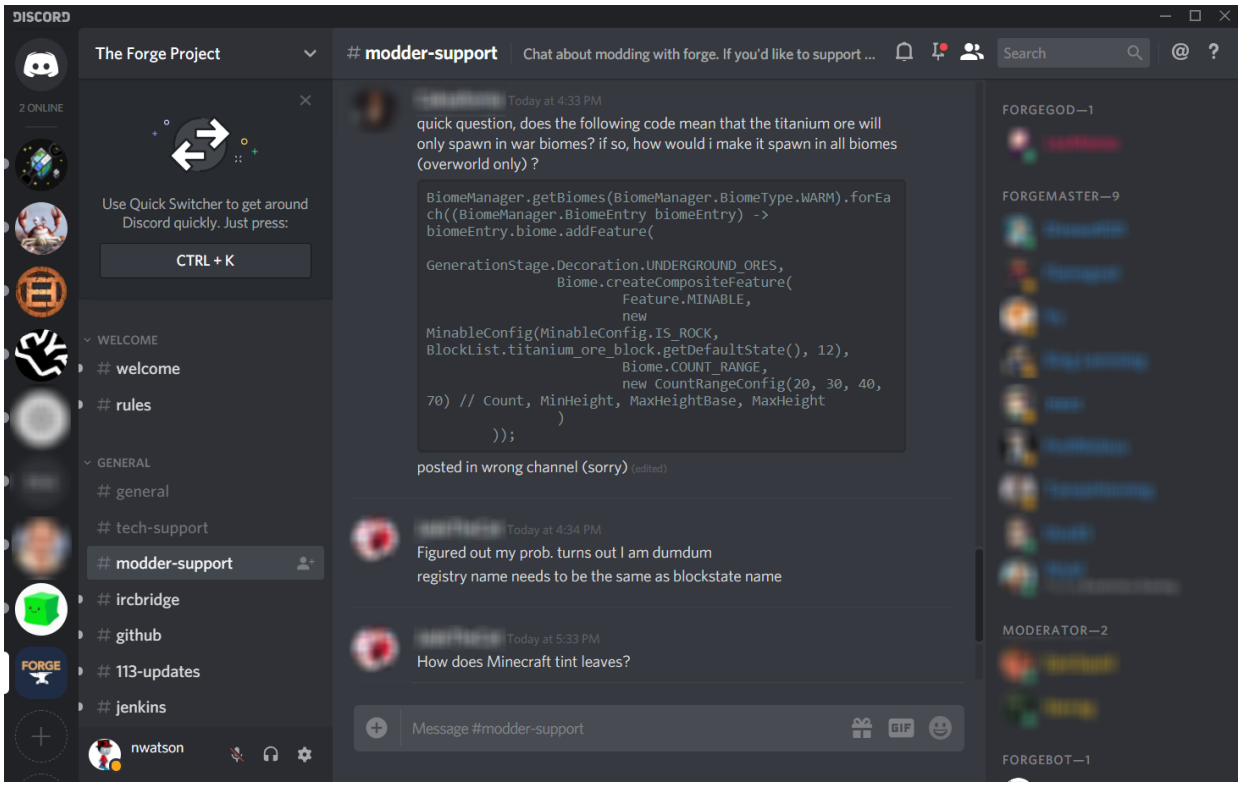


Figure 7-1. Screenshot of the #modder-support channel on Minecraft Forge Discord. The main chat window is in the middle, with the list of users present on the right. The left-hand pane shows the various channels hosted on the server, while the icon bar further to the left allows switching the view to another server. (Screenshot by N. Watson.)

7.1.3 Wikis

Fan wikis are a common and effective way for players to share knowledge about a game in an organized, centralized form. Networks like Fandom Wikis (also known as Wikia) host hundreds

of repositories of game lore running on MediaWiki software (the same platform used for Wikipedia itself). Operating in a monologic mode, they have become the de facto replacements for the printed strategy guides and magazine paratexts of past decades that are described by Consalvo (2007).

The *Minecraft Wiki* (https://minecraft.gamepedia.com, formerly https://minecraftwiki.net), was never a Fandom Wiki, but always served a similar function. It was founded in 2009, along with its affiliate MinecraftForum.net, by citricsquid. Both sites were acquired by Curse, an operator of a growing network of game fandom sites, in 2010. Curse (now a subsidiary of Twitch) later launched Gamepedia as a competitor to Fandom Wikis, and moved the Minecraft Wiki to the Gamepedia.com domain. The Wiki's front page reported on October 4, 2017 that it had 4,724 articles and 274 active contributors.

While the wiki articles focus on documenting and explaining game features, they offer technical information that is of incalculable value to modders—much of which is very likely written up by modders themselves. One of the key contributions of the Minecraft Wiki has been the documentation of *data values* for blocks and items, and all of their possible variant "states." Data values are the components of the inescapable operational logic of enumeration, whereby different types of objects and their different ways of being are assigned numbers to distinguish them from each other (see Chapter 6)—like how the ASCII format uses a numerical code to represent each text character in a file. Often, they are kept safely under the hood, used only for internal computational representation. But when data values are exposed in source code, they become what programmers call "magic numbers"—they mean something, but it is not obvious what, and the arbitrary nature of their representational assignment is apparent. The Minecraft Wiki provides documentation that demystifies data values so that, for example, if you want your code to test whether an in-game door is open or closed, you can find out from the Wiki that you need to check if the second binary digit of the metadata value associated with the bottom block of the door is a 1 (open) or 0 (closed). Since version 1.7, Minecraft has moved away from magic numbers, towards more self-explanatory text-

based data tags, but modders still need to consult a reference to find out the exact text they need to use.

Like Wikipedia, the Minecraft Wiki presents its articles in a monologic mode: they speak with a single, authoritative voice, imparting facts that do not seem to originate from anything except their own *fact-ness*. Of course, wiki articles are anything but monologic: they are an agglomeration of utterances from different interlocutors, all masquerading as a unified text, and all addressing the same imagined audience (at least, if people are following the style guides). In order to differentiate the utterances, we must compare different versions of the article across time. Alternatively, we could head to the article's "Talk" page to see a dialogue unmasked (though not simply a different version of the same one that takes place on the main page, of course).

I have written previously about the Minecraft Wiki article on "Transportation," and its associated Talk page (see Watson, 2017), noting that contributors leveraged two different methods of creating knowledge about all of the factors that affected a player's movement speed. On the one hand there was the empirical method, in which players built tracks and ramps out of various materials in-game, and tested them in a series of timed trials. On the other hand is what I have called the generative method, which appeals directly to what the source code actually does. A 2013 comment near the bottom of the Talk:Transportation page[74] purports to provide "the internal speed formula as calculated in the 1.6.2 client source decompiled with MPC" and "written in pseudocode":

```
speed = speed * [0.16277136 / (slipperiness * 0.91)^3]
If strafing & forward at same time
        strafe = strafe * (speed / sqrt(strafe^2 + forward^2));
        forward = forward * (speed / sqrt(strafe^2 + forward^2));
Else
        strafe = strafe * speed;
        forward = forward * speed;
motionX =  motionX + (strafe * cos(rotationYaw * pi / 180) - forward *
sin(rotationYaw * pi / 180));
motionZ =  motionZ + (forward * cos(rotationYaw * pi / 180) + strafe *
sin(rotationYaw * pi / 180));
```

[74] http://web.archive.org/web/20160416084242/http://minecraft.gamepedia.com/Talk:Transportation#Speed_formula

```
Apply speed to entity at this point
motionX = motionX * 0.54600006
motionZ =  motionZ * 0.54600006
```

Of course the pseudocode is still quite opaque to the uninitiated: at the very least, one needs some understanding of trigonometry; the typed-out math notation is messy (e.g. using "sqrt" instead of the square-root symbol); and there are still a few magic numbers (what's the 0.16277136 for?). The post continues with further explanation of the variables (e.g. "strafe is from -1 to 1, where 1 is when you hold the 'a' key, -1 is 'd' key, 0 is stopped"). This is a mobilization of modder knowledge. Although the commenter may or may not have released mods, they used a modder's tool (MCP) to engage in a fundamentally modderly activity—studying decompiled Minecraft code to discover how it works, and reporting those findings to a community online. As we will see throughout this chapter, studying how existing code does what it does is an important technique that modders use to develop their own code.

It is not just Talk pages either—many of the Wiki articles themselves rely on detailed information about Minecraft behaviour derived from studying decompiled Minecraft code in MCP. A tutorial on enchantment mechanics,[75] written to help players figure out how to maximize their chances of getting certain item enchantments, provides fourteen lines of pseudocode, with the ultimate authority cited as "Minecraft 1.8 source code." Twenty-four lines of pseudocode are used to explain a mechanic called "regional difficulty" by which monsters in an area become tougher the more time the player spends there.[76] Where players of other games rely on empirical observation and theorycrafting to glean knowledge about hidden game mechanics (Paul, 2011), the Minecraft community is in the unusual position of being able to benefit from access to, and knowledge of, the game's source code—thanks to its Java implementation and its entirely-homebrewed modding system. There also appears to be an expectation on the part of Wiki contributors that players are able to understand the pseudocode, and are comfortable reading it. At any rate, claiming that one's

[75] http://web.archive.org/web/20181231173103/https://minecraft.gamepedia.com/Tutorials/Enchantment_mechanics
[76] http://web.archive.org/web/20170731023511/https://minecraft.gamepedia.com/Difficulty

assertions about how Minecraft works are backed by references to the supreme authority of the code itself insulates them against challenges from all but those people who are also conversant in computer programming.

To summarize, this is an example of [1] a modding utility (MCP) being [2] leveraged in paratexts that are *not* primarily about modding (Minecraft Wiki articles about game mechanics), in a way that [3] displays embodied gaming capital (expertise to understand code; ability to speak authoritatively about Minecraft's mechanics) and [4] constructs a social hierarchy between the creators/purveyors of knowledge (modders, code-readers, Wiki authors) and consumers/readers.

7.1.4 CurseForge

CurseForge[77] is a Curse-operated website that hosts modding projects for a range of games, although most of its traffic is specific to Minecraft—as of December 28, 2018, the site hosted 42,666 Minecraft mods, dwarfing the next biggest category, World of Warcraft with its 6,622 addons.[78] CurseForge is tiny compared to the giant, independent mod distribution platform that is Nexus Mods (compare 35 games on CurseForge to 665 on Nexus; or CurseForge's 203 *Skyrim: Special Edition* mods to Nexus's 16,800) but it retains relevance by having games that Nexus does not, Minecraft being chief among them.

For Minecraft modders and mod-users, CurseForge is a convenient central distribution platform maintaining a searchable database of mod "project pages" (Figure 7-2, below). Each project page can host a set of downloadable mod files for various Minecraft versions, and typically also provides a Wiki-like space for mod documentation, a comments and feedback section, and a utility for tracking dependencies. Open-source mod projects also often include links on their CurseForge project pages to their GitHub source code repositories and bug report pages. The two sites

[77] CurseForge is not related to or affiliated with Minecraft Forge. Given that the CurseForge.com domain name was first registered in 2007 (see https://www.whois.com/whois/curseforge.com), there is likely no relation between the names. Rather, the name was perhaps an homage to the open-source project hosting site SourceForge.
[78] http://web.archive.org/web/20181226103531/https://www.curseforge.com/

complement each other: GitHub facilitates online modder collaboration and version control, but is not optimized as a player-facing distribution site; CurseForge's project pages are primarily for documenting and hosting mods for players to find, not for actually working on the mods.

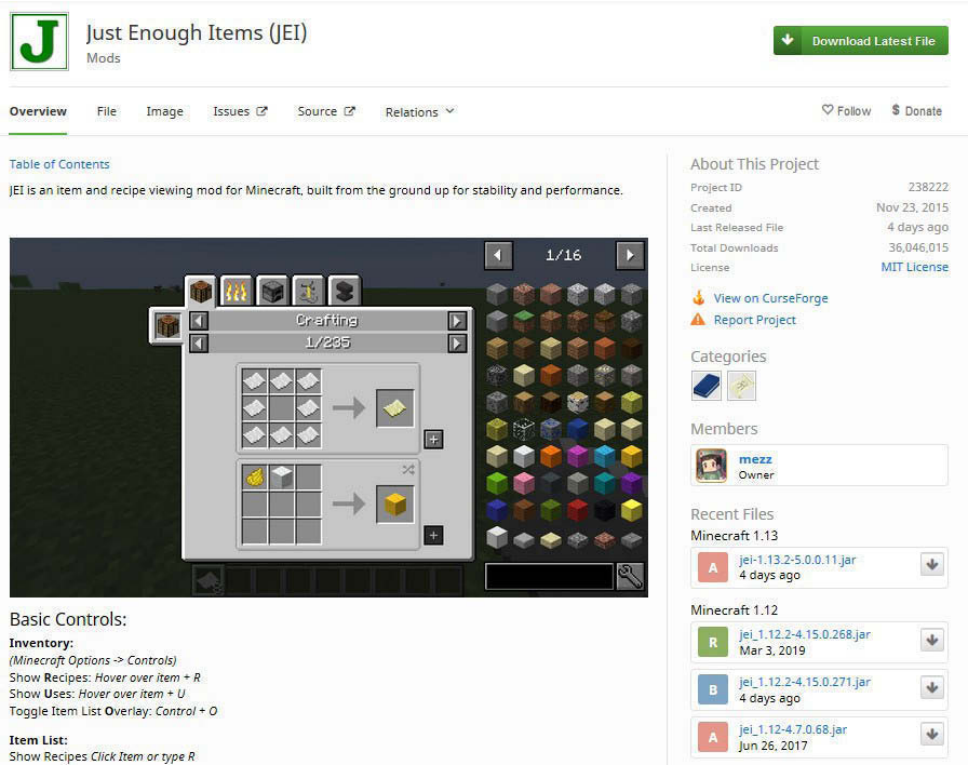


Figure 7-2. Example of a CurseForge project page, featuring documentation for players and downloadable files going back several versions. This project page is for Mezz's Just Enough Items mod (described in Subsection 6.4.1). https://minecraft.curseforge.com/projects/jei (Screenshot by N. Watson.)

Before CurseForge, mods were typically promoted and documented with MinecraftForum posts (see Subsection 7.1.6, below), and the files themselves were provided on a variety of generic hosting platforms scattered across the web.

CurseForge is part of the trend of increasing centralization in mod distribution (see Yang, 2012). It is an explicitly strategic effort to impose a consistent, formal structure on the circulation of mods (naturally, one that Curse can monetize), and as such, it certainly makes participation in the mod-world easier to navigate for players and to some extent also for modders. However, several of the modders I spoke to were distrustful of CurseForge. Copygirl, who generally takes the position that it is important to "keep the web decentralized," is unsparing in her criticism: "They bought up

MCF [MinecraftForum], pretty much became the monopoly of mod(pack) distribution, and now sold out to Twitch. They profit from people's free and hobby work."

Capitalthree comments:

> I think the main thing that a lot of people don't like is the fact that CurseForge has a monopoly. And, well, maybe there's more to it than that. It's owned by Twitch . . . which has a pretty shitty reputation for a lot of reasons. . . . I don't want there to be a situation where Curse is able to, for example, release a launcher with modpacks that only allow loading mods from Curse, which—there's definitely rumblings that they would like to do that if they could. Trent in particular has fought against that by not letting some of his mods be on Curse, like fontcap and Oplint. In particular Oplint, which is so essential for any modern modpack. It's one of the big bastions against being able to do a modpack purely based on Curse, and that's a good thing.

For those suspicious of Curse's motives, its strategic apparatus is seen not as a public utility for the benefit of the community like those discussed in Section 6.4, but as a cynical attempt to co-opt, control, and profit from fan playbour. The resistance is, of course, tactical, with key modders like Trent seizing the opportunity to intervene by withholding consent and cooperation. Trent's action is both a discursive statement of principle and a direct structural intervention in the circulation of cultural artifacts.

7.1.5 Other venues and honourable mentions

There are many other online spaces that make up the modder landscape, and there is not enough time to catalogue—much less analyze—each one. Still, a brief acknowledgment of the roles played by some other sites is in order.

Documentation and tutorials: Modders, especially novices, rely on these to become conversant in modding methods. In early days, a majority of the comprehensive documentation for using ModLoader and Forge was found in the "Tutorial" sections of the forums described above. Such forum posts, in addition to personal blogs and YouTube video tutorials, retain their importance, but

more recently official, authoritative, monologic-presenting documentation has appeared, such as Forge's official documentation at https://mcforge.readthedocs.io.

Twitter: Modders will often announce new updates on their Twitter feeds, but also often use Twitter for non-Minecraft-related discussion. BTM Moon was first publicly announced, then cancelled, then subsequently re-announced on Twitter during October 2017. My study did not seek to capture inter-modder communications over Twitter, but there may be rich material there for future research efforts.

YouTube and Twitch: These video sites have undeniably played a major role in popularizing Minecraft as well as its mods (MacCallum-Stewart, 2014; Brackin, 2014). YouTuber direwolf20 is known for his video demos and reviews of mods, and has accrued enough cultural capital to be regarded as a curatorial maven whose opinion holds weight—major fan websites offer downloadable direwolf20-branded modpacks so that players can follow along and experience exactly the same mod configurations that he is using. Being featured on direwolf20's channel is also a major breakthrough for any aspiring modder. Although this is not an example of modders talking to each other, it is a means by which modders receive messages about the reception of their outputs. Tothor highlights the significance of these messages:

> One of the most rewarding things is actually seeing people play with your mods. Like, you'll go on YouTube and see a YouTuber with ten million subscribers playing with your mod. They have no idea who I am, but I made that. That's pretty cool. . . . I was on a plane . . . and I saw someone, I think they were watching a YouTube video of someone playing with my mod. That person on the plane is watching someone playing with my mod. And that was pretty cool.

YouTube and Twitch also act as additional channels for the distribution of instructional knowledge about modding in the form of video tutorials. Some modders, like Amadornes (BTM Moon's official streamer/documentarian), actually conduct live Twitch streams of themselves coding mods, with the feed displaying their IDE window as they type.

7.1.6 Usage patterns

The communications venues described were not used equally by all modders I interviewed. Of the nine interviewees who were modders, almost all reported hanging out in IRC or Discord spaces, but only two were active on MinecraftForge forums. Two more had previously been active on MinecraftForum.net, one of whom recalls having been one of many unofficial helpers in the modding community: "I earned a reputation for helping out with modding questions, and for hating MCreator."

According to capitalthree, it was "pretty standard" before CurseForge for modders to make threads dedicated to their mods on MinecraftForum.net—and many still do. Although the forum site does not directly host the downloadable files, these threads contain download links as well as basic documentation and introductory materials. Other players can reply with feedback, questions, or bug reports on the same thread—for some of the oldest mod threads, those that have been around since 2010 or 2011, it is not uncommon for the replies to number in the tens of thousands. When the mod is updated, the author can edit the original post directly to provide new information. Being active in one's own mod thread does not necessarily mean that one is active in the rest of the forums, however.

