Improving Video Game Balance Testing Using Autonomous Agents

Cristiano Politowski

A Thesis
in
The Department
of
Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of
Doctor of Philosophy (Software Engineering) at
Concordia University
Montréal, Québec, Canada

July 2022

CONCORDIA UNIVERSITY
School of Graduate Studies

This is to certify that the thesis prepared

By: **Cristiano Politowski**

Entitled: **Improving Video Game Balance Testing Using Autonomous Agents**

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

Doctor of Philosophy (Software Engineering)

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

Signed by the final examining committee:

_____ Chair
Dr. Bruno Lee

_____ External Examiner
Dr. Antonio Bucchiarone

_____ External to Program
Dr. Wahab Hamou-Lhadj

_____ Examiner
Dr. Emad Shihab

_____ Examiner
Dr. Juergen Rilling

_____ Thesis Supervisor
Dr. Yann-Gaël Guéhéneuc

_____ Co-supervisor
Dr. Fabio Petrillo

Approved by _____
Dr. Leila Kosseim, Graduate Program Director

_____ _____
Dr. Mourad Debbabi, Dean
Gina Cody School of Engineering and Computer Science

Abstract

Improving Video Game Balance Testing Using Autonomous Agents

Cristiano Politowski, Ph.D.
Concordia University, 2022

As the complexity and scope of game development increase, playtesting (game testing) remains an essential activity to ensure the quality of video games. Yet, the manual, ad-hoc nature of game testing gives space for improvements in the process. In this thesis, we research, design, and implement an approach to enhance game testing to balance video games. Instead of manually testing games, we present an automated approach with autonomous agents to aid game developers to assess the game's balance. We describe the process of training the agents, playing the game, and assessing the game balance using game attributes. We validated our testing process with two platform games. We conclude that the use of autonomous agents to test games is faster than the manual feedback loop and provides a viable solution for game balancing, showing spikes in difficulty between game versions and issues with the game design.

Acknowledgments

Doing a Ph.D. is not easy, believe me. It affects your mental health in a very distinguished way. Impostor syndrome, anxiety, depression, you name it. If you do not have a strong foundation you will perish[1]. Family, friends, a good work environment, and a good relationship with your supervisors is essential to endure the journey. Luckily, I had them.

I can stress enough how well my supervisor Professor Yann-Gael treated me. Our relationship was always the best and his mentoring was on point. I learned a lot about many things, not only science and academia. Professor Yann is one of those special human beings that helps you keep going forward. Thank you very much for everything Yann (including the *divided by 2 + 7*) and sorry for the countless times I crossed the line.

I would like to thank all the committee members for accepting to review my thesis. Also, thanks to my lab mates that helped me give important feedback and listened to all my presentation about video games. In special to Gabriel, that helped me directly with the papers.

A special thanks to all the special people that helped me in some way or another. Sophie, my roommate during all this time, never argued or fought with me. Frank, my extremely intelligent friend needs to finish that book. Farzaneh for being bright and motivating me to work harder.

Agradecimentos (pt-br):

Eu agradeço a minha mãe, dona Ivone, claro. Eu deixei ela sozinha e vim pro Canadá fazer o doutorado. Como eu sou filho único, sei que isso a deixou muito triste. Ainda mais passando por toda a pandemia, ela ficando isolada e nós sem saber o dia de amanhã. No fim deu tudo certo. A mãe sempre me apoiou nas decisões sem nunca nem questionar. Talvez seja por isso que sou um pouco mimado. Um beijo pra melhor mãe do mundo.

Outro fator importante, acredite ou não, sao os amigos. Quando vim pro Canada, estava sozinho, deixei todos que conhecia no Brasil. Felizmente (ou não, é debatível), fiz algumas amizades que me ajudaram durante esses quatro anos em Montreal. Em especial: Rodrigo, o metódico de bom coração; Plinio, o caótico "não tem tempo ruim"; Leandro, o bardo; Tainá, a única pessoa do mundo que odeia chocolate; Dani, a "me respeita"; e Nat, a sertaneja.

Por fim, mas não menos importante, um abraço especial ao meu co-orientador Fabio Petrillo.

[1] Oh, finally I understood the "publish or perish" thing.