CurseForge reduced the need for modders to have active MinecraftForum.net threads, not so much because it could host files, but because it provided a place for modders to post promotional and instructional materials, documentation, and update news to accompany those files—the very paratexts for which the forum threads were originally used.

Capitalthree points to one reason why some modders have given up on the forums—due to their social architecture, they may be of questionable utility to many:

> Nobody ever really paid attention to my MinecraftForum threads anyway—I think because of the way bumping works in forums and stuff. It just overwhelmingly continues to privilege the things that are already popular. It's like a snowballing system for popularity.

For ScriveShark, the choice of which social venues to monitor is linked to the emotional well-being of one's modding life, emphasizing the importance of *social* capital:

> It's a lot of time and energy to go through more than one or two [forums], and the communities vary in terms of expectations and interactions. I'm perfectly happy in the Discord one, now, as people can chat and hang out and are generally positive. As a modder, you have to stay in touch with the people who like your work; you'll hear enough from the ones who don't.

Having surveyed the major online venues for modder discourse, I now turn to an exploration of how those discourses themselves function.

7.2 Software infrastructure as discourse

To make the case that software itself embodies discourse and endorses certain value systems over others, one need look no further than the internal structure of the Forge Mod Development Kit (MDK), and how it has changed over the years. The Forge MDK is a downloadable package, backed by bundled MCP tools, that provides all the necessary components of a development environment for a Forge modder, including decompiled and modified Minecraft source code, and both source code and binaries for the Forge API's additional modules (i.e. additional class files that are entirely written by the Forge team, as opposed to Minecraft base classes that contain modifications). The MDK also includes tools of convenience to automate the task of setting up an Eclipse or IntelliJ workspace. Different versions of the MDK target different versions of Minecraft.

If one were to use MCP alone to decompile and deobfuscate Minecraft, one would have a collection of source code files that could be directly edited in an IDE or even just a text editor, recompiled, and repackaged for mod distribution. This is exactly how jar modding was done in the early days. While Forge was intended to eliminate jar modding, it was still possible to directly edit base class source code, even when using Forge, up to Minecraft version 1.7.10 (June 2014). This is because Forge was not so much distributing a comprehensive MDK as they were simply providing scripts to patch Forge modifications into MCP-decompiled Minecraft base classes, along with the

source code for their supplemental modules (.java files). As a consequence, it was possible for Forge modders to make their own local changes to Minecraft base classes and even to Forge's internal components—and it was often tempting to do so, since accomplishing certain simple goals can take many more lines of code, and involve significantly more overhead effort, when doing it the "right way" with Forge. Consider the example use case for the Forge event model, described in Appendix C: trying to prevent wolves from being hurt when they touch cacti. One could go the "proper" route, making an entire self-contained mod with an event handler and a bunch of annotations and metadata, which would take at least a few dozen lines of code. On the other hand, a single line of code inserted in the right spot in a Minecraft base class could accomplish the same objective. Of course, the resulting mod would have all of the compatibility and legal issues associated with jar modding (as described in Subsection 4.2.1).

Starting with Minecraft version 1.8, source code is "attached" to Minecraft and Forge .class files, but they all remain bundled in a JAR file, and can only be read—not written. This allows modders to study Minecraft/Forge code, learn how it works, and discover hooks that they can use in their own mods, but prevents them from actually making jar mods or changing how Forge works. In a standard reading, this appears not to be discourse at all—just a sort of technological safeguard. This is, of course, precisely the power of software-architecture-as-discourse: it is able to embody a monologic position that hides the social processes of its production. Under a dialogic analysis, it turns out that this aspect of the Forge MDK is attached to utterances on the Forge message boards, such as this December 2015 exchange between a user and one of the Forge developers, from a thread called "[TUTORIAL] Getting Started with ForgeGradle" (due to the user not being a native speaker of English and the subject matter being somewhat technical, I have rewritten their posts for clarity, based on context of the thread and my domain knowledge):

User: *Hi, can someone help me? I did all of the steps from the originally-posted tutorial twice, but every time I open Eclipse, I can't find the .java archives. In my project explorer I found only .classes*
[Screenshot from Eclipse]

Forge dev: `If you mean "java archive", which literally means ".jar file", then I do not know where your problem is. Your screenshot is full of them.`
`If you mean ".java file" (which is not an archive, it's just a text file), then I also do not see your problem. You can either navigate through the jar files I already mentioned above and see what they contain or you can use eclipse's "go to type" feature where it will open any class you tell it to (press Ctrl-Shift-T).`

User: *I didn't explain it well. Inside the .jar file there are no .java files to edit. [Screenshot from Eclipse] How can I edit entities [modules that correspond to Minecraft's creatures, vehicles, and other moving objects]? I cannot edit .class files. Sorry for any mistakes. I'm a beginner, I don't know much.*

Forge dev: `Yes, of course you cannot edit them. They are part of libraries that Minecraft uses . . . part of Minecraft itself. You cannot just edit other people's code.`

User: *And what do I do to edit (create) mods?*

Forge dev: `You create a mod by writing code in src/main/java. If you want to alter the behavior of existing things, you can use a variety of techniques, e.g. events, reflection, etc. It depends on the use-case.`

User: *Then why have I seen people editing it? In tutorials, all of these .classes are .javas, and they edit them.*

Forge dev: `Then you probably saw old tutorials. Editing the jar files has been highly discouraged since forever`

Note the phrasing: "of course" the files can't be edited—the decision to restructure the Forge MDK to prevent such edits has been reframed as simply an aspect of the natural order of things. Forge's design operates from within its "proper" place to ensure that modding practice is also proper. The issue isn't even explained in terms of the legal and logistical pitfalls of jar modding. Instead, it is linked to the value judgment about authorship and interference that lies in the statement, "You cannot just edit other people's code."

7.3 Dissemination and co-regulation of modder expertise

That Minecraft and Forge source code remains *readable* in the MDK is significant. This is because, even after more than eight years of modding, there is still no top-down, strategically-developed pedagogy, no canonical book that explains how to do it. O'Reilly, a major publisher of

tutorial and reference books for various computer applications and programming languages, actually did publish a *Minecraft Modding with Forge* book in 2015 (Gupta & Gupta, 2015). It has never been updated (as of 2018) and is effectively useless for Minecraft versions newer *or* older than 1.8. The constant changes to Minecraft's code base and Forge's structure tend to frustrate attempts to solidify a single, sanctioned set of instructions. To return to de Certeau's Manhattan metaphor, there is no skyscraper upon which to witness the city in its totality, no map that circumscribes the territory. At best, there are scraps of documentation, tutorials written on forums and personal blogs, video explanations on YouTube, with obsolete directives mixed in with the up-to-date—like scattered pages from memo pads, with scrawled step-by-step driving directions from wherever the writer was to wherever they happened to want to go that day. Some of the tutorials are very helpful, but broadly speaking, there is no "by the book" way to do things, because there is no book. The effort to disseminate knowledge about how to make Forge mods is decentralized and somewhat chaotic, with plenty of contradictions and obsolete instructions to go around (recall how the Forge developer in the quote above blamed "old tutorials" for spreading the idea that modders could/should edit base classes).

The typical way to learn Forge modding is by immersion and experience: explore the read-only source code in the MDK to figure out Minecraft's data structures and operational logics, and to find hooks that might be helpful; follow a tutorial to the letter to get a basic mod up and running, and then start messing around; study code from open-source mods to see how other modders have approached similar problems; and of course, when you get an error that you just can't figure out, post on a forum or pop into a Discord channel for some quick answers from the sorts of modders that like to make themselves available as helpers (but don't expect them to spell it all out for you). This is how expertise are disseminated throughout the community, how particular forms of embodied gaming capital pertaining to the ability to make, talk about, help with, and critique mods are forged. It is a tactical process, a bazaar of cultural exchanges.

An exchange between 'R' and 'F' from r/feedthebeast demonstrates the normative expectation that aspiring modders will learn by studying existing code.

- F says: "I want to create a mod that adds some cool new blocks and therefore I need to add a custom 3D model… but how can I do this?"
- R replies by offering a link to his own code on GitHub, telling F to look in a specific file called "blockstates" therein[79] for a guiding example.
- R continues: "I also recommend the official Forge repository. Debug classes have many examples. If you need any explanations message me, [we] will talk it over."
- F comes back with an error message. "Do you know how I can fix this?"
- R: "You did not look in blockstates like I said. It said it in plain text. . . . Look and read the code, look where it is pointing. Also your obj[80] is in the wrong format. Look at any of mine."
- F: "But how can I make my Objs the correct format?"
- R: "Open any of my obj files with a text editor (Notepad++ if you're on Windows) and spot the difference. Look in the Forge debug ones, and do that too. Like I said, I have a script for Linux. If you are on Windows, write your own."

Rather than giving line-by-line instructions on how F needs to fix their files, R repeatedly tries to redirect F to existing mod projects, which provide the answers "in plain text." As for the "script for Linux," R is referring to a tool they made that automatically rewrites a particular kind of data file (the OBJ) in such a way that Forge will be able to read it. There are two possible ways to read, "If you are on Windows, write your own." R may be signaling an expectation that modders would have the requisite skills to write a script, but another possible reading is that R is implying that Windows users have cut themselves off from useful tools by their own choice, and therefore have to accept that sometimes they are on their own. Either way, this is an implicit statement about what tools "proper" or serious Minecraft modders should use and what they should be capable of.

For all that expert modders want novices to learn from examples as much as possible (R is far from alone in this), this approach does contain hazards for helpers. When 'J' posted to the Forge forums asking for help with an error, and provided code that they had developed based on one of the

[79] This is a blockstates JSON file, an example of which can be found in Appendix B.
[80] A Wavefront OBJ is a type of data file commonly used for representing 3D graphical models.

many modding tutorials out there, they were informed by Forge expert 'D' that they were doing it wrong.

J countered: "If so, why do so many tutorials have you [do it that way]?"[81]

And D replied:

> Because no community I have seen so far tops Minecraft Modding in the amount of cargo culting that is happening. Someone, somewhere, started this. And everyone blindly copies it without ever questioning what they are doing or why they are doing it. And if they try to do so, they fail, because they pretty much completely lack any kind of practical programming knowledge.

"Cargo-culting," or cargo-cult programming, refers to programmers mimicking example code that they saw somewhere because it seems to work, without understanding what it really does or why they would use it. The term alludes to a popular narrative that following the Second World War, Melanesian islanders developed rituals mimicking the activities of Japanese and American military forces that had established outposts and deployed cargo on the islands during the war. These rituals (supposedly) included clearing swathes of land for dirt airstrips, and wearing mock radio headsets made from carved wood, in hopes that the gods would be appeased and reward the islanders with boons of cargo delivered from the sky. What this has in common with cargo-cult programming is the idea that the uninitiated are emulating the superficial form of a practice they do not understand, in a way that is unlikely (or less likely) to produce the anticipated results.

Notwithstanding the extent to which the popular cargo cult narrative does or does not correspond to a historical reality, the expansion of the term to other domains is inevitably entangled in colonialist value judgments.[82] The (temporary) American colonists are figured as the originators of

---

[81] The original text is, "why do so many tutorials have you make a CommonProxy class?" The example code in Appendix B provides a cursory explanation of what this is. Since I use it in my example code, I suppose I too must've been reading the "wrong" tutorials, and/or am guilty of cargo-cult programming myself!

[82] It would not be entirely fair to suggest that the term "cargo-cult programming" is necessarily used maliciously. The term does not hinge on the actual details or veracity of the original cargo-cult concept. Rather, it functions like a literary allusion, similar (for western audiences) to expressions like "thirty pieces of silver" or "tilting at windmills," in that it facilitates the communication of a concept through the meanings

something of value (cargo; manufactured goods), and participants in a complex sociotechnological system. The islanders, with their "superficial" culture, are unable to perceive the "depth" of the American practice, and thus fail to understand that mimesis will not bring cargo. It is a fantasy-parable of a victory of strategy over tactics, in which the islanders' attempts to seize opportunities on the wing, to appropriate a piece of the West's prosperity for themselves, are not a form of threatening resistance or self-determination, but merely an amusing and quaint failure.

Numerous scholars have challenged the veracity of this popular interpretation of the Melanesian encounter, which Lindstrom (1993) calls "cargoism." Otto and others have argued that the islanders' practices are less about a literal expectation of receiving material goods than they are about the process of reconfiguring social relationships and forging new meanings in the wake of a complete disruption of the prior social order (Otto, 2009), in which case it is ironically the cargoist narrative that falls into the trap of superficial thinking, failing to perceive the social realities underpinning the ritual practices.

It is perhaps fitting, given Minecraft's frontier motif and my ambivalent interpretation of Minecraft modders as settlers, that one of the chief hurdles in the way of strategic modding is named after the narrative of the cargo cult. D's utterance implies that cargo-cult programming *is* seen as a threat to efforts to build a rational, strategic order for Minecraft modding. However, as with the cargoist narrative, there is more to the appropriation of programming formulae by novice modders than the expectation that mimicking an incantation will produce rewards. In the absence of a truly monologic pedagogy of mod-making, novices have little choice but to engage in a tactical play that entails the opportunistic experimentation, reconfiguration, and redirection of source-code units—and by extension, their underlying operational logics—towards novel purposes. In any case, this is precisely what is encouraged by many modders, like R, who are operating from within the "institution" of modding.

---

condensed in the literary referent. That said, it would also not be entirely fair to insist that usage of the term can be entirely, innocently isolated from the troubling colonialist connotations of its antecedent.

The proper way to make mods is one arena in which authority is contested, but we also see a performance of gaming capital in the way that contested notions of the proper way to *play* are tied up in claims to mod ownership and the authority to dictate terms of consumption. It is to these issues that I now turn.

7.4 Ownership, authorship, and authority

FlowerChild, a participant on the Minecraft Forum and early contributor to Forge, thought wolves were a bad idea.

Wolves first appeared in Beta 1.4 (March 31, 2011) as neutral mobs a player could encounter in the wilderness. They could be tamed by way of a gift of bones, after which they would wear a collar, follow the player, attack hostile mobs, and sit on command. FlowerChild was quick to criticize the new feature on the forums:

> I think an idea has been suggested that is totally secondary to the core gameplay of Minecraft, which to my mind revolves around exploration and construction. Wolves have nothing to do with either of these. They are entirely secondary to the core gameplay experience. . . . In other words, I believe that in adding Wolves to the game, you've put yourself on a development slippery slope. You've basically committed yourself to spending a huge amount of development time on a feature that is entirely secondary to your core gameplay experience, and features that are unlikely to be reusable in the rest of the game. [83]

Within 24 hours, the post had garnered 263 replies worth of debate. FlowerChild maintained that he had no animosity towards wolves *per se*, but thought development resources would be better spent on features that "have far more relevance to the gameplay experience," such as new blocks. When a commenter argued that thinking of good ideas for new blocks was nontrivial, and wolves could inspire such ideas, FlowerChild countered, "I think practically anyone on these forums could throw out a half-dozen blocks they'd like to see in the game off the top of their head."

[83] http://web.archive.org/web/20171231230517/http://www.minecraftforum.net/forums/minecraft-java-edition/survival-mode/220651-notch-wolves-are-a-bad-idea

To prove the point, he started a new thread in the "Suggestions" sub-forum, challenging contributors to brainstorm a list of six new blocks they would like to see that "would contribute more to the game than wolves" and require less development effort to implement. His own ideas and several of the replies became the basis for his new mod *Better Than Wolves* (BTW), billed as a suite of features that would have been a better use of Mojang's time—from windmills and cement buckets to pottery-making and leather tanning. As for the wolves themselves, FlowerChild reimagined them as functional blocks which, if fed periodically and kept in a dark enclosure, will periodically produce dung that can be used for tanning leather.

BTW is an exemplar of gamers expecting developers to listen and enter into dialogue with them (Banks, 2013, p. 23), and of gamers deciding they can "do one better" (Joubert, 2009) than the developer's offering. It is exactly the sort of thing that Yang (2012) refers to when he writes of "mods making critical arguments." However, FlowerChild's brainstorming challenge also echoes Notch's active solicitation of community input on block design in Minecraft's early (2009) development (Redmond, 2014, p. 12-13). Ultimately, Mojang's creative direction and FlowerChild's sense of what makes for good game design were increasingly divergent, such that by 2013, he had taken to disabling a substantial number of vanilla features and classifying BTW as a "total conversion":

> I've ripped out any [vanilla] features with sub-standard design that I wouldn't have personally considered of high enough quality to go into the mod. . . . 1.5 was an absolutely horrible release, the worst yet IMO, and I'm done scrambling to try and correct Mojang's increasingly poor design work. . . . Vanilla is going one way, and I'm no longer willing to follow it, so I'm going another. [84]

FlowerChild saw BTW as an effort at "turning MC into an actual coherent game instead of a loosely connected set of toys." His opinions and design decisions attracted a mixture of enthusiastic praise and vehement disagreement on the forums, but perhaps its most controversial aspect is that, at

[84] http://web.archive.org/web/20190401191652/https://www.minecraftforum.net/forums/mapping-and-modding-java-edition/minecraft-mods/1272992-better-than-wolves-and-and-and-total-conversion-v4?page=1343

a time when the project of mod interoperability was proceeding apace, BTW was incompatible with everything.

7.4.1 Two types of incompatibility

Generally, there are two ways that mods can be mutually incompatible. In *procedural incompatibility*, bytecode from both mods simply cannot co-exist in the same runtime. It could be because both are jar mods that replace the same class file, such that only one mod's changes actually get loaded. It could be that one mod's code relies on assumptions about Minecraft functionality—"contracts" in programmer-speak—that the other mod alters and invalidates. Procedural incompatibility often means the program will crash, or else exhibit incoherent and unintended behaviours. Block ID collisions (multiple mods trying to claim the same ID numbers for their blocks) were a major source of procedural incompatibility before Minecraft 1.7.2 and its corresponding Forge release (as discussed in Subsection 6.3.3).

If procedural incompatibility arises from a violation of the contract between codebase and mod, then *mechanical incompatibility* is a violation of the contract between game and player. The program runs just fine, but play itself is incoherent or prone to "game-breaking" troubles. For instance, one mod might add nuclear power plants to the game, along with a new type of ore called pitchblende (a uranium source); another mod that changes how vanilla ores are distributed underground, in overriding the game's usual behaviour, might inadvertently prevent any pitchblende from appearing at all. When both mods are played together, the nuclear power mod might work just fine in Creative Mode, wherein the player can conjure up as much pitchblende as they like, but in survival play, there would be no way to acquire fuel for the reactors! (A mod called CRAFTTWEAKER, described in Subsection 7.4.3 below, embodies one possible strategy for alleviating such incompatibilities.)

Total conversion mods are quite likely to have mechanical incompatibilities with just about any other gameplay mod because, by definition, they radically alter the underlying structure of the

game. However, they need not be procedurally incompatible. Card1null's *Biolithic* mod, which offers a kind of realist's stone-age survival experience, adding falling blocks,[85] ambient temperature, item weight, butchery, complex nutritional needs, and more, does not play nicely with other gameplay mods. Card1null told me that this incompatibility was one of the main complaints (out of a largely positive reception) about Biolithic, but that it was something he and his co-designer/father "already knew and made our peace with"—framing interoperability as an ideal to be considered, but one that could be sacrificed if necessary: "We've thought of Biolithic more like a modpack all of its own. It changes many fundamental aspects of the game," the duo explain on the mod's CurseForge page. Even so, they intentionally worked in support for Mezz's Just Enough Items recipe manager—a valuable asset to players in a mod that throws out all of the familiar vanilla recipes. Since JEI only affects the user interface, and since both Biolithic and JEI are Forge-based, the two mods could be integrated without either type of incompatibility.

BTW exhibits both kinds of incompatibility: in addition to its mechanics colliding with the assumptions of just about every other gameplay mod, its code cannot be run alongside Forge, nor (therefore) with any Forge-based mod.

7.4.2 Better Than Wolves and "intentional" incompatibility

BTW's evolution into a total conversion is as much about the software platform as it is about game mechanics. It was once one of the champions of Forge's interoperability mission, and FlowerChild was a direct contributor to the API in the early days. However, by the end of 2011 he had quit the team, and began to transition BTW away from Forge, making it a more typical jar mod. In a YouTube interview[86] with Battosay, a maker of Minecraft "Let's Play" videos, FlowerChild explained that he had become uncomfortable with how the API had expanded into an "invasive"

[85] In Minecraft, sand and gravel blocks will fall if they have empty space below them. Other blocks, however, will stay put even if they are hanging in mid-air with no supports. Biolithic makes these "solid" blocks subject to gravity too under certain conditions, forcing the player to think about structural support within their cave-home.

[86] https://www.youtube.com/watch?v=B6SN4QQ_WSc

monstrosity that was increasingly difficult to maintain, and took ever-longer to load in a Minecraft game:

> As far as I was concerned, we [Forge devs] should basically be doing the same things that we were doing as individual modders, trying to minimize the number of base classes that we were modding, trying to keep the API small while providing the functionality that was desired by everybody. But basically doing so in moderation, right?

Particularly troubling to FlowerChild was a decision that required changing thirteen base (vanilla) class files for the sake of one feature in the Redpower mod. Complicating matters was what FlowerChild referred to as a "personality conflict" with Redpower's creator, who was also a Forge team member. BTW's reliance on Forge came to feel like a liability:

> I saw that the [Forge] codebase was getting larger and larger. I didn't even want to bother trying to understand everything that was going into the API. It was slowing down in its update cycle. . . . It began to annoy me that my mod was more and more dependent on what Forge was doing, and meanwhile it was going down a path that made me uncomfortable. . . . The mod was becoming reliant on somebody that I just didn't get along with.

By this time, FlowerChild had already begun to advocate for using BTW as a "standalone" mod because he felt that—due largely to its "immersive" difficulty—its use alongside other mods suppressed some of its best features:

> Practically every tech mod out there gives you more powerful ways of doing things. So what ends up happening is there's very little reason to use the stuff that's in Better Than Wolves, and . . . people tend to use it for decorative stuff, like the windmills or the water wheels or whatever, and that really isn't the intention of the mod. There's a lot of really interesting stuff, a lot of interesting systems within Better Than Wolves that don't end up getting used as a result because they're being played alongside other mods.

From there, it wasn't a big leap to a total-conversion jar mod. Why worry about code compatibility if the mod is supposed to stand alone? Why worry about conflicting gameplay features when the mod can't be run with anything else anyway? This is a positive feedback loop, in which consciously *choosing* either type of incompatibility (mechanical or procedural) provides rhetorical

cover for acquiescing to the other. However, with BTW having been a popular mod that players were often wanting to use together with other popular mods, some players and modders put extra emphasis on the "intentional" aspect of its incompatibility. For example, a compatibility page on the wiki for the *Millenaire* mod,[87] after listing numerous mods that are compatible out of the box or with minor fixes, lists Better Than Wolves as "intentionally incompatible with Forge." It is not clear whether such phrasing is intended to insinuate a more spiteful motivation behind BTW's incompatibility, or if it simply saves players the trouble of asking (and modders the trouble of answering) questions about whether compatibility will be added in the future.

7.4.3 Terms of play

Better Than Wolves endorses a specific understanding of what good game design and good game play mean. It uses the force of its license terms—a strategy based on a legal institution—to enforce the purity of that vision. The copyright notice accompanying version 4.B0000001 indicates that the package "may not be reproduced under any circumstances except for personal, private use *as long as it remains in its unaltered, unedited form*" (emphasis added). Furthermore, "Decompilation and reverse engineering is strictly prohibited without advanced written permission." Finally, FlowerChild will "NOT grant redistribution rights to this mod to anyone that asks, whether it be for a mod-pack or whatever." This effectively introduces a third type of incompatibility based on legal restrictions. The fact that reverse-engineering and alterations are prohibited is significant because it prevents anyone from patching the mod to make it compatible with others. Although BTW development depends on being able to reverse-engineer and alter Minecraft code, and on Notch's techno-libertarian belief that people should be free to do what they like with their own copies of software, its own design is both technologically sealed and legalistically secured. Mojang, for its part, has benefited immensely from modders' extension of its product, but as BTW is a small, non-

[87] http://web.archive.org/web/20170724225746/https://millenaire.org/wiki/Help:Compatibility/1.4.5

commercial offering, the same business incentive does not exist for FlowerChild to allow modding of his mod.

Some modders are strongly critical of others using closed architectures and restrictive licenses, however. One interviewee remarked:

> If you show up to someone's . . . proprietary game and you decompile it and then you make a mod for it, and then you tell other modders not to decompile your mod . . . why would they listen to you if the whole community is already based on decompiling other people's work without permission? It's very silly.

There are many modders who seek to exercise some kind of control over the terms under which their mods are used, with varying degrees of permissiveness. "Reika" makes a suite of interrelated tech and magic mods that, while closed-source, are Forge compatible. He specifically allows any kind of decompilation and modification for strictly personal use, and conditionally allows distribution and inclusion in modpacks subject to certain restrictions,[88] including a prohibition on redistributing recompiled versions of modified source code or using binary patches (ASM or bytecode hacks). This much is not especially controversial, but some modpack curators take issue with an aspect of Reika's policieson public packs, which "must not use external mods to significantly change the way" Reika's mods work, such that (for instance) the tech trees "are radically altered" or "features are missing."[89]

This becomes an issue because there are ways to change how a Forge mod works without actually altering the mod's own files. Modpack curators commonly use a mod called CraftTweaker (or its predecessor, MINETWEAKER), a strategic solution for banishing mechanical incompatibilities between mods and establishing some approximation of the ever-sought-after Platonic ideal of "gameplay balance"—or sometimes just for simplifying and streamlining by, for example, reducing

[88] Reika's rules for "semi-private" modpacks are more permissive than those for publicly-available packs, but restrictions are still present. In principle, a group of friends passing a custom modpack or altered Reika mod could be considered a "semi-private" use.
[89] http://web.archive.org/web/20171227031133/https://sites.google.com/site/reikasminecraft/pack-permissions-and-rules

the number of different item-types called "copper wire" from four non-interchangeable objects to one jack-of-all-mods. This can all be accomplished by writing simple directives in a text file, so there is no need for a modpack creator to do actual programming either. A typical operation for CraftTweaker is the removal of extraneous crafting recipes or the specification of different ingredients. Mods that add new crafting recipes do so by submitting them to Forge's *recipe registry*, so CraftTweaker can change them by altering the contents of the registry afterwards, without touching the mods themselves.

Reika does not permit the use of MineTweaker (and, by extension, CraftTweaker) to "significantly change the way" his mods work. For certain mods, Reika insists that he must be notified before the release of any modpack that includes MineTweaker changes. Furthermore, he writes, "the mod's fundamental identity must remain intact," because "packs are most people's first experience with my mods and I want the first impression to be both accurate and positive, something not true if the mod is heavily altered."

The importance of author control and keeping the consumer's experience consistent with the creator's vision is recurrent. Notch himself struggled with this issue, not wanting to lose creative control but also refusing, based on his techno-libertarian sympathies, to take legal action to prevent modding (Murphy, 2015). ScriveShark maintains two Minecraft-related projects, one of which is a closed-source content mod, while the other is an open-source code library/utility for use with other mods. He distinguishes between mod development as writing a book versus inventing a tool, noting that the intended audiences for the two projects differ:

> [My content mod] is closed source for the reason I didn't want people cloning it whenever I had downtime without my consent. The modding community has seen it a few times that someone simply steals a mod to update it to a new version or tweak it to their tastes. They are usually well meaning, but it often ends up being very detrimental to the owner or mod itself. Basically, [my content mod] is closed source for the same reason I wouldn't write a novel in an open source context. [My library mod] is open source because it is meant to be shared. I want the tech to be improved by others, and I want it

> to be useful for everyone. It's more of a library/toolset for modders than a player toy, but meant to enable player toys to be implemented by modders. I'm very happy for [the library mod] to be cloned or extended. [The library mod], then, isn't a book/novel but a tool for making them. I'm more than happy to share my tools.

7.4.4 Control and concealment

The examples in this section have shown how gaming capital is embodied and mobilized in the (strategic) "doing" or performing of the trappings of authorship and ownership of intellectual property. Yet they can also be linked back to the discourse of strategic professionalization and "best practices," particularly with respect to coremods. A 2017 post on the Forge forums complains about a performance optimization mod called *Perfswitch*[90] that has been run through a code obfuscator to make it nigh-impossible for anyone to decompile it and examine the code. The poster suggests that it may be a coremod, and notes that it likely runs afoul of Forge's June 2017 "New Policy on Coremods"[91] which was issued concurrently with the release of Minecraft 1.12. The policy, posted by LexManos, seeks to impose soft regulation on coremodding by endorsing a set of three "best practices":

> Sadly core modding is still a thing. As always we request that you guys attempt to work with us to get changes into Forge instead of core modding it yourself. However, if you MUST we have decided to put forth to the community a few 'Best Practices' for core modding. The intention is that large communities such as FTB, Technic, and Curse work with us to promote these best practices.

Key among these is the assertion that coremods should provide visible source code. Lex writes:

> This will be a controversial standard, but my thoughts on it is that if you're screwing with someone else's code (which is the only reason to ever write a coremod), then you should be willing to show what you are doing. It is stressed that this is VISIBLE SOURCE only. It is not a requirement that you allow others to use your code, or modify and distribute it. It's simply that we can see it.

---

[90] A pseudonym, not the mod's actual name.
[91] http://web.archive.org/web/20171225221642/http://www.minecraftforge.net/forum/topic/58706-regarding-minecraft-112-and-policy-changes/

The goal is to "make the community more open and try to stem the number of coremods out there that have no reason to be coremods."

In response to the complaint about Perfswitch, Lex noted that Perfswitch was actually one of the issues the team had in mind when developing their new coremod policy. He further opined: "Perfswitch is a cancer on the community. People need to stop using it."

The tension between commercial logics of authorship and intellectual property on the one hand, and having an open community that follows best practices on the other, is not easily cast in terms of tactics and strategy. Instead, each position claims the backing of proper institutions: it is an instance of dueling strategies—but one between loose associations of amateur produsers, rather than the corporate giants or state actors that we would normally expect to find battling each other on equally-strategic footing. It also implies the need to bring a different inflection to the strategy/tactics narrative in the context of modder discourse.

7.5 Modding discourses in retrospect

What I mean is that, while I have been categorizing discursive acts as having a strategic or tactical character throughout this chapter, I actually hold that the presentation of an action as one or the other is itself discursively performed: modders are just as much talking *about* strategies and tactics as they are actually using them. I suggested in Chapter 6 that the strategic deployment of gridded chunk logics is a gesture towards a fantasy of being able to infinitely enclose and inscribe space. Similarly, I argue that rhetorics of "best practices" are actually expressions of a fantasy about rationalized, professionalized modding. Even strategic mobilizations of intellectual property law rest on fantasies—not only of flawless and equitable enforcement of law, but on the notion that legal codes have the power to impose stable order on the world, when in fact the body of law is always evolving, unfinished and unsettled. It may be that all strategy is, ultimately, a discursive gesture towards a fantasy.

7.6 Coda: Some final thoughts on gaming capital

Based largely on the investigations carried out in Chapter 7, there are indications that Bourdieu's original three categories of cultural capital can be meaningfully discerned as qualities attached to instances of gaming capital. Bourdieu (1986) proposed three "states" of cultural capital: the embodied (cultural literacy/competency), the institutionalized (credentials and qualifications), and the objectified (possessed artifacts for cultural consumption). The embodiment of gaming capital is well-established throughout Chapter 7, since I was primarily interested in gaming capital as a means of claiming and displaying cultural competencies in the domain of modding and game design.

Institutionalized gaming capital can be seen in the provision of "flair" tags to mod developers in r/feedthebeast. The forums and chat rooms discussed also use ranking and reputation systems: in addition to the possibility for some users to be forum moderators and administrators while others are ordinary users, many forums also assign ranks to users based on their accumulated activity. On MinecraftForum.net, these titles evoke the stages of progression in a Minecraft game. For instance, newcomers who have made a few posts are "Tree Punchers," while forum veterans with thousands of posts under their belts are "End Dwellers." On the Minecraft Forge Discord, the project lead is classified as "Forge God" (see Figure 7-1 on page 204). The exercises of power enacted through code itself—such as the Forge MDK change that made the Minecraft base classes read-only—also hinge on institutionalized gaming capital: those granted credentials by the institution (roles within the Forge development team) are in a position to extend the standards of the institution to other domains.

Objectified gaming capital—the possession of cultural artifacts—is not so readily apparent in this discussion, probably because Bourdieu did not anticipate a world of digitized cultural goods, which are not subject to the scarcity pressures that affect non-digital objects, and cannot generally be exclusively possessed. While objectified cultural capital can serve as a symbol or metonymy for

embodied competencies by acting as a signal for consumption and taste, it differs in that it is congealed in physical objects—which makes it more difficult to clearly separate embodied and objectified forms in digital contexts. We see little sign of objectified gaming capital in modder discourses because it is strongly attached to possession and consumption, something which is harder to meaningfully display in an online venue (embodied cultural capital, by contrast, is a key aspect of production). Displays of objectified gaming capital could include computer hardware enthusiasts' descriptions of their "rigs" on tech forums, or publicly viewable lists of the games in one's Steam library. In modded Minecraft *play*, objectified gaming capital is exemplified in multiplayer modpack contexts by possession of rare materials (like HT's amethysts and the ill-gotten diamonds I traded for them) and well-functioning laboratories that take advantage of all of the mod-specific machines and systems. Those examples, however, pertain to the consumption, not the creation, of mods.

VIII. CONCLUSION

> *I honestly hope that, twenty years from now, Minecraft will be seen by most people as just another one of those weird cringey fads that couldn't die out fast enough, yet there will be some weird avid community that's still passing around Java Edition—getting people to buy the game and get involved in the modding community.*
>
> —Omira, Minecraft modder

8.1 Other worlds of modding

My Master's thesis advisor liked to say that research projects are never finished, only abandoned. Aside from the obvious attractions of having more data of the same kind, across a wider range of sites or a longer period of time (more interviews; more forum threads; more chat room hours; more Better Than Minecamp conventions!), there are several trajectories of inquiry that were abandoned in my research plan, but that could bear fruit in future efforts. Before wrapping up my primary analysis, I will take a moment in this section to acknowledge the roads not taken.

This project was almost entirely limited to Minecraft Forge modding, which is only one type of Minecraft modding. The Bukkit API saga represents a whole different approach to understanding how mods should interact with the base game and how they should be able to reshape gameplay. It is a story replete with its own dramatic twists, including Mojang's hiring of key Bukkit developers in 2014, and the subsequent Digital Millennium Copyright Act takedown notice from a Bukkit contributor that effectively killed further distribution of the CraftBukkit server software (necessitating its eventual replacement with products like Spigot). As noted previously, Bukkit-compatible mods are the foundation upon which countless public servers operate, particularly those that focus on mini-game play. These servers, which accommodate tens of thousands of players, account for a significant share of the many Minecrafts in the world. A comparison of the methods of Forge and Bukkit modding, as well as the different player-ecosystems they foster, would further enrich scholarly perspectives on Minecraft as a co-created artifact. As Microsoft moves towards establishing

a new plugins interface and monetized marketplace for Bedrock edition mods, there will be further opportunities to study Minecraft modding in a radically different context.

Even within the domain Forge modding, this study has observed and interviewed a relatively small sample of modders. While many of the participants are well-known contributors within the modding community, more than half of the recruitment occurred through the cultural "niche" of Better Than Minecamp or through subsequent snowball sampling, which could create a selection bias.

Next, my research only accomplished a cursory analysis of the use and consumption of mods within the broader player base. Web-based surveys asking a larger sample of Minecraft players about the mods they use would allow for the construction of a broader catalogue of the many Minecrafts in existence. There are also intermediary producers of cultural goods that have only been briefly mentioned: those who build and maintain modpacks. Modpack curation is an art unto itself, with its own standards and practices, tactics and strategies. Interviews with modpack creators would provide a fuller picture of the circulation and articulation of cultural artifacts between developer, modder, curator, and player.

Finally, there is a need for meta-analyses comparing modding practices across multiple games. As Section 1.4 discussed, different game communities have different definitions of what constitutes a mod. They also produce different sorts of mods, and have different ways of making them. Most modding scholarship to date has focused on one game or one family of games (especially FPS games), and my work is no exception. Like these other authors, I have been keen to point out characteristics specific to the object of study I have chosen—such as the fact that Minecraft has one of the most vibrant modding scenes of any contemporary game, with mods playing a huge role in popular understandings of what Minecraft is, all without having any official modding support, all thanks to unsanctioned hacking. As a result, there are about as many theories of modding as there are modded games. A comparison across many games, offering very different contexts for modding, would be a step towards more generalizable theories of game modding.

As for the interpretations of modding that are expounded upon in these pages, there remain some loose ends to tie up.

8.2 Strategy, fantasy, and the promise of the platform

I have claimed many times now that strategies are based on fantasies, but I do not mean that they are not useful, or that they do not produce anything of value. They may not deliver on their ultimate promise, but (to steal a phrase from Mackenzie Wark)[92] the operations of strategy create the possibility for new things to come into the world, things that could not exist without a rationalized structure to attach them to. Without Forge's compatibility-enhancing strategies, I could never play Minecraft with a handpicked collection of 130 mods. Without the strategy, adopted by most major game developers, of separating the game engine and its development tools from game products, and of separating process from data, I would not be able to play Skyrim with a handpicked collection of 130 mods either. Useful rationalized scaffolds arise even from incomplete and ambivalent attempts to bridge the divide between a current reality and a fantasy destination.