Nós trabalhamos juntos desde o meu mestrado, por volta de 2016. Sem ele meu mestrado teria pego outro rumo e eu não estaria em Montreal hoje, terminando o doutorado. O Fabio certamente merece um cafezinho.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

"Because of the nature of Moore's law, anything that an extremely clever graphics programmer can do at one point can be replicated by a merely competent programmer some number of years later."

JOHN CARMACK, LEAD PROGRAMMER OF GAMES LIKE DOOM AND QUAKE

1.1 Context

For decades, video games have been a joyful hobby for many people (3 billion players in 2021[2]) around the world [5]. Also, the competitive nature of some game types created the E-Sport industry, filling big arenas and having millions in the prize pool[3]. These are some of the reasons why the game industry surpassed, in revenue, the movie and music industries combined [6].

While the video game market is expanding, this does not mean every game developer partakes in its growth. The game market is large. Platforms such as Steam[4] receive hundreds of new video games every month[5]. Standing out from this crowd and making a profit, for a game, is a hard endeavour.

Making video games is also expensive. A former Playstation executive explained[6] that the cost of AAA video games, i.e., games produced by large companies, with large teams and large budgets, is doubling with each new console generation[7]. For example, the executive expects that every new

[2] https://tinyurl.com/mtryknes
[3] The Game Dota 2, for example, had a prize pool of almost U$35 million in 2019 just for one tournament.
[4] Steam is a video game digital distribution service for PC (https://store.steampowered.com/).
[5] https://steamdb.info/stats/releases/
[6] https://tinyurl.com/3hpb33zn
[7] Video game consoles are segmented into generations, grouping them when they share competitive market-space.

AAA[8] Playstation 5[9] game will cost U$ 200 million dollars, at least. Making AAA-games is a high-risk investment because the development takes years to be completed. This scenario skews the game industry towards safer alternatives to new games, like sequels.

Creating video games is hard. A recent example of the difficulties in developing a video game is the case of the game "No Man's Sky". Crowdfunded and produced by the small studio "Hello Games", it suffered from strong criticism for not delivering promised features [7], translating into consumers massively asking for a refund [10]. A similar situation also happened with large studios, as in the cases of "Aliens: Colonial Marines" [8] and "Marvel vs. Capcom Infinite"[11], which were expected to generate large profits but did not meet the expectations of the players and underperformed financially.

In the game industry, the first impression is of utmost importance; therefore, only high-quality games can expect to succeed. In December of 2020, the game "Cyberpunk 2077"[12] was released after seven years of development[13], hundreds of millions of dollars[14], multiple delays[15], and high expectations. Upon release, it immediately received strong negative criticism for its "buggy state" and, not long after, it was even removed from the PlayStation Store[16] because it was not delivering the expected experience to its users. It caused revenue loss and other collateral effects, like loss of prestige and goodwill for the studio CD Projekt Red[17].

Many observers in industry and academia ponder how a company with a reputation for quality games[18], like CD Projekt Red, with no shortage of money, time, skill, and experience could release a game with such poor quality. These observers, as well as personnel from the company itself, blame the scope, the management, and the lack of testing[19] for the poor quality of the game. According to the company CEO, Marcin Iwiński, "Every change and improvement needed to be tested, and as it turned out, our testing did not show a big part of the issues you experienced while playing the game"[20].

[8] AAA-studios are large game companies with a high budget to develop big-scope games, so-called AAA-games.

[9] As of 2022, Playstation 5 (https://www.playstation.com) is the latest version of the Sony game console.

[10] https://tinyurl.com/2p9d7r48

[11] https://tinyurl.com/2th4s67z

[12] https://www.cyberpunk.net/ca/en/

[13] https://www.youtube.com/watch?v=cGmWwFpNIHg

[14] https://tinyurl.com/2p8n4rxd

[15] https://tinyurl.com/2p84pjzd

[16] https://www.playstation.com/en-ca/cyberpunk-2077-refunds/

[17] https://en.cdprojektred.com/

[18] The previous game released by the same company, "The Witcher 3", was considered a great success (https://tinyurl.com/zaewye94).

[19] https://tinyurl.com/38dpvhc2

[20] https://tinyurl.com/jpvpmfyn

1.2 Video Game Testing

Like with any piece of software, to deliver high-quality games, game developers must test their games rigorously during their development. In traditional software development, tests are considered essential (unit, component, integration, or end-to-end tests), and