Much ink has been expended in these pages on discussing the rationalization of modding, which is linked to the rationalization of game design and programming more generally. The further we go down the rationalizing rabbit-hole, the harder it is to ignore that all of the efforts, transformations, and rhetorical moves towards seamlessly rationalized modding point to the fantastical promise of a modular and infinitely-extensible all-platform on which any game idea can be built. This is, of course, a far-fetched fantasy, and nobody has actually claimed that Minecraft is a universal game engine, but emphasis on understanding it foremost as a "platform" (Duncan, 2011; Leavitt, 2013) or "grammar" (Tremblay et al., 2014) are gestures in this direction.

The dream of a universal platform on which any imaginable virtual experience can be built has long fascinated futurists, from science-fiction novelists to software and game developers. The Metaverse of Neal Stephenson's *Snow Crash*, the Oasis of Ernest Cline's *Ready Player One*—in their

[92] Wark wrote: "Hackers create the possibility of new things entering the world. Not always great things, or even good things, but new things" (2004, para. 004).

respective universes, these are *the only virtual platforms that matter*, with all user-facing interactive digital media being implemented on them. The term "metaverse" in particular suggests a system that universally supports other, more specialized systems. The word was used by Linden Lab founder Phillip Rosedale and his employees as a moniker for Second Life and a signifier of what they hoped to accomplish—an extensible platform for boundless creativity, a garden plot for emergence on which the world's residents would build whatever they imagined. As I have pointed out in these pages, this idea of a developer creating an environment that users will do the work of filling with content is a strategic move on the part of the developer to capture the creativity of crowds. Yet platforms are seldom as universal and extensible as the visionaries hope, being inevitably shaped by the political ideologies of their creators and the specific ways in which they imagine their users (Malaby, 2009).

We should be cautious, therefore, about making too much of claims that Minecraft is not a game, but a play-platform or a grammar out of which games and other interactive digital experiences can be made. There are many Minecrafts but that does not mean that every conceivable Minecraft is possible, or that all conceivable things are Minecrafts. This is a fantasy, one towards which a proliferation of APIs, best practices, and attempts to modularize the code are all oriented but can never achieve—nor do they need to, since their true value is in the possibilities they enable for us along the way. Like all platforms, Minecraft is more easily bent to some purposes than others: recall from Chapter 4 how, before Adventure Mode features were implemented, it was *possible* to make adventure experiences, but they were plagued by difficulties and contradictions. A "superflat" Minecraft world, which can be generated by setting an option in the game menu, presents itself as a blank slate. It beckons us to traverse its space, to inscribe our imaginations on it in three dimensions. It literally looks like a building platform. But Minecraft-as-platform is not *terra firma*—it is a bit muddy in spots; some uses are easy, but others get us bogged down in short order.

Is Minecraft really "built for mods," as Christiansen (2014, p. 32) claims? In *some* ways it is. Its open-ended gameplay, its near-infinitely expandable space, and its emphasis on building structures of infinite variety invite players to tinker and push to discover new possibilities. Its Java codebase makes reverse-engineering easier than it might otherwise have been. But Minecraft is certainly *not* built for modding in other ways. Parts of the source code are a tangled mess, being the result of a series of *tactical* moves in its early design to rapidly prototype a proof-of-concept, rather than the product of top-down planning. In earlier versions, process and data are not well-separated. Most of all, the developer has never provided any official support or tools for modding. This creates a strange contradiction in which games that offer more traditionally closed narratives and fixed worlds, like Half-Life and Skyrim, are better-built to facilitate modding than the open-ended "platform" world of Minecraft, because they have official modding tools available.

On the other hand, the unofficial nature of Minecraft modding is what enables the range of possible mods. Its origins in fan-driven code decompilation mean that Minecraft modding can alter the game in ways that are well beyond the capabilities of Skyrim's Creation Kit to reshape its own referent game. Icoso believes the lack of official support actually works in modders' favour—at least for Minecraft:

> Minecraft is entirely entangled between the game engine and the code, and I actually think that … if Minecraft had a mod API and people started only using the modding API and not just modding the game's code itself, modding would probably suffer heavily because we would just lose so many capabilities in what we can do to the game engine, because it's not well modularized for that.

This is perhaps what Zoll meant when he expressed concern at BTM that Minecamp would "rain on our parade" with an announcement of an official modding API—an announcement that did not materialize. There is a deep ambivalence between having structured support for creative activity, as provided through intentional strategy, and having the flexibility that comes from seizing opportunities on the wing. Strategy, like a road network, makes it easier to get to designated locations that have been enclosed within the logics of transportation, but at the cost of making the

territory a product of the road itself, erasing the idea that there were ever places to visit off-road. A compromise is perhaps the most that can be expected, as the fantasy of complete freedom combined with comprehensive structural support is unreachable. That said, Icoso suggests that some games, due to a modular design at their core, may more effectively manage the compromise than Minecraft does.

8.3 Whose game is it anyway?

In Alex Leavitt's (2013) analysis of Minecraft co-production, he finds that the vision of the "alpha artist" does not completely disappear, and that many players still regard Mojang as rightly having the final word in creative decisions. "It's Mojang's game," says one of Leavitt's interviewees (2013, p. 22). But is it really?

Minecraft was younger when Leavitt conducted those interviews, and he did not focus exclusively on modders. Whether because of the further proliferation of "many Minecrafts" since 2013, or my narrower focus on the subset of players who make mods of their own, my findings suggest that, outside of a technical legal definition, it is hard to maintain the impression that Minecraft belongs to Mojang anymore. We have seen how play at the edge of the game modes afforded by vanilla Minecraft expanded ludic possibilities in ways that were later re-codified in the software. We have seen how players and modders responded to Notch's call for "a huge melting pot of emergent gameplay" by forging a new identity for their community and for the product alike. We have seen how modders are increasingly willing to strike out on their own, rejecting the narratives and creative maneuvers offered by Mojang, and asserting their right and power to play, and to procedurally alter the parameters of play, on their own terms. We have seen them develop their own practices, standards, linguistic codes, and cultural signifiers, to the point that the variations and relationships between groups of modders are at now at least as important as—if not more than—the studio/modder relationship that has been the focus of most prior modding research.

8.4 Co-productive oscillations: power, rhetoric, and capital in the re-crafting of games

Contemporary perspectives on game modding have been informed by theories of audience activity, participatory culture, and media that originated in the milieux of mass media and mass print—not the least of these being Jenkins' (1992) theory of the textual poacher, propped up by de Certeau's model of institutional strategies and nomadic tactics. However, scholars have been careful to point out the ways in which these frames fall short for describing game mods (see Sotamaa, 2009, pp. 86-89; Sihvonen, 2009, pp. 125-130). One of the key issues with Jenkins' 1992 work is that it asserts the existence of a boundary between commercial producers and audiences that, according to later work (e.g. Banks, 2013), has since faded significantly. The strategy/tactics binary and the "poaching" concept have proven too limited to describe more recent developments. As discussed in Chapter 2, Jenkins himself later updated his take on participatory culture by introducing "convergence" to describe the blurring of producer/consumer boundaries. It is no accident that the turn to convergence and co-creation in the period 2000-2010 coincides with what is popularly understood as the new media revolution, with the rise of YouTube, personal blogging, and Web 2.0—coupled with increasingly visible online fan communities, and increasingly mainstream game modding taking advantage of online distribution. The reverse-side of Jenkins' convergence coin is, after all, the convergence of media *forms* under new media.

In this work, I have returned to the older theory of tactics and strategies, finding that it continues to have relevance. While I do not argue that it alone is sufficient, I have considered how its specific shortcomings can be used to illuminate in detail the everyday practices underpinning convergence and co-production. This is what I have learned:

1. Strategies and tactics are symbiotic.

There are two aspects to this symbiosis. First, *tactics evolve into strategies as players establish permanent settlements in developer space*. De Certeau claimed that the tactic "does not keep" whatever assets it obtains. Yet Minecraft modding, from the early days of jar hacking, shows that the artifacts

of tactical production are not only persistent and durable, but also assertive. Modders do not sneak in, poach what they wish, and vanish into the night—they show up, put down roots, and demand attention. This does not mean that the initial activities weren't tactical, but that many tactics are, from the moment of their appearance, in the process of establishing a terrain for new strategies. This is why I have preferred to think of modders as settlers and immigrants, rather than poachers. They become part of the establishment, while also introducing new strategies that reshape the establishment's identity. Sotamaa writes of corporations increasingly "colonizing the sphere of play and commodifying the results of players' work" (2009, p. 104), which is certainly true, but it is equally true that players are increasingly colonizing the sphere of game development. The settlement narrative of Minecraft gameplay and its developer's use of "melting pot" metaphors serve to reinforce the perception that some degree of player-driven production is to be expected.

Second, *strategies are what create the field for new tactics to arise*. I wrote above (Section 8.2) that strategies create frameworks for making new things possible, but tactics are all about bringing new things into the world too. Tactics, however, do so by redirecting the tools of strategy in novel ways towards unexpected results. Tactics in turn provide the fragments out of which new strategies are built, in an endless back-and-forth oscillation.

2. The ambivalent oscillations between strategy and tactics, and between player and corporate power, are the rhythms of media production in transition.

Sotamaa writes that "game culture originates in many sites, often at the same time defined both by resistance, exploitation and mutually beneficial relations" (2009, p. 101). My findings agree: the convergence of industry and modder practice has been bidirectional. Modding becomes professionalized, while commercial developers try to tap into "working anarchy" (Dyer-Witheford & de Peuter, 2009, p. 42) or the ingenuity and versatility of "ninjas" (Gallagher et al., 2017). The former—professionalization—is driven by a fantasy about the power of strategies, while the latter—

the enclosure of working anarchy within capitalism, the co-optation of nomadic power—is not a corporate use of tactics *per se*, but a strategic fantasy about the power of tactics.

The practices are not, however, converging on a common, settled state. Instead, it is an elastic relationship of pull-and-rebound. Enclosure is not complete; "capture is not seamless" (Dyer-Witheford & de Peuter, 2009, p. 27). Much like the tether/accretions oscillation that drives forward Minecraft gameplay (Goetz, 2012), the strategy/tactics oscillation is what drives forward the expansion of ludic possibilities.

Trying to predict where these oscillations are all leading (A new, radically democratized digital commons? A world of absolute corporate control over media content, backed by the cease-and-desist regime?) is probably a fool's errand, and I have a strict policy of one fool's errand per dissertation—a limit which I have already met (see the attempts to define "modding" in Section 1.4). What can be said, however, is that a transition away from the old order is in progress, as it has become impossible to even pretend that the ecosystem of corporate-controlled, top-down game production has not been destabilized. Minecraft would not be where it is today if not for its mods. As such, it is a prime example of audiences—no longer satisfied with peripheral participation—emerging from the shadowed stalls and stepping into the spotlight.

In the present moment, it is worth remembering that video games do not share the same history as other media forms. Digital games first emerged as *co-created new media* during the heyday of centralized *mass media*. Although they were subsequently absorbed into the mass media model for several decades, it is fitting that digital games are the vanguard of the post-mass media transition.

Since that vanguard is occupied by modders and mod-consumers as a new class of cultural stakeholders, and since their own decision-making processes are shaping the nature of ongoing modding practice, attending to the inter-modder relationship is more important than ever.

3. Modders develop rules and enact power over each other.

Although discussion of the modder/developer relationship remains important, and has been a necessary part of this analysis, I have stepped away from it where possible to focus greater attention on the under-examined structures, institutions, and boundaries within the modder community. My work has shown that modders co-regulate, establishing their own heterogeneous field of institutions and rules-systems. Minecraft modders contest the sites of meaning-making as much among themselves as they do against Mojang (if not moreso). While there remain multiple ways to mod (and even methods canonized in Forge modding are subject to being overturned every few years), every modder experiences forces, based on the circulation of gaming capital within the community, that push their activities this way or that. Player demand for interoperability between their favourite mods provided the initial opportunity for modders to stake out positions of strategic power from which to build the nascent institutions of modding, such as Risugami's ModLoader and the Forge API: in a world without mousetraps, the path is beaten to the door of whoever has one ready first, only in this case the mousetrap is a system for reducing the headaches of inter-mod incompatibility.

Such power continues to be enacted on a daily basis through the discursive operations of gaming capital in multiplicit forms, whether embodied—as in the expertise and exposure of the writers of modding tutorials, or institutionalized and proceduralized—as in the consciously-chosen features and limitations of the Forge MDK. Modders are also constantly resisting, redirecting, and renegotiating such power, but to ignore it entirely is to commit the "political suicide" of producing a jar mod that is incompatible with everything (recall Omira's observation from the beginning of Chapter 7). If scholars of digital games want to watch cultural production as it evolves, we must pay attention to how power is negotiated and parlayed among modders, through the vehicle of gaming capital.

Where does this leave Minecraft modders themselves? For now, in a state of unsettled settlement, constructing strategy inwards, threading tactics outwards, thriving in a milieu of contradiction. In his BTM address (Chapter 5), Zoll said that despite the "messy state" of Minecraft modding, the community was still around, and still producing new content. "Ultimately, we're still here," he remarked. "We've stagnated for six years now. I think we can get a few more going."

I think so, too.

APPENDIX A.
A VERY BRIEF SUMMARY OF OBJECT-ORIENTED PROGRAMMING CONCEPTS IN JAVA

In order to discuss Minecraft's operational logics at the computational and practice levels, it is necessary to first understand some core concepts of object-oriented programming (OOP)—namely, *instantiation*, *inheritance*, *overriding*, and *polymorphism*.

The object-oriented paradigm is an operational logic of programming in Java and many other languages: a running program is conceived as a collection of "objects", each of which contains some data or properties (called "fields" or "members") and has some set of defined procedural behaviours (called "methods" in Java). The program proceeds through the sequential invocation of methods, which can read and manipulate data and pass them between objects.

Most of the programmer's work is in defining the characteristics and behaviours of these objects. However, it is rarely necessary to design each object individually, as a program will often work with a large set of objects that all do more or less the same thing with different pieces of data. For instance, a programmer making database software for a bank might think of the customer accounts system as a collection of "Account" objects. Each Account[93] object *has* certain properties, such as a number, a customer name, a balance, and a list of past transactions. An Account object can also *do* certain tasks, like debiting, crediting, and compounding interest—tasks that will update balances and transaction lists. The important point is that while each Account object contains *different data* (different customers, different balances, etc.) and will do *different things* (the bank's clients are not all making the same transactions for the same amounts at the same time!), they nevertheless all track the *same sorts of data* (all Account objects have a balance, whatever it may be), and do the *same sorts of things in the same way*.

[93] In Java convention, a class name takes an initial capital letter, while variable names for specific objects or pieces of data do not.

This is why object-oriented programming involves defining the characteristics and behaviours of abstract types, or *classes*. These classes act as templates establishing the general characteristics of behaviours of a whole category of objects. At runtime, a program can *instantiate* one or more specific, concrete objects to contain and operate on relevant data. The Account *class* would therefore dictate that all Account objects have a numerical property called a balance, a text property called a customer name, and so on, without actually assigning specific values to those properties. It would further dictate that all Account objects can carry out some procedures called debiting and crediting, and provide instructions for how those are accomplished. Any instance of Account can then have its fields populated and its methods invoked in a manner specific to its own circumstances. The database program can instantiate Account objects from the same template en masse—which saves the programmer a lot of typing!

The next OOP concept we need to address is *inheritance*. Sometimes, the programmer wants to make a new type of object that is substantially similar to an existing class, but has or does some extra things. There is a way to make the new class *inherit* the characteristics and behaviours of the existing one, so as to avoid having to duplicate the common code. To illustrate, I will shift from banking to a different use case: that of designing a text-based adventure or interactive-fiction game, in which the player interacts with a text-based virtual world by means of typed commands. These games were popular from the 1970s to the 1990s, with Will Crowther's *Colossal Cave Adventure* (1977) and Infocom's *Zork* (1980) regarded as exemplars of the genre. OOP is particularly well-suited to this sort of project because what the computer regards as objects can correspond to those game-world entities that the player is supposed to regard as objects (for instance, an object in the OOP sense can also correspond directly to a fictional-world object like a rubber ball that the player can look at, pick up, drop, or throw).[94]

[94] While world-objects would typically correspond to programming-objects, the converse is not necessarily true. The command-line parser, and even the text commands themselves, might be implemented as objects in the program, despite having no tangible presence in the game-world.

Suppose, then, that as developers on a text-based adventure game, we have defined a class called `Thing`, which is the template for tangible, movable game-world items. Anything that the player can examine, touch, pick up, drop, or carry will be represented by an instance of `Thing`, which means that the class definition of `Thing` will describe how all of those actions will work, as well as defining basic properties of all `Thing` objects such as name and description.

Now we would like to make a more complex type of item—a `Container`, which is just like any other Thing except that the player can also put items into it and take them out. The `Container` class should be a *subclass* that *extends* the `Thing` class, so that it can inherit all of the core Thing behaviours and properties (saving us the trouble of duplicating that code), while adding new properties, such as `containedItems`—a list of contained items (each an instance of Thing) and `maxCapacity`—the maximum number of objects that can be contained (a number). We will also need to add new methods: `addItem` for putting something into the container (adding it to the list of contained items) and `removeItem` for taking something out (removing it from the list). Another new method, `canAddItem`, will determine whether it is possible to add something new, based on the container's `maxCapacity` and the number of items it is currently holding. The `canAddItem` method will return a simple true or false result, so our `addItem` method will have to check `canAddItem` first before doing the work of actually adding something.

So far, so good—our text-based world can now be populated with bins into which items can be put, and from which they can be taken. But now the creative director has informed us that they want more than just bins in this world—they want chests, which can be opened and closed, and only support item placement/removal when open. Naturally, we start by defining a new subclass to extend `Container`, which we will call `OpenableContainer`. It will have a new data field, `isOpened`, as well as two new methods to `openIt` and `closeIt` (these methods simply need to toggle the `isOpened` value). However, `OpenableContainer` does not just do something new that its superclass `Container` did not do, it also handles some `Container` behaviours

differently—specifically, `canAddItem` must check not only that the container has available capacity, but that it is opened (`isOpened` is true) as well. `OpenableContainer` needs to *override* the behaviour of Container's `canAddItem` method to handle this new situation.

It turns out that there are to be wandering cats in this game-world, and for the sake of immersive realism, the creative director wants these cats to attempt to insert themselves into any boxes, bins, or chests they might encounter in their travels. With different types of containers, there will be different criteria for whether a cat can occupy them. Bins of type `Container` can be entered as long as they have sufficient capacity, but chests of type `OpenableContainer` must also be open. Fortunately, the rules of inheritance mean that any object that is an instance of `OpenableContainer` *is also* an instance of `Container`, so that a cat's behaviour does not have to differentiate between the two, nor do we have to change anything about the way either type of container works in order to accommodate the addition of cats. A `Cat` object simply needs to be able to notice anything that is an instance of `Container` (directly or indirectly), and query its `canAddItem` method. The exact implementation of that method varies between `Container` and `OpenableContainer`, but the `Cat` need not trouble itself with such details. This situation, in which two objects of apparently the same type behave differently because one of them is actually an instance of a different sub-type, is called *polymorphism*. With polymorphism, the utility of OOP goes beyond saving the programmer from having to duplicate or rewrite code: it means that procedures can be black-boxed so that only the surface-level or "interface" behaviours of an object need to be exposed to other objects, allowing for different implementations of the same interface.

APPENDIX B. CUSTOM BLOCK MOD CODE

This annotated Java code, with accompanying JSON data, make up all the basic components of a simple Forge mod designed to run on Minecraft version 1.12.2. It defines a new custom block type and makes it accessible to the player in Creative Mode. The code is informed by prescriptions from the official Forge documentation at https://mcforge.readthedocs.io/[95] and is based heavily on Shadowfacts' Forge Tutorials (https://shadowfacts.net/tutorials/forge-modding-112/)[96] with minor modifications. All code comments (prefixed with a double slash '//') are my own.

File: ModBasic.java

This is the "main" source code file for the mod.

```
 1  package net.nicwatson.mods.basicblock;
 2
 3  import net.minecraft.block.Block;
 4  import net.minecraft.block.material.Material;
 5  import net.minecraft.creativetab.CreativeTabs;
 6  import net.minecraft.item.Item;
 7  import net.minecraftforge.client.event.ModelRegistryEvent;
 8  import net.minecraftforge.event.RegistryEvent;
 9  import net.minecraftforge.fml.common.Mod;
10  import net.minecraftforge.fml.common.SidedProxy;
11  import net.minecraftforge.fml.common.Mod.EventHandler;
12  import net.minecraftforge.fml.common.event.FMLInitializationEvent;
13  import net.minecraftforge.fml.common.event.FMLPreInitializationEvent;
14  import net.minecraftforge.fml.common.eventhandler.SubscribeEvent;
15
16  // The @Mod annotation helps Forge Mod Loader (FML) identify the mod's vitals.
17  @Mod(modid = ModBasic.MODID, name = ModBasic.NAME, version = ModBasic.VERSION)
18  @Mod.EventBusSubscriber          // This mod will listen for "events" fired on the Forge event bus
19  public class ModBasic
20  {
21          public static final String MODID = "modbasic";    // Unique mod identifier (all lowercase)
22          public static final String NAME = "Basic Block Mod";      // Proper (readable) name for my
23  mod
24          public static final String VERSION = "0.1";               // Current mod version
25
26          // The "proxy" handles anything that the server needs to do differently from the client.
27          // Since this means mostly client-side rendering, the proxy will be "activated" on both \
28          // sides but only the client version will do anything.
29          // The @SidedProxy annotation causes FML to populate the proxy field with the appropriate \
30          // kind ("common" version if the program is running server-side, "client" if client-side).
 >          @SidedProxy(serverSide = "net.nicwatson.mods.basicblock.CommonProxy", clientSide =
31                  "net.nicwatson.mods.basicblock.ClientProxy")
```

[95] Archived (August 2018): http://web.archive.org/web/20180810015052/https://mcforge.readthedocs.io/en/latest/
[96] Archived (May 2018): http://web.archive.org/web/20180509121249/https://shadowfacts.net/tutorials/forge-modding-112/

```
32          public static CommonProxy proxy;
33
34          // This is the declaration of the new block type object.
35          // It is an instance of the custom class BlockNifty.
36          // Its unique ID (string) is "customblock". ID number will be chosen by FML at runtime.
37          public static final BlockNifty myCustomBlock = new BlockNifty(Material.ROCK, "customblock");
38
39          // The EventHandler annotation means that this method will be called when FML dispatches a \
40          // "FMLPreInitializationEvent". This is a good place to handle some basic setup of the \
41          // custom block type's characteristics.
42          @EventHandler
43          public void preInit(FMLPreInitializationEvent e)
44          {
45                  // In Creative Mode, the player will be able to find this new type of block under \
46                  // the "Decorations" tab.
47                  myCustomBlock.setCreativeTab(CreativeTabs.DECORATIONS);
48          }
49
50          // FML must "register" any mod/custom block types into a master list. (Part of the reason \
51          // is to map a unique ID *number* to that block type, since the code here only defines the \
52          // ID as a string, and the chunk arrays in the save file store integers, not strings.)
53          // This method listens for FML "announcement" (event) that it's time to register blocks, \
54          // and then asks FML to register it.
55          @SubscribeEvent
56          public static void registerBlocks(RegistryEvent.Register<Block> e)
57          {
58                  e.getRegistry().register(myCustomBlock);
59          }
60
61          // This is where the mod asks FML to register items (i.e. things that can be carried or \
62          // dropped). Here we use it to generate an "ItemBlock" which will serve as the prototype \
63          // for the in-inventory version of the custom block.
64          @SubscribeEvent
65          public static void registerItems(RegistryEvent.Register<Item> e)
66          {
67                  e.getRegistry().register(myCustomBlock.createItemBlock());
68          }
69
70          // This registers special "item renderer" handlers, which will tell Minecraft how to draw \
71          // carried/droppable items, such as our block when it is being carried (as opposed to \
72          // placed in-world as a block). Note that it calls a method on the proxy, which means that \
73          // the result will differ on the client side vs. server side (polymorphism at work).
74          @SubscribeEvent
75          public static void registerModels(ModelRegistryEvent e)
76          {
77                  proxy.registerItemRenderer(Item.getItemFromBlock(myCustomBlock), 0,
78                          myCustomBlock.name);
79          }
    }
```

File: BlockNifty.java

This subclass of Minecraft's Block defines special behaviour for the new block. Although the custom block does not actually do anything special compared to ordinary blocks like stone or wood, Forge is set up to expect all custom block types to be instances of custom Block subclasses, for the purpose of handling their render logic.

```java
package net.nicwatson.mods.basicblock;

import net.minecraft.block.Block;
import net.minecraft.block.SoundType;
import net.minecraft.block.material.Material;
import net.minecraft.block.state.IBlockState;
import net.minecraft.item.Item;
import net.minecraft.item.ItemBlock;
import net.minecraft.item.ItemStack;
import net.minecraft.util.math.BlockPos;
import net.minecraft.world.World;

public class BlockNifty extends Block
{
        protected String name;  // This stores the block's string ID

        // The constructor method is called to make a new instance of BlockNifty, specifying two \
        // properties - Material (rock, wood, etc.) and name (which will be the block's unique \
        // string ID)
        public BlockNifty(Material mat, String nameToSet)
        {
                super(mat);                             // Invokes superclass behaviour
                this.name = nameToSet;                  // Set string ID
                // The unlocalized name is the default "display name" of the block, which is used \
                // as a key to look up its translation in a data file according to the language \
                // setting the player is using. It doesn't have to match the string ID, but it's \
                // less confusing if it does.
                setUnlocalizedName(nameToSet);
                // The registry name, used by FML, *should* match the string ID to avoid serious \
                // confusion.
                setRegistryName(nameToSet);
                // These hard-coded properties will apply to ALL custom block types that \
                // instantiate BlockNifty.
                setHardness(3.5F);              // How hard the block is to break.
                setSoundType(SoundType.ANVIL);  // When placed or stepped on, block will make \
                                                // sounds like anvil.
        }

        // Creates an ItemBlock instance to represent this block as an item type in inventory.
        // (All custom blocks that drop as themselves will do more or less the same thing here)
        public Item createItemBlock()
        {
                return new ItemBlock(this).setRegistryName(getRegistryName());
        }
}
```

Info: Common and client proxies

As a multiplayer game, Minecraft needs processes running separately on the server, which tracks the Minecraft world itself, and on the client, which presents information about the world to the player. Even in single-player mode, Minecraft uses separate server and client processes. Graphics are only handled on the client, but a lot of code is common to both processes, including most of the content of ModBasic.java and BlockNifty.java above.

In order to ensure that graphics logic only runs on the client side, Forge uses a "proxy" system. A special handler object, called a proxy, is attached to the main mod file. On the server side, the proxy will just be dummy, while on the client side, the proxy object will perform functions having to do with client-side graphics. The proxy is activated and told to "do its stuff" on *both* sides, but the server version simply does nothing when invoked.

The proxy object instantiates either `CommonProxy`, or its subclass `ClientProxy`, which means that in either case it can be declared as an instance of `CommonProxy` (just as in the polymorphism example from Appendix A, where any instance of `OpenableContainer` could also be treated as an instance of its superclass `Container`). The @SidedProxy notation in ModBasic.java tells Forge that, on the server side, the proxy object should be a direct instance of `CommonProxy`, while on the client side it should be a direct instance of `ClientProxy`. Thanks to method overriding and polymorphism, the proxy's behaviour will be determined by which of the two classes is its direct type.

File: CommonProxy.java

This is the server-side "dummy" proxy that does nothing.

```
 1  package net.nicwatson.mods.basicblock;
 2
 3  import net.minecraft.item.Item;
 4
 5  public abstract class CommonProxy
 6  {
 7          public void registerItemRenderer(Item item, int meta, String id)
 8          {
 9
10          }
11  }
```

File: ClientProxy.java

This is the client-side proxy, which generates the necessary data to link custom items from the mod to their render model JSON files.

```java
package net.nicwatson.mods.basicblock;

import net.minecraft.client.renderer.block.model.ModelResourceLocation;
import net.minecraft.item.Item;
import net.minecraftforge.client.model.ModelLoader;

public class ClientProxy extends CommonProxy
{
        // The @Override annotation means that this method is overriding the behaviour established \
        // by the superclass. It actually has no significance to the compiled code, but exists for \
        // the benefit of the programmer, because it informs the compiler of the programmer's \
        // intention for this method to be an override. If, due to a typo, the method name did not \
        // match that in the superclass, overriding wouldn't happen, but thanks to @Override, the \
        // compiler will notice that the misnamed method isn't overriding anything and throw a \
        // warning.
        @Override
        public void registerItemRenderer(Item item, int meta, String id)
        {
                // Registers the JSON file that provides the rendering model for the given item type
                ModelLoader.setCustomModelResourceLocation(item, meta,
                        new ModelResourceLocation(ModBasic.MODID + ":" + id, "inventory"));
        }
}
```

Resource files

These files provide Minecraft with the information it needs to visually render blocks and translate their display names into the local language. Separating these data files from the code makes it easier to change block appearances and names with custom resource packs and translation files.

Most of these files use JavaScript Object Notation (JSON). To work with the Java code above, the file names must correspond exactly to the custom block's ID ("customblock"), and they must be organized in a specific folder structure. The subdirectory of the assets folder must exactly match the mod's unique ID ("modbasic").

File: assets/modbasic/blockstates/customblock.json

This file can be used to tell Minecraft to use different render models for the same block according to the "state" it is in. In this case, it only has one default state.

```
{
    "variants": {
        "normal": { "model": "modbasic:customblock_model" }
    }
}
```

File: assets/modbasic/models/block/customblock_model.json

This file defines the model that Minecraft will use to render the block in-world. It uses an existing "cube" template and links each of the six faces to one of six texture files. The "particle" refers to the

texture that will be used to show fragments when the block is struck with a tool. This example uses the texture from vanilla Minecraft's lapis lazuli block.

```
{
    "parent": "block/cube",
    "textures": {
        "down": "modbasic:blocks/block_simple_face0",
        "up": "modbasic:blocks/block_simple_face1",
        "north": "modbasic:blocks/block_simple_face2",
        "east": "modbasic:blocks/block_simple_face5",
        "south": "modbasic:blocks/block_simple_face3",
        "west": "modbasic:blocks/block_simple_face4",
        "particle": "blocks/lapis_block"
    }
}
```

File: assets/modbasic/models/item/customblock.json

This file describes how the block should be rendered as an item in inventory. In this case, the inventory model is just the same as the in-world block model.

```
{
  "parent": "modbasic:block/customblock_model"
}
```

Directory: assets/modbasic/textures/blocks/

Contains PNG files for block textures referred to in the JSON files above:

- block_simple_face0.png
- block_simple_face1.png
- block_simple_face2.png
- block_simple_face3.png
- block_simple_face4.png
- block_simple_face5.png

File: assets/bin/modbasic/lang/en_us.lang

This translation file maps custom blocks' unlocalized names (as seen in BlockNifty.java, above) to their proper display names for American English.

```
tile.customblock.name=That thar dern ol' block thing
```

File: assets/bin/modbasic/lang/en_uk.lang

This translation file maps custom blocks' unlocalized names (as seen in BlockNifty.java, above) to their proper display names for British English.

```
tile.customblock.name=Her Majesty's loyal olde example block of goode custom
```

APPENDIX C.
FORGE HOOKS: REGISTRIES AND EVENTS

The basic concept of a code hook as an entry point for a mod to intervene in Minecraft's behaviour is described in Subsection 4.2.1, while the event model is introduced in Subsection 6.4.1.

Any edit that Forge makes to the base code to allow modders to intervene or attach new behaviour is called a "hook," but hooks come in different types in terms of how they are presented to the modder. In general, Forge distinguishes between interventions that add (or substitute) new data structures (blocks, items, creatures) with their own self-contained behaviours, and those that make changes to standard procedures that are already well-defined in vanilla. In the former case, the modder specifies the properties of the new game element and then "registers" it with Forge (in the Appendix B example code, the properties of a new block are established in BlockNifty.java, and the registration happens in ModBasic.java on line 57). Once registered, the new data structure can actually be recognized by other code modules. Under the hood, Forge registries are procedural interruptions in Minecraft's core "setup" methods that define all of the vanilla data structures—that is, lines of code (hooks) that refer to the Forge registries must be hacked into the base classes, so that those registries actually do something. This doesn't mean that each new block type needs its own hook—only the registry as a whole needs a reference in the base classes, because it contains an expandable list of new data structures. The concept of the registry hides the procedural nature of the intervention from the modder, presenting the operation as adding something to a list.

The event model, as modders typically use the term, is for interventions that are explicitly and unapologetically procedural. Figure C-1 (page 255) depicts the event model in action. Suppose I want to patch a Minecraft annoyance relating to the fact that tamed wolves (see Section 7.4) sometimes sit on cacti, taking damage until they die, because they are stupid to move! (Their AI is a bit rudimentary.) Rather than trying to fix how wolf AI works, I can just make a mod that cancels all damage to wolves, if the source is a cactus. This is possible because the Forge API installs a hook in

the Minecraft module that handles deducting health from creatures when they are hurt (Figure C-1, label 1). Whenever any creature is hurt, before actually assessing the damage, this hook dispatches a `LivingHurtEvent` (2), which is an object containing information about what is happening. The event goes to the event bus[97] (3), which maintains a list of "subscribed" event handlers, and the sorts of events that they want to hear about (4). My own mod contains a method that I have designated as an event handler (5) for events of type `LivingHurtEvent`, and it have previously subscribed to these events—meaning I told the event bus that I want this method to be activated and receive information whenever a `LivingHurtEvent` is dispatched to the bus. The bus thus invokes my method, passing it the event object (6). In examining the object, I can determine whether the entity being hurt is a tamed wolf, and whether the source of the damage is a cactus. I can then decide what I want to do about it. One possibility is to *cancel* the by invoking a cancellation subroutine attached to its object: the hacked Minecraft method that dispatched the event will check to see if it was cancelled by any of its handlers (7) (the original method doesn't lose its ability to see the event just because it has dispatched it). If so, the method will omit carrying out the task of actually deducting the creature's health, but otherwise it will proceed as normal. Cancelling doesn't mean removing the event object (preventing other subscribers from seeing it)—it means vetoing the underlying task. Not all events are cancellable, but there are workarounds: for instance, if I were not able to cancel `LivingHurtEvents`, my event handler could instead increase the wolf's health by the exact amount that it is about to be reduced. Whether or not the event is cancelled, the handler can still perform other tasks, such as making a diagnostic note in a text file: "*sigh* A wolf stuck itself on a cactus, AGAIN."

[97] This is a "bus" in the sense of a data conduit, but could also be thought of as a transport vehicle that carries references to the event object to subscribed event handlers at each "bus stop." Thanks to Mia Consalvo for providing this insight.

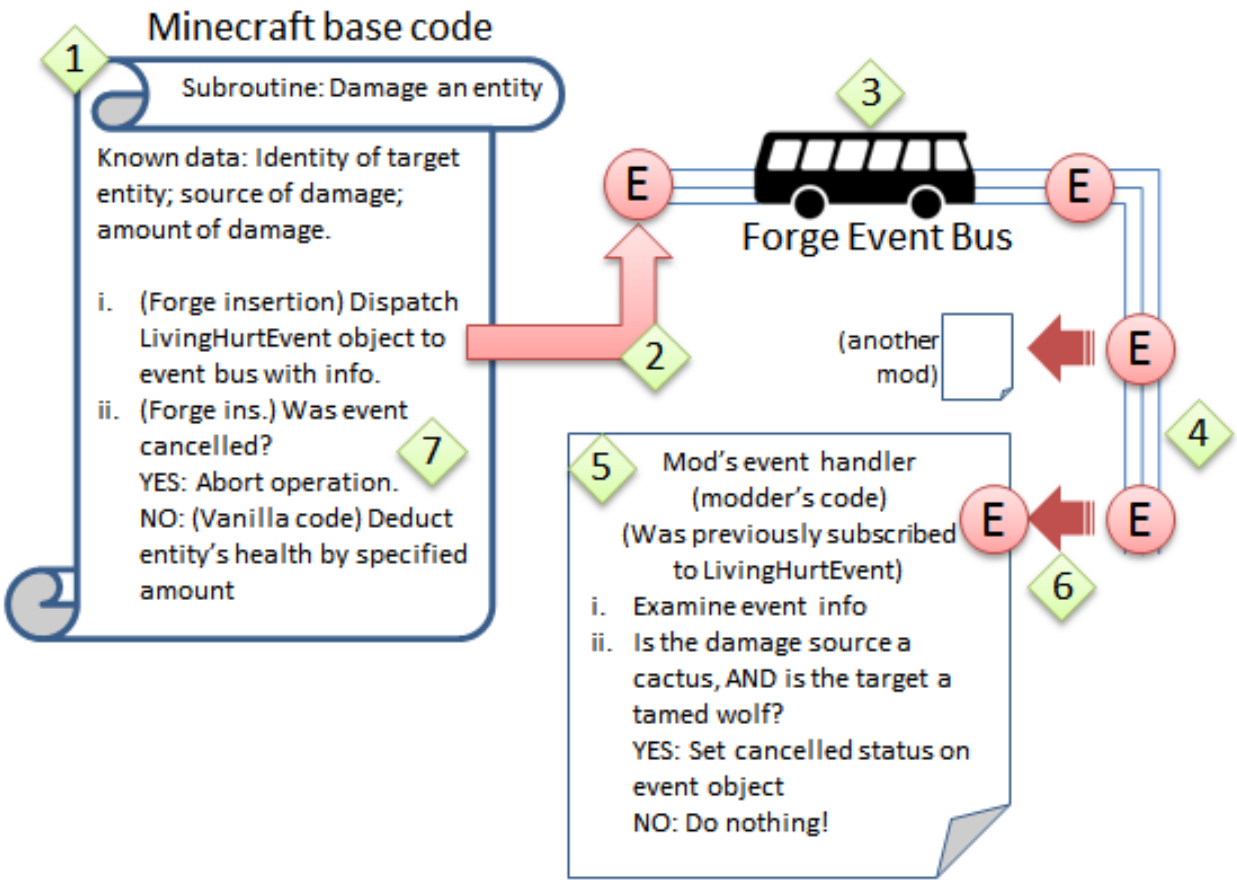


Figure C-1. The Forge event model in action.

This example has been a case of what capitalthree means when he talks about mods acting as event consumers (Subsection 6.4.1). When he refers to mods being event *producers*, what he means is that they would implement their own equivalent to the method at (1) in the diagram, which dispatches its own event, which could be an instance of one of the standard event types specified in the Forge API, or a custom type of event. This would allow mods to intervene not just with core Minecraft behaviours, but with the things that *other* mods do. A potential use case would be a magic mod in which players can cast spells on animals, in which I want other modders to be able to intervene with the spellcasting operation. In the method that carries out the results of spellcasting, I would install a hook (no bytecode hacking required this time, as this is my own source code to begin with) that dispatches an event, so that any other modder can subscribe their own handler to it. That

way, other modders can change how my mod interacts with theirs, without having to actually edit my code.

GLOSSARY

Adventure Mode – A GAME MODE in which the player is able to move around, collect objects, fight monsters, and place blocks, but cannot destroy existing blocks. Used by end-user players of puzzle and adventure MAPS built within Minecraft worlds. See Section 4.1.

API – Application Program Interface, a set of programmed tools and standards that allows a programmer to write new code that can use and communicate with existing program modules (which provide the API). The API defines the possible interactions between its underlying modules and external code.

Base class – A CLASS file belonging to the original Minecraft codebase, as opposed to one that is added by a mod. Some mods work by directly editing base classes, while other mods add new classes (although at least one base class modification is always required to provide a HOOK for new classes to intervene in the program). A VANILLA class is a base class that has not been modified.

Bedrock (1) – A rock layer that covers the entire bottom layer of a Minecraft world, 63 metres below sea level. In ordinary SURVIVAL play, bedrock cannot be broken. It can, however, be removed in CREATIVE mode. Beneath the bedrock lies the lower VOID.

Bedrock (2) – A Minecraft game engine, written in the C++ programming language, on which the Windows 10, Xbox, and Nintendo Switch EDITIONS rely. The term is also used to refer to this family of editions. Bedrock-based Minecraft editions differ from the Java Edition that is the primary focus of this document.

Block (1). A one-cubic-metre spatial slot in the Minecraft world's three-dimensional grid. It may or may not be occupied by some stationary physical object or material.

Block (2). An object that occupies a spatial grid-slot (see **block**(1)) in the Minecraft world's three-dimensional grid—for instance, a block of granite, wood, wool, or water. The object need not be cubic or solid either—hanging vines, flowers, and doors are all considered blocks.

Block (3). A unit of distance in Minecraft, equal to the width of one **block**(1), and generally said to be equivalent to one metre.

Block Entity – A complex object that attaches to certain types of BLOCK(2), when the representational possibilities of a numerical BLOCK ID plus 16 possible METADATA values is insufficient. For instance, a storage chest, which can contain many different items in many different orders, needs a block entity to keep track of the contents of its inventory. Prior to 1.7.2, block entities were called TILE ENTITIES. A block entity is not the same sort of thing as an ENTITY.

Block data –Extra numerical data associated with a BLOCK(2), defining secondary characteristics such as the block's orientation, whether it is receiving REDSTONE power, and so on. Also used to distinguish between "subtypes" for certain blocks (e.g. different types of wood, different colours of wool). As block data is a number from 0 to 15, a single block can have at most 16 possible combinations of its various secondary characteristics, with each combination corresponding to one of the numerical values. As of Minecraft 1.8, source code now manipulates block data by means of abstract concepts called BLOCK STATES. Also called METADATA.

Block ID – A data value that uniquely identifies a type of BLOCK(2)—for instance, distinguishing a cube of glass from a ladder or a flower. Block IDs are numbers under-the-hood (as stored in SAVE FILES), but have been presented as text strings in-game since Minecraft version 1.7.2.

Block state – An abstraction, used in Minecraft code and in-game commands since version 1.8, that describes transitory or ancillary characteristics of a block, such as its subtype, orientation, or REDSTONE powered status. The block state abstraction uses meaningful text labels to separate and describe these secondary properties. However, beneath the abstraction, block states correspond to numerical BLOCK DATA values from 0 to 15.

Bukkit – A Minecraft modding API for building SERVER-side plugins that can change game mechanics. Bukkit plugins can do things like adding new typed administrator commands, providing scripted encounters with NPCs (non-player characters), implementing trade systems, enforcing property ownership regimes on multiplayer servers, teleporting players around, and changing world generation rules. They cannot add new kinds of blocks or machines, or otherwise make changes that would need to be reflected in each player's CLIENT program. Bukkit plugins are often used on servers that host Minecraft MINI-GAMES like SKYBLOCK and HUNGER GAMES deathmatches. The original modified server package capable of running Bukkit plugins was called *CraftBukkit*, but it has been defunct since 2014. Other Bukkit-compatible server packages have stepped in to replace it, such as *Spigot*.

Bytecode – The actual contents of a Java CLASS file: binary machine code that can be interpreted by the JAVA VIRTUAL MACHINE in order to execute the instructions of a program that was written in Java. The bytecode is the result of compiling the human-written source code. Bytecode is not human-readable without special tools, and even then is considerably more arcane and opaque than original Java source code. It is, however, platform-independent, in that a Java Virtual Machine installed on any platform or operating system can read Java bytecode that was compiled on any other platform. It is also considerably easier to directly manipulate, and to reverse-engineer (decompile), than platform-specific "native" machine code generated by other types of compilers, like C++. Bytecode injection, also called binary patching, means directly editing the bytecode contained within a class file to alter the behaviour of the program.

Chunk – A geographic and computational constituent of a Minecraft WORLD, corresponding to a column of BLOCKS(1) 16 by 16 metres wide and 256 metres tall. Portions of the world are loaded just-in-time as the player moves towards them, and unloaded once the player moves away; this loading and unloading is always done in chunk-sized units. See Subsection 6.3.2.

Class file – A digital file that contains BYTECODE and corresponds to a module in a Java program. It corresponds to a *class definition* in the object-oriented programming sense (see Appendix A). A Java program, as deployed to an end user, usually consist of multiple classes bundled together in a JAR FILE.

Client – In a multiplayer context, the instance of Minecraft that is running locally on a player's computer, presenting graphical information about what is going on in the world to which the player is connected. The actual internal game logic (world generation, monster AI, coordinating between multiple connected clients, etc.) is being handled by the remote SERVER. In single-player Minecraft, computational tasks are still conceptually divided between a "client thread" (or sub-process) that handles the user-facing presentation of the game, and a "server thread" that performs the core game logic.

Code hook - A minimal change to a base class, providing a means of interrupting the program to perform some new, modded procedure, which is specified in a separate class file. See Subsection 4.2.1.

Command block – A BLOCK(2) in a Minecraft world that can execute administrator commands of the kind that would normally be typed by a player in the chat interface. When pressed, a pressure plate or button attached to a command block will trigger execution of the command.

Commit – See GITHUB

Coremod – Also "core mod", a type of mod that uses on-the-fly ASM bytecode injection to directly modify (or "transform") portions of VANILLA code (BASE CLASSES) whenever the Minecraft program is run. Coremods can change the program at a low level, provide access to normally protected data, and accomplish tasks that are not possible with the stock CODE HOOKS provided by Forge. However, coremods are controversial because they can cause stability, compatibility, and even security issues by invalidating assumptions that other mods make about the behaviour of vanilla code. (See Section 4.2.)

CraftTweaker – A Minecraft mod used to configure (tweak) aspects of mods in a MODPACK, in order to ensure interoperability, gameplay balance, and fun. It is often used to alter vanilla or mod-provided crafting RECIPES—for instance, by making one mod's recipe use an item that is part of another mod. Server administrators sometimes also use CraftTweaker to adjust the parameters of a server's modpack. Tweaks are specified in a plain-text file, which is loaded into the game at runtime. CraftTweaker is the successor to MINETWEAKER, which was discontinued in 2015. https://www.curseforge.com/minecraft/mc-mods/crafttweaker

Creative Mode – One of Minecraft's GAME MODES, in which the player is invincible, can fly, and has access to all types of blocks and items in unlimited quantities. It effectively turns a Minecraft world into a virtual Lego playground.

Difficulty – A Minecraft game preference that determines how much trouble the environment will give to the player. Options are Peaceful, Easy, Normal, and Hard. At higher difficulties, monsters spawn at greater rates, are tougher, and do more damage to the player. On the Peaceful setting, hostile monsters do not appear at all, although the player could still be harmed by falling off of a cliff or trying to swim in a lake of lava (unless in CREATIVE MODE).

Dimension – An adjunct space within a Minecraft world, like a parallel universe. Dimensions are separate from one another, and movement between them generally involves the use of a mystical portal. In addition to the primary Overworld, vanilla Minecraft provides a Hell-like dimension called the Nether, and a mysterious dragon's abode called The End. Many mods add additional dimensions for players to explore.

Edition – In this document, refers to one of the families or lineages of Minecraft, such as Java Edition, Pocket Edition (a historical name), Bedrock Edition (an unofficial label), Legacy Console Edition, and Education Edition. Different editions are typically built using different programming languages and/or different game engines, and correspond roughly to different families of computing platform (except for Education Edition).

Entity – A dynamic, movable object within a Minecraft world that is not a BLOCK(2) and is not locked to the BLOCK(1) grid. Player avatars, creatures, and monsters are examples of entities, as are minecarts, flying arrows, dropped items, and hangable paintings. The term should not be conflated with BLOCK ENTITY or TILE ENTITY, which refer to a different concept.

Event – An object used in the Forge API to communicate to mods that something has happened in the game, giving those mods the opportunity to intervene. See Appendix C.

Game Mode – A general parameter that determines how a game of Minecraft is played. See ADVENTURE MODE, CREATIVE MODE, SURVIVAL MODE. See also Section 4.1.

GitHub - A website for hosting repositories ("repos") containing the source code of open-source projects. It uses a version control system called *git* to keep track of incremental changes to the code, and also facilitates *forking* a project into branches that can be updated independently thereafter. Would-be contributors can submit *pull requests* that ask the project maintainer to "pull" the suggested source code edits into the repo. They can also make their own forks of the project, either to take development in an entirely different direction, or just as a test bed for experimentation. This leads to the perception of forking as a remedy of last resort for contributors whose suggestions are rejected: if you think you can do better, then prove it by making your own fork!

To *commit* is to instruct the repository to incorporate a set of updates and changes into the repo, and "commit" is also commonly used as a noun to refer to that set of changes. Although strictly speaking the commit only contains information about what has changed, in practice viewing a commit on GitHub shows the unchanged parts of the code too, so viewing a commit is like seeing a snapshot of the repo immediately following the update.

GUI – Graphical User Interface

Hook – see CODE HOOK

Hunger Games – A competitive multiplayer Minecraft MINI-GAME that emulates the titular premise of the books and films of the same name: players run around a virtual arena, fighting to the death.

Item – An object in Minecraft that can be picked up, carried, and dropped. This includes tools, weapons, food, and building BLOCKS(2). When dropped, an item becomes an ENTITY version of itself until retrieved. Most building blocks can be harvested and collected as items. Strictly speaking, when collected and carried, they are not blocks, but items (sometimes called "item blocks" to distinguish them from other items). These can be placed, turning them back into blocks in the environment. They can alternatively be dropped like other items.

Jar – Java ARchive, a file that bundles multiple Java class files, as well as various data assets, into a package that can be deployed to the end-user.

Jar mod – A rudimentary form of modding in which vanilla BASE CLASSES are replaced with modded versions. This is "destructive modding" because two mods will be incompatible if they both rely on replacing the same base class. See Subsection 4.2.1.

Java Virtual Machine – Part of the Java Runtime Environment, this is a kind of virtual computer (i.e. one that is simulated on another platform, a computer built entirely of software) that interprets Java BYTECODE and translates it on the fly into instructions that can be understood by the host platform.

JSON - JavaScript Object Notation, a standard for using plain text to structure data. Minecraft stores some game data in JSON, particularly that which defines how objects are to be presented visually in-game. JSON files can be edited externally using ordinary text editor software. Players can also use JSON notation to visually format text for in-game signs and chat messages.

JVM – See JAVA VIRTUAL MACHINE

Launcher - A program that does the initial work of loading and executing the Minecraft program. It validates the player's Mojang account credentials, checks the resource files, downloads updates as necessary, and then spins up the game. Since 2013, the official Minecraft Launcher has supported multiple user-configurable "Profiles" which can be set to use different versions of the game.

Several fan-made launchers also exist. MultiMC is popular for its focus on making it easy to manage multiple modpacks and switch between them. Some large mods, especially those approaching "total conversion" status, have offered their own custom launchers—The Aether II being an example. The Technic and Feed The Beast (FTB) launchers began as preconfigured packages for their titular modpacks, but have since expanded into modpack "platforms"—the FTB launcher provides easy access to the dozens of modpacks that have been assembled by the FTB team, while the Technic Platform hosts thousands of modpacks, most of them assembled by individual users rather than by a curatorial team. The custom launchers are really just conveniences for users, as any modpack can, in principle, be played using the original Mojang Launcher alone.

Library – A set of program modules that are intended to provide support functionality to other programs. For instance, a *font rendering library* would be a suite of program modules providing subroutines that could be used by another program for rendering and displaying stylized text on the screen. A library usually defines valid interactions with other programs by means of an API.

Localization – The process of altering content in a software package to better match the practices and preferences of a particular linguistic "locale." This means not only translating text into different languages or dialects, but also changing date formats, or converting between metric and imperial units. Developers typically separate such text data from game code in order to streamline the process of localizing a piece of software for multiple target markets, because this way localization files can be swapped without having to change the program code. This also makes it easier for end-users to make and distribute their own localizations—a process that Minecraft affords through its RESOURCE PACK system.

Map – Generally synonymous with a Minecraft WORLD itself (rather than a diagram thereof). The term likely originates with first-person shooter games. While the spatial units that make up a single-player linear-progression game like DOOM may be called "levels," the non-ordinal relationship between arenas in multiplayer games like Quake may be what led to them being called "maps." (Not to be confused with the in-game Minecraft item called a "map," which actually *is* a diagram of the terrain in a portion of a Minecraft world. Fortunately, those sorts of maps are not discussed elsewhere in this document. However, strict adherence to the principle of Exactitude in Science compels me to mention them here.)

Metadata – see BLOCK DATA

Method – A subroutine in Java code (or object-oriented programming more generally). A method is attached to (belongs to) a specific object. See Appendix A.

MCEdit – An external editing utility that allows one to view and make changes to one's Minecraft world out-of-game. Provides tools for quickly building (or deleting) large, complex structures. Different incarnations of MCEdit can be found at https://www.mcedit-unified.net/ and http://www.mcedit.net/.

MineTweaker – A predecessor to CRAFTTWEAKER, serving the same function. Continued development on MineTweaker ended in 2015, but it remains available for use with Minecraft versions up to 1.8.8. http://minetweaker3.powerofbytes.com/

Mini-game – A circumscribed play event that takes place within a Minecraft world, but has rules different from typical Minecraft play. Examples include HUNGER GAMES tournaments, SKYBLOCK competitions, and SPLEEF.

Mob – Any non-player creature (human or non-human, hostile or friendly) that occupies the Minecraft world. The term is short from "mobile" and originates with early digital role-playing games and MUDs (Multi-User Dungeons, the text-based ancestors of modern Massively Multiplayer Online Role Playing Games). A mob was so-called because it could independently move around the world (although not all mobs necessarily did so).

Mod – A piece of software or a body of digital data that modifies the process and experience of a digital game such that it differs tangibly from its original form, on either a procedural level or a cultural/experiential level. See Section 1.4.

In this document, "mod" always means a game modification. Elsewhere, it has other potential meanings.

In some game architectures, like BioWare's *Neverwinter Nights*, "mod" is understood to stand for "module" and refers to a fan-made playable adventure or "campaign" separate from the original story. Modules are typically made using developer-provided tools, like BioWare's *Aurora Toolset*, and the game software natively supports loading custom modules.

In web-based message boards, "mod" is short for "moderator." This can make reading message boards pertaining to game modding somewhat confusing.

Mod loader – A type of mod that edits (or HOOKS) Minecraft BASE CLASSES so as to allow other mods to intervene in Minecraft behaviour and add features without doing any further base-class editing themselves.

Modpack – A collection of mods that are distributed together, with the intention that they be played together. Modpacks are assembled by curators who may or may not be mod-makers themselves, and can contain mods from many different authors. A modpack constructs a particular context of play with its own rhythms and dynamics: thus, the curator must carefully choose mods for inclusion based on their interactions. Curators can use tools like CRAFTTWEAKER or MINETWEAKER to ensure balance and interoperability between mods in a pack.

NBT – Named Binary Tags, a structured format developed by Mojang for representing data about Minecraft worlds. Like XML, it supports hierarchical elements (tags) that can contain other tags. However, unlike XML, it uses binary codes instead of text to identify tags, meaning that it cannot be manipulated directly with a text editor.

Obfuscation – A practice by which what was once human-readable source code is scrambled and rendered difficult-to-understand by automated means. This makes it much more difficult for unauthorized parties to reverse-engineer the bytecode into something intelligible.

Pull request – See GITHUB

Recipe – The means by which an item is *crafted* in Minecraft play. It generally refers to the list of components that will be needed, how they are to be arranged physically on the crafting *grid* (a part of the GUI), and what sort of special device (if any) is needed to carry out the item creation. For example, a piece of *ladder* (the crafting product) can be crafted on a *crafting table* (the device) by arranging *seven wooden sticks* (the components) on the grid in an 'H' pattern (the physical arrangement, or shape, of the recipe). See Chapter 1, Figure 1-2 (page 5).

Recipe manager – A type of Minecraft mod that provides a reference to players, in which they can look up the required RECIPE for crafting any item. Many recipe managers also provide a reverse-lookup function for viewing all of the recipes in which an item is used as a component. Recipe managers help players deal with the otherwise-overwhelming proliferation of hundreds or thousands of new block types that might be present in a typical MODPACK. They are particularly useful in cases where familiar recipes have been altered by CRAFTTWEAKER. Just Enough Items (JEI) is an example of a recipe manager (see Section 6.4).

Redstone – A material in Minecraft used to build electric circuits. Also refers to the general system of building such circuits. Certain redstone components behave like transistors, allowing one to build simple automated logic systems, calculating machines, and even rudimentary computers.

Reflection (Java) – A set of tools provided by the core Java API that allow Java objects to examine themselves, extracting (and potentially altering) information about their own structures.

Resource Pack – A bundle of data assets used to modify the look and feel of Minecraft. This includes replacing graphics, such as block textures, with alternate versions, and swapping out audio files.

Save file – The collection of digital files on disk that correspond to a Minecraft WORLD. Despite the use of the singular noun form, a single Minecraft world actually consists of multiple separate digital files, stored together in a directory/folder.

Server – In multiplayer Minecraft play, the server is the program that handles all of the core game logic, and communicates what is going on to the players' CLIENTS. In single-player contexts, the Minecraft program is still internally divided into server and client tasks.

Skyblock – A Minecraft MINI-GAME in which the player spawns on a small, isolated dirt island in the sky, and has to carefully manage limited resources and restricted space. The single-player version can be made using an external map editor to create the terrain setup. The save-game files can then be shared, with the recipient dropping them into the appropriate folder to enable play in vanilla Minecraft. In the multiplayer version, a new modded rule is required to override Minecraft's normal random player-spawn locations and ensure that players are properly placed on the islands.

SMP – Survival Multiplayer, an expression used by Minecraft players to describe playing on a multiplayer SERVER in SURVIVAL MODE

Spawn – As a verb, the action by which an entity appears in the Minecraft world. For instance, when a new monster appears in a dark corner of a cavern, it is said to have *spawned*. The player also spawns in the Minecraft world when starting out, and after each death (in these latter instances, it is called *respawning*). Minecraft players also often use the term "spawn" to refer to the physical area within a Minecraft world at which newly-arrived and resurrected players will appear.

Spleef – A competitive elimination-based MINI-GAME in Minecraft that is played on an arena made out of a thin layer of some easy-to-break block, such as wool (which can be readily broken with bare hands) or dirt (which is quickly removed with a shovel). Players are not permitted to strike each other directly, but attempt to eliminate their opponents by literally digging the ground out from beneath their feet. Those who fall through the floor are out of the game (it is not uncommon for there to be a pool of lava below to catch them), and the last player standing is the winner. As the game progresses, it becomes increasingly challenging to stay alive, since more and more of the floor is missing.

SSP – Survival Single Player, referring to playing on one's own in a local, single-player world, in SURVIVAL MODE.

Survival Mode – A GAME MODE in which the player has to harvest raw materials from the world in order to build or craft anything. Furthermore, the player must eat food regularly, and can be hurt or killed by falling too far, drowning, or getting incinerated. If the DIFFICULTY is set higher than Peaceful, the player will also have to reckon with hostile monsters, creating the need to make safe, well-lit shelters in which to hide during the dark nights.

Texture Pack – What RESOURCE PACKS were called when they were only capable of altering textures, prior to Minecraft 1.6.1.

Tick – A fundamental unit of processing time in Minecraft, corresponding to one full iteration of the game program's core loop. During each tick, the positions of moving objects are updated, the status of ongoing machine processes (e.g. cooking food) progresses by some increment, and user inputs are processed.

Ticks per second (TPS) - The number of ticks that the Minecraft program is able to complete in one second, used as a general measure of the performance of the server process. The standard is for one tick to last for one-twentieth of a second (for a TPS of 20). However, if the game has too many complex tasks to perform in each tick (e.g. too many MOBS to carry out AI decision-making processes on), ticks may end up taking longer. Players may experience this as visible lag.

Tile entity – See BLOCK ENTITY.

Vanilla – Refers to unmodified Minecraft (any edition or version). When using modded Minecraft, the term also refers to those specific features or modules present in-game that are provided by the vanilla product and have not been altered (e.g. if new mineral ores such as copper and uranium are added to the world by a mod, the original Minecraft ores like coal, iron, and gold—still present in-game—can be identified collectively as "vanilla ores").

Version – Refers to a specific Minecraft release, a stage in the Minecraft update chain. New versions tend to contain new features, and sometimes change the way previously-existing features work. Distinct from GAME MODE and EDITION (each edition has multiple, consecutively-released versions, as shown in Figure 1-3 on page 10).

Void – An empty region that occupies the space above (the upper void) and below (the lower void) the wafer-like BLOCK(1) grid of a Minecraft world. See Subsection 6.3.2.

World – The virtual space in which Minecraft gameplay occurs. In multiplayer Minecraft, the world persists even after all players have logged out, as long as the SERVER program is running. In single-player, the world is saved to disk when the player exits the game, and can be loaded and resumed from the same state at a later time. Minecraft players may maintain several separate worlds concurrently, pursuing different gameplay goals in each. Synonymous with MAP. May also refer to the SAVE FILE(S), the digital manifestations (stored on disk) of the data associated with the world.

BIBLIOGRAPHY

Note: Where applicable, additional URLs for archived versions of web-based resources are provided in italic text. Articles hosted on an academic journal's website, resources with an existing DOI or database presence, video files, and downloadable documents (e.g. PDFs) are excluded from archiving. The archives are hosted by the Internet Archive Wayback Machine. https://web.archive.org/

Aarseth, E. (1997). *Cybertext: Perspectives on ergodic literature*. Baltimore, MD: Johns Hopkins UP.

Aarseth, E. (2001). Computer game studies, year one. *Game Studies, 1*(1). http://www.gamestudies.org/0101/editorial.html

Abrams, S. S. (2017). Emotionally crafted experiences: Layering literacies in Minecraft. *The reading teacher, 70*(4), 501-506. https://doi.org/10.1002/trtr.1515

Adorno, T. W. & Horkheimer, M. (2002). *Dialectic of enlightenment*. (E. Jephcott, Trans.) Stanford, CA: Stanford University Press. (Original work published 1944)

Anderson, C. A, & Ford, C. M. (1986). Affect of the game player: Short-term effects of highly and mildly aggressive video games. *Personality and Social Psychology Bulletin, 12*(4), 390-402. https://doi.org/10.1177/0146167286124002

Apperley, T. (2015). Glitch sorting: Minecraft, curation and the postdigital. In Berry, D.M., & Dieter, M. (Eds.), *Postdigital aesthetics: Art, computation and design* (pp. 232-244). Basingstoke, UK: Palgrave-MacMillan.

Ashton, D. (2010). Player, student, designer: Games design students and changing relationships with games. *Games and Culture, 5*(3), 256-277. https://doi.org/10.1177/1555412009359766

Banks, J. (2013). *Co-creating video games*. London, UK: Bloomsbury Academic.

Benkler, Y. (2006). *The wealth of networks: How social production transforms markets and freedom*. New Haven, CT: Yale UP. Available (Creative Commons Attribution Noncommercial Sharealike license): http://www.benkler.org/Benkler_Wealth_Of_Networks.pdf

Blumler, J. G., & Katz, E. (1975). *The uses of mass communications: Current perspectives on gratifications research*. Beverly Hills, CA: SAGE.

Boellstorff, T. (2008). *Coming of age in Second Life*. Princeton, NJ: Princeton University Press.

Bogost, I. (2005). Procedural literacy: problem solving with programming, systems, and play. In *Journal of media literacy*, 52(1-2). Retrieved from: http://bogost.com/writing/procedural_literacy/ *https://web.archive.org/web/20160127050226/http://bogost.com/writing/procedural_literacy/*

Bogost, I. (2007). *Persuasive games: The expressive power of videogames*. Cambridge, MA: MIT Press.

Bogost, I. & Montfort, N. (2009). *Platform studies: Frequently questioned answers*. Proceedings, *Digital Arts and Culture, December 12-15, 2009*. Irvine, CA. http://bogost.com/downloads/bogost_montfort_dac_2009.pdf

Bourdieu, P. (1986). The forms of capital. In J. Richardson (Ed.), *Handbook of theory and research for the sociology of education* (pp. 241-258). Westport, CT: Greenwood.

Brackin, A. L. (2014). Building a case for the authenticity vs. validity model of videogame design. In N. Gerrelts (Ed.), *Understanding Minecraft: essays on play, community, and possibilities* (pp. 191-202). Jefferson, NC: McFarland.

Brand, J. E., de Byl, P., Knight, S.J., & Hooper, J. (2014). Mining constructivism in the university: The case of creative mode. In N. Gerrelts (Ed.), *Understanding Minecraft: Essays on play, community and possibilities.* Jefferson, NC: McFarland.

Bruckman, A. (1995). Cyberspace is not Disneyland. Retrieved from: https://www.cc.gatech.edu/~asb/papers/getty/disneyland.html *https://web.archive.org/web/20160114130159/https://www.cc.gatech.edu/~asb/papers/getty/disneyland.html*

Bull, I.R. (2014). Just Steve: Conventions of gender on the virtual frontier. In N. Gerrelts (Ed.), *Understanding Minecraft: essays on play, community, and possibilities* (pp. 88-105). Jefferson, NC: McFarland.

Burawoy, M. (1998). The extended case method. *Sociological Theory 16*(1), 5-33. https://doi.org/10.1111/0735-2751.00040

Bush, V. (1995). As we may think. *Journal of Electronic Publishing, 1*(2). https://doi.org/10.3998/3336451.0001.101 (Reprinted from *Atlantic Monthly 176*(1), 101-108.) (Original work published 1945)

Caillois, R. (1961). *Man, play & games*. (M. Barash, Trans.) New York, NY: The Free Press.

Callaghan, N. (2016). Investigating the role of Minecraft in educational learning environments. *Educational Media International, 53*(4), 244-260. http://dx.doi.org/10.1080/09523987.2016.1254877

Canossa, A. (2012). Give me a reason to dig: Qualitative associations between player behaviour in Minecraft and live motives. In *Proceedings of the International Conference on the Foundations of Digital Games* (FDG 2012). ACM (pp. 282.283). New York, NY.

Champion, E. (2012a). Mod, mod, glorious mod. In E. Champion (Ed.) *Game mods: Design, theory and criticism*. Pittsburgh, PA: ETC Press, pp. 9-26. Available: http://www.etc.cmu.edu/etcpress/content/game-mods

Champion, E. (2012b). Teaching mods with class. In E. Champion (Ed.) *Game mods: Design, theory and criticism*. Pittsburgh, PA: ETC Press, pp. 113-148. Available: http://www.etc.cmu.edu/etcpress/content/game-mods

Christiansen, P. (2012). Between a mod and a hard place. In E. Champion (Ed.). *Game mods: Design, theory and criticism* (pp. 27-49). Pittsburgh, PA: ETC Press. Available: http://www.etc.cmu.edu/etcpress/content/game-mods

Christiansen, P. (2014). Players, modders and hackers. In N. Gerrelts (Ed.), *Understanding Minecraft: essays on play, community, and possibilities* (pp. 23-37). Jefferson, NC: McFarland.

Consalvo, M. (2007). *Cheating: Gaining advantage in videogames* [Amazon Kindle Edition]. Cambridge, MA: MIT Press.

Corbin, J., & Strauss, A. (2015). Basics of qualitative research: Techniques and procedures for developing grounded theory (4th ed.). Thousand Oaks, CA: SAGE.

Critical Art Ensemble. (2003). Nomadic power and cultural resistance. In N. Wardrip-Fruin & N. Montfort (Eds.), *The new media reader*. Cambridge, MA: MIT Press.

De Certeau, M. (1984). *The practice of everyday life* [Amazon Kindle Edition]. (S. Rendall, Trans.) Berkeley, CA: University of California Press.

Dezuanni, M. (2018). Minecraft and children's digital making: Implications for media literacy education. *Learning, Media and Technology, 43*(3), 236-249. https://doi.org/10.1080/17439884.2018.1472607

Dill, K, & Dill, J. (1998). Video game violence: A review of the empirical literature. *Aggression and violent behavior, 3*(4), 407-428. https://doi.org/10.1016/S1359-1789(97)00001-3

Duncan, S. C. (2011). Minecraft, beyond construction and survival. *Well Played: a journal on video games, value and meaning, 1*(1), 1-22. Retrieved from: http://dl.acm.org/citation.cfm?id=2207097

Dyer-Witheford, N. & G. de Peuter. (2009). *Games of Empire: Global capitalism and video games*. Minneapolis: University of Minnesota Press.

Elliot, D. (2018). A Minecraft-based response to 'new literacies' in the Middle Years. *Literacy Learning: The Middle Years, 26*(2), 22-24. Retrieved from EBSCOhost.

Ellison, T. L., & Evans, J. N. (2016). Minecraft, teachers, parents, and learning: What they need to know and understand. *School Community Journal, 26*(2). Retrieved from http://www.schoolcommunitynetwork.org/SCJ.aspx

Emerson, R. M., Fretz, R.I. & Shaw, L.L. (1995). *Writing ethnographic fieldnotes*. Chicago, IL: University of Chicago press.

Fanning, C., & Mir, R. (2014). Teaching tools: Progressive pedagogy and the history of construction play. In N. Gerrelts (Ed.), *Understanding Minecraft: essays on play, community, and possibilities* (pp. 38-56). Jefferson, NC: McFarland.

Fassbender, E. (2012). Use of "The Elder Scrolls Construction Set" to create a virtual history lesson. In E. Champion (Ed.), *Game mods: Design, theory and criticism*. Pittsburgh, PA: ETC Press, pp. 113-148. Available: http://www.etc.cmu.edu/etcpress/content/game-mods

Fiske, J. (2001). Intertextuality. In C. L. Harrington & D. D. Bielby (Eds.), *Popular culture: production and consumption* (pp. 219-233). Malden, MA: Blackwell. (Original work published 1987)

Frasca, G. (1999). Ludology meets narratology: Similitude and differences between (video)games and narrative. Retrieved from: http://www.ludology.org/articles/ludology.htm *https://web.archive.org/web/20160526211049/http://www.ludology.org/articles/ludology.htm*

Fuchs, L. H. (1990). *The American kaleidoscope: Race, ethnicity, and civic culture.* Middletown, CT: Wesleyan UP.

Gallagher, S., Jong, C., & Sinervo, K. A. (2017). Who wrote the Elder Scrolls? Modders, developers, and the mythology of Bethesda Softworks. *Loading… The Journal of the Canadian Game Studies Association, 10*(16), 32-52. http://journals.sfu.ca/loading/index.php/loading/article/view/169

Gee, J. P., & Hayes, E. (2009). "No quitting without saving after bad events": Gaming paradigms and learning in The Sims. *International Journal of Learning and Media, 1*(3), 49-65. https://doi.org/10.1162/ijlm_a_00024

Geertz, C. (1988). Thick description: Toward an interpretive theory of culture. In P. Bohannan & M. Glazer (Eds.), *High points in anthropology* (2nd ed.). New York: NY: McGraw-Hill. (Original work published 1973)

Genette, G. (1997). *Paratexts: Thresholds of interpretation* (J. E. Lewin, trans.). Cambridge, UK: Cambridge University Press.

Gerrelts, N. (2014). Introduction: Why Minecraft matters. In N. Gerrelts (Ed.), *Understanding Minecraft: essays on play, community, and possibilities* (pp. 1-5). Jefferson, NC: McFarland.

Glaser, B. G., & Strauss, A. L. (1967). The discovery of grounded theory: strategies for qualitative research. Chicago, IL: Aldine.

Gluckman, M. (2006). Ethnographic data in British social anthropology. In T. M. S. Evens & D. Handelman (Eds.), *The Manchester School: Practice and ethnographic praxis in anthropology* (pp. 13-22). New York, NY: Berghahn Books (Original work published 1961)

Goetz, C. (2012). Tether and accretions: Fantasy as form in videogames. *Games and Culture, 7*(6), 419-440. https://doi.org/10.1177/1555412012466288

Golding, D. (2013). To configure or navigate? On textual frames. In Z. Waggoner (Ed.), *Terms of play: Essays on words that matter in videogame theory*. Jefferson, NC: McFarland.

Goldstein, J. (2005). Violent video games. In J. Raessens & J. Goldstein (Eds.), *Handbook of computer game studies* (pp. 341-353). Cambridge, MA: MIT Press.

Gu, J. (2014). A craft to call mine: Creative appropriations of Minecraft in YouTube animations. In N. Garrelts (Ed.), *Understanding Minecraft: essays on play, community, and possibilities* (pp. 132-147). Jefferson, NC: McFarland.

Gunkel, D. J., & Gunkel, A. H. Terra Nova 2.0 – The new world of MMORPGs. *Critical studies in media and communication, 26*(2), 104-127. https://doi.org/10.1080/15295030902860195

Gupta, A., & Gupta, A. (2015). *Minecraft modding with Forge*. Sebastopol, CA: O'Reilly Media.

Hall, S. (2001). Encoding/decoding. In C. L. Harrington & D. D. Bielby (Eds.), *Popular culture: production and consumption* (pp. 123-132). Malden, MA: Blackwell. (Original work published 1980)

Hanghøj, T., & Hautopp, H. (2016). Teachers' pedagogical approaches to teaching with Minecraft. In *Proceedings of the 10th European Conference on Games Based Learning 6-7 October 2016* (pp. 265-272). Paisley, UK: Academic Conferences and Publishing International.

Hayes, E. R., & Gee, J. P. (2010). No Selling the Genie Lamp: A Game Literacy Practice in The Sims. E-Learning and Digital Media, 7(1), 67–78. https://doi.org/10.2304/elea.2010.7.1.67

Himanen, P. (2001). *The hackers ethic and the spirit of the information age* [Amazon Kindle Edition]. New York: Random House.

Hine, C. (2000). *Virtual ethnography*. London: SAGE Publications.

Hine, C. (2015). *Ethnography for the internet: Embedded, embodied, and everyday.* London, UK: Bloomsbury Academic.

Holt, R. (2004). Dialogue on the Internet: Language, civic identity, and computer-mediated communication. Westport, CT: Praeger.

Jenkins, H. (1992). *Textual poachers*. New York, NY: Routledge.

Jenkins, H. (2002). Interactive audiences?: The "collective intelligence" of media fans. In D. Harries (Ed.), *The new media book*. London: British Film Institute.

Jenkins, H. (2006). *Convergence culture: Where old and new media collide.* New York, NY: NYU Press.

Jenkins, H., & Fuller, M. (1995). Nintendo and new world travel writing: A dialogue. In S. G. Jones (Ed.), *Cybersociety: Computer-mediated communication and community* (pp. 57-72). Thousand Oaks, CA: SAGE.

Joubert, R. (2009). A (really) brief history of game modding. In *NAG online.* [Electronic version]. Retrieved from https://www.nag.co.za/2009/03/19/a-really-brief-history-of-game-modding/
http://web.archive.org/web/20161011031518/https://www.nag.co.za/2009/03/19/a-really-brief-history-of-game-modding/

Juul, J. (2001). Games telling stories? A brief note on games and narratives. *Game Studies, 1*(1). http://www.gamestudies.org/0101/juul-gts/

Juul, J. (2005). *Half-Real: Video games between real rules and fictional worlds*. Cambridge, MA: MIT Press.

Kelland, M. (2011). From game mod to low-budget film: The evolution of machinima. In H. Lowood & M. Nitsche (Eds.), *The machinima reader* (23-36). Cambridge, MA: MIT Press.

Keogh, B. (2013). When game over means game over: Using permanent death to craft living stories in Minecraft. In *Proceedings of the 9th Australasian Conference on Interactive Entertainment: Matters of Life and Death*, Melbourne, Australia. 20:1-20:6. https://doi.org/10.1145/2513002.2513572

Kirsh, S. J. (1998). Seeing the world through Mortal Kombat-colored glasses: Violent video games and the development of short-term hostile attribution bias. *Childhood, 5*(2), 177-184. https://doi.org/10.1177/0907568298005002005

Konzack, L. (2002). Computer game criticism: A method for computer game analysis. F. Mäyrä (ed.), *CGDC Conference Proceedings*, pp. 89-100. Tampere, Finland: Tampere UP. http://www.digra.org/wp-content/uploads/digital-library/05164.32231.pdf

Kow, Y. M. & Nardi, B. (2010). Who owns the mods? *First Monday, 15*(5). https://doi.org/10.5210/fm.v15i5.2971

Kringiel, D. (2011). Machinima and modding: Pedagogic means for enhancing computer game literacy. In H. Lowood & M. Nitsche (Eds.), *The machinima reader* (257-273). Cambridge, MA: MIT Press.

Kücklich, J. (2005). Precarious playbour: Modders and the digital games industry. *The Fibreculture Journal* 5. Retrieved from: http://five.fibreculturejournal.org/fcj-025-precarious-playbour-modders-and-the-digital-games-industry/

Kuhn, J. (2018). Minecraft: Educational Edition. *Calico Journal, 35*(2), 214-223. https://doi.org/10.1558/cj.34600

Lastowka, G. (2013). Minecraft as web 2.0: Amateur creativity and digital games. In Hunter, D., Lobato, R., Richardson, M., & Thomas, J. (Eds.), *Amateur media: Social, cultural and legal perspectives* (pp. 153-170). New York, NY: Routledge.

Leavitt, A. (2013). Crafting Minecraft: Negotiating creative produsage-driven participation in an evolving cultural artifact. *Selected Papers of Internet Research*. Retrieved from http://spir.aoir.org/index.php/spir/article/view/798

Lessig, L. (2008). *Remix: Making art and commerce thrive in the hybrid economy*. London, UK: Bloomsbury.

Lindstrom, L. (1993). *Cargo cult: Strange stories of desire from Melanesia and beyond*. Honolulu, HI: Univ. Hawaii Press.

Lunenfeld, P. (1999). Unfinished business. In P. Lunenfeld (ed.), *The digital dialectic: New essays on new media.* Cambridge, MA: MIT Press.

MacCallum-Stewart, E. (2014). "Someone off the YouTubez". In N. Gerrelts (Ed.), *Understanding Minecraft: essays on play, community, and possibilities* (pp. 148-159). Jefferson, NC: McFarland.

Macron, N., & Faulker, J. (2016). Exploring Minecraft as a pedagogy to motivate girls' literacy practices in the secondary English classroom. *English in Australia, 51*(1), 63-69. Retrieved from EBSCOhost.

Malaby, T. (2006). Parlaying value: Capital in and beyond virtual worlds. *Games and Culture, 1*(2), 141-162. https://doi.org/10.1177/1555412006286688

Malaby, T. (2007). Beyond play: A new approach to games. *Games and Culture, 2*(2), 95-113. https://doi.org/10.1177/1555412007299434

Malaby, T. (2009). *Making virtual worlds: Linden Lab and Second Life*. Ithaca, NY: Cornell UP.

Mateas, M. & Wardrip-Fruin, N. (2009). Defining operational logics. In *Breaking new ground: Innovation in games, play, practice, and theory – Proceedings of DiGRA 2009.* http://www.digra.org/wp-content/uploads/digital-library/09287.21197.pdf

Mavoa, J., Carter, M., & Gibbs, M. (2018). Children and Minecraft: A survey of children's digital play. *New Media & Society, 20*(9), 3283-3303. https://doi.org/10.1177/1461444817745320

McDonald, J. J. (2007). *American ethnic history: Themes and perspectives.* Edinburgh, UK: Edinburgh UP.

Mitchell, E. (1985). The dynamics of family interaction around home video games. *Marriage and family review, 8*(1-2), 121-135. https://doi.org/10.1300/J002v08n01_10

Montfort, N. (2006). Combat in context. *Game Studies, 6*(1). http://gamestudies.org/0601/articles/montfort

Morris, S. (2003). WADS, bots, and mods: Multiplayer FPS games as co-creative media. *DiGRA '03 – Proceedings of the 2003 DiGRA International Conference: Level Up, 2*. http://www.digra.org/digital-library/publications/wads-bots-and-mods-multiplayer-fps-games-as-co-creative-media/

Murphy, D. (2015). Tenuous freedom? Decoding the politics of Minecraft's development. Paper presented at *The Building Blocks of Life: A Minecraft Colloquium,* 13 February 2015, Concordia University, Montréal, QC.

Murray, S. (2004). "Celebrating the story the way it is": Cultural studies, corporate media and the contested utility of fandom. *Continuum: Journal of media & cultural studies, 18*(1).

Nardi, B. (2010). *My life as a night elf priest: An anthropological account of World of Warcraft.* Ann Arbor, MI: University of Michigan Press.

Nebel, S., Schneider, S., & Rey, G.D. (2016). Mining learning and crafting scientific experiments: A literature review on the use of Minecraft in education and research. *Educational Technology & Society, 19*(2), 355-366.

Nieborg, D.B. (2005). Am I a mod or not? An analysis of first person shooter modification culture. Paper presented at *Creative gamers seminar: Exploring participatory culture in gaming.* Tampere, Finland: University of Tampere. Retrieved from: http://www.gamespace.nl/content/DBNieborg2005_CreativeGamers.pdf

Nieborg, D. B., & van der Graaf, S. (2008). The mod industries? The industrial logic of nonmarket game production. *European Journal of Cultural Studies, 11*, 177-195.

Niemeyer, D.J., & Gerber, H.R. (2015). Maker culture and Minecraft: Implications for the future of learning. *Educational Media International, 52*(3), 216-226. https://doi.org/10.1080/09523987.2015.1075103

Otto, T. (2009). What happened to cargo cults? *Social analysis, 53*(1), 82-102. https://doi.org/10.3167/sa.2009.530106

Paul, C. A. (2011). Optimizing play: How theorycraft changes gameplay and design. *Game Studies*, *11*(2). http://gamestudies.org/1102/articles/paul

Pauls, K. (2017). *Playing in the digital sandbox: An exploration of social play behaviours in Minecraft*. Master's thesis, Concordia University, Montréal, Canada. https://spectrum.library.concordia.ca/982723/

Pearce, C. (2006). "Productive play: Game culture from the bottom up." *Games & Culture, 1*(1), 17-24.

Pearce, C. (2009). *Communities of play*. Cambridge, MA: MIT Press.

Pellicone, A., & Ahn, J. (2018). Building worlds: A connective ethnography of play in Minecraft. *Games and Culture, 13*(5), 440-458. https://doi.org/10.1177/1555412015622345

Petry, A. S. (2018). Playing in Minecraft: An exploratory study. *Revista Famecos 25*(1). https://doi.org/10.15448/1980-3729.2018.1.27156

Phillips, A. (2014). (Queer) algorithmic ecology: The great opening up of nature to all mobs. In N. Gerrelts (Ed.), *Understanding Minecraft: essays on play, community, and possibilities* (pp. 106-120). Jefferson, NC: McFarland.

Postigo, H. (2007). Of mods and modders: Chasing down the value of fan-based digital game modifications. *Games and Culture* 2(4), 300-313. DOI: 10.1177/1555412007307955

Provenzo, E. (1991). *Video kids: Making sense of Nintendo*. Cambridge, MA: Harvard UP.

Punday, D. (2015). *Computing as writing*. Minneapolis, MN: University of Minnesota Press.

Raessens, J. (2005). Computer games as participatory media culture. In J. Raessens & J. Goldstein (Eds.), *Handbook of Computer Game Studies*. Cambridge, MA: MIT Press.

Raymond, E. (2000). *The cathedral and the bazaar* (v. 3.0). Retrieved from http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/ *https://web.archive.org/web/20171220145837/http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/*

Redmond, D. (2014). The videogame commons remakes the transnational studio. In N. Gerrelts (Ed.), *Understanding Minecraft: essays on play, community, and possibilities* (pp. 7-22). Jefferson, NC: McFarland.

Ritzer, G. (1983). The "McDonaldization" of society. *Journal of American Culture, 6*(1), 100-107. https://doi.org/10.1111/j.1542-734X.1983.0601_ 100.x

Royce, W. (1970). Managing the development of large software systems. *Proceedings of IEEE WESCON*, August 1970, pp. 1–9. http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf

Salen, K., & Zimmerman, E. (2003). *Rules of play – Game design fundamentals*. Cambridge, MA: MIT Press.

Sanjek, D. (2001). "Don't have to DJ no more": Sampling and the "autonomous" creator. In C. L. Harrington & D. D. Bielby (Eds.), *Popular culture: production and consumption* (pp. 243-256). Malden, MA: Blackwell. (Original work published 1994)

Scacchi, W. (2010). Computer game mods, modders, modding and the mod scene. *First Monday*, *15*(5). https://doi.org/10.5210/fm.v15i5.2965

Scacchi, W. (2011). Modding as a basis for developing game systems. In *Games And Software Engineering, GAS'11*, 22 May 2011. Waikiki, Honolulu, HI: ACM. Retrieved from: https://www.ics.uci.edu/~wscacchi/Papers/New/Scacchi-GAS-Paper-March11.pdf

Schifter, C. C., & Cipollone, M. (2015). Constructivism vs. constructionism: Implications for Minecraft and classroom implementation. In P. Isaías, J.M. Spector, D. Ifenthaler, & D.G. Sampson (Eds.), *E-Learning systems, environments, and approaches* (pp. 213-227). Heidelberg, Germany: Springer.

Schlinsog, M. J. (2013). Endermen, creepers, and copyright: The bogeymen of user-generated content in Minecraft. *Tulane Journal of Technology and Intellectual Property, 16*, 185-206. Retrieved from EBSCOhost.

Schneier, J., & Taylor, N. (2018). Handcrafted gameworlds: Space-time biases in mobile Minecraft play. *New Media & Society, 20*(9), 3420-3436. https://doi.org/10.1177/1461444817749517

Seidman, I. (2006). *Interviewing as qualitative research* (3rd ed.). New York, NY: Teachers College Press.

Short, D. (2012). Teaching scientific concepts using a virtual world – Minecraft. *Teaching Science, 58*(3), 55-58. Retrieved from https://www.learntechlib.org/p/91796/

Sicart, M. (2009). *The ethics of computer games* [Amazon Kindle Edition]. Cambridge, MA: MIT Press.

Siegert, B. (2015). *Cultural techniques : Grids, filters, doors, and other articulations of the real* (G. Winthrop-Young, Trans.) New York, NY: Fordham University Press.

Sihvonen, T. (2009). *Players unleashed!: Modding The Sims and the culture of gaming.* Turku, Finland: Uniprint.

Sihvonen, T. (2011). *Players unleashed!: Modding The Sims and the culture of gaming*. Amsterdam: Amsterdam University Press.

Simon, B., & Wershler, D. (2018). Childhood's end (or, we have never been modern, except in Minecraft). *Cultural Politics, 14*(3), 289-303. https://doi.org/10.1215/17432197-709331

Simon, B., Wershler, D., & Watson, N. (2015, March 29). Block by block: Minecraft and the manufacture of expertise. Paper presented at 2015 Annual Conference of the Society for Cinema & Media Studies, Montréal, QC.

Smith, M., & Kollock, P. (Eds.) (1999). *Communities in cyberspace*. London, UK: Routledge.

Sotamaa, O. (2009). *The player's game: Toward understanding player production among computer game cultures*. (Doctoral dissertation, University of Tampere, Finland). Retrieved from http://tampub.uta.fi/handle/10024/66445

Sotamaa, O. (2010). When the game is not enough: Motivations and practices among computer game modding culture. *Games and Culture, 5*(3), 239-255. https://doi.org/10.1177/1555412009359765

Stevens, P. (1980). Play and work: A false dichotomy? In H. B. Schwartzman (Ed), *Play and culture: 1978 proceedings of the Association for the Anthropological Study of Play*. New York: Leisure Press.

Taylor, T.L. (2006). *Play between worlds: Exploring online game culture*. Cambridge, MA: MIT Press.

Thomét, M. (2014). Look what just happened: Communicating play in online communities. In N. Gerrelts (Ed.), *Understanding Minecraft: essays on play, community, and possibilities* (pp. 121-131). Jefferson, NC: McFarland.

Tremblay, A. J., Colangelo, J., & Brown, J. A. (2014). The craft of data mining: Minecraft and the constraints of play. In N. Gerrelts (Ed.), *Understanding Minecraft: essays on play, community, and possibilities* (pp. 76-87). Jefferson, NC: McFarland.

Turner, F. J. (1921). The significance of the frontier in American history. In F. J. Turner, *The frontier in American history* (pp. 1-38). Retrieved from: https://www.gutenberg.org/ebooks/22994

University of Surrey. (2013, July 2). "Christine Hine on online research methods." Retrieved from: http://youtu.be/No8RZOebhX8

Voiskounsky, A. E., Yermolova, T. D., Yagolkovskiy, S. R., & Khromova, V. M. (2017). Creativity in online gaming: Individual and dyadic performance in Minecraft. *Psychology in Russia: State of the Art, 10*(4), 144-161. https://doi.org/10.11621/pir.2017.0413

Wardrip-Fruin, N. (2009). *Expressive processing*. Cambridge, MA: MIT Press.

Wark, M. (2004). *A hacker manifesto.* Cambridge, MA: Harvard University Press.

Watson, N. (2012). *Game developing, the D'ni way: How Myst/Uru fans inherited the cultural legacy of a lost empire.* Master's thesis, Georgia Institute of Technology, Atlanta, Georgia. https://smartech.gatech.edu/handle/1853/44898

Watson, N. (2017). Procedural elaboration: How players decode Minecraft. *Loading… The Journal of the Canadian Game Studies Association, 10*(16), 75-86. http://journals.sfu.ca/loading/index.php/loading/article/view/181

Webb, S. (2001). Avatar culture: Narrative, power and identity in virtual world environments. *Information, Communication, & Society, 4*(4), 560-594. https://doi.org/10.1080/13691180110097012

Weber, M. (1930). *The Protestant ethic and the spirit of capitalism* (T. Parsons, tr.). New York, NY: Scribner. Available: https://archive.org/details/protestantethics00webe

Weisert, C. (2003, Feb 8). *There's no such thing as the Waterfall approach! (and there never was).* Retrieved from: http://www.idinews.com/waterfall.html *https://web.archive.org/web/20180217091854/http://www.idinews.com/waterfall.html*

Wershler, D. (2015). Minecraft and the management of light. Paper presented at *The Building Blocks of Life: A Minecraft Colloquium,* 13 February 2015, Concordia University, Montréal, QC.

Willett, R. (2018). "Microsoft bought Minecraft… who knows what's going to happen?!" A sociocultural analysis of 8-9-year-olds' understanding of commercial online gaming industries. *Learning, Media and Technology, 43*(1), 101-116. https://doi.org/10.1080/17439884.2016.1194296

Williams, D. (2003). The video game lightning rod. *Information, communication & society, 6*(4), 523-550. https://doi.org/10.1080/1369118032000163240

Yang, R. (2012, September 19). A people's history of the FPS [Blog post]. *Rock, Paper, Shotgun*. Retrieved from: https://www.rockpapershotgun.com/2012/09/19/a-peoples-history-of-the-fps-part-1-the-wad/ *https://web.archive.org/web/20150716054040/https://www.rockpapershotgun.com/2012/09/19/a-peoples-history-of-the-fps-part-1-the-wad/*

Yee, N., Ellis, J., & Ducheneault, N. (2008). The tyranny of embodiment. *Artifact, 2*(2), 88-93. https://doi.org/10.1080/17493460903020224

OTHER REFERENCES

Agile Alliance (2001). Manifesto for agile software development. https://www.agilealliance.org/agile101/the-agile-manifesto/ *https://web.archive.org/web/20170909214648/https://www.agilealliance.org/agile101/the-agile-manifesto/*

Almasy, S. (2017, December 21). 'Slenderman' stabbing: Teen committed to mental institution. *CNN*. Retrieved from: https://www.cnn.com/2017/12/21/us/slenderman-teen-sentence/index.html *https://web.archive.org/web/20181103185445/https://www.cnn.com/2017/12/21/us/slenderman-teen-sentence/index.html*

Chatfield, T. (2012). Ending an endless game: an interview with Julian Gough, author of Minecraft's epic finale. Retrieved from: https://boingboing.net/2012/01/09/ending-an-endless-game-an-int.html *https://web.archive.org/web/20120111043609/https://boingboing.net/2012/01/09/ending-an-endless-game-an-int.html*

citricsquid. (2017, September 21). An update to the Minecraft Forum category structure [Online forum comment]. Retrieved from: https://www.minecraftforum.net/forums/forums/forum-discussion-info/2860829-an-update-to-the-minecraft-forum-category *http://web.archive.org/web/20170923174241/https://www.minecraftforum.net/forums/forums/forum-discussion-info/2860829-an-update-to-the-minecraft-forum-category*

Dent, S. (2019, May 17). 'Minecraft' has sold 176 million copies worldwide. *Engadget*. Retrieved from: https://www.engadget.com/2019/05/17/minecraft-has-sold-176-million-copies-worldwide/ *https://web.archive.org/web/20190517145017/https://www.engadget.com/2019/05/17/minecraft-has-sold-176-million-copies-worldwide/*

Newman, J. (2017, April 10). Microsoft is ready to cash in on Minecraft mods. *Fast Company*. Retrieved from https://www.fastcompany.com/4034389/microsoft-is-ready-to-cash-in-on-minecraft-mods *https://web.archive.org/web/20190317221154/https://www.fastcompany.com/4034389/microsoft-is-ready-to-cash-in-on-minecraft-mods*

Orland, K. (2011, April 6). Minecraft draws over $33 million in revenue from 1.8M paying customers. *Gamasutra*. Retrieved from: http://www.gamasutra.com/view/news/33961/Minecraft_Draws_Over_33_Million_In_Revenue_From_18M_Paying_Customers.php *https://web.archive.org/web/20110408063046/http://www.gamasutra.com/view/news/33961/Minecraft_Draws_Over_33_Million_In_Revenue_From_18M_Paying_Customers.php*

Thompson, P. (2013, September 30). Florida boy, 9, brought weapons to school to act out video game Minecraft. *Daily Mail Online*. Retrieved from: http://www.dailymail.co.uk/news/article-2439462/Florida-boy-9-brought-weapons-school-act-video-game-Minecraft.html *https://web.archive.org/web/20151014042107/http://www.dailymail.co.uk/news/article-2439462/Florida-boy-9-brought-weapons-school-act-video-game-Minecraft.html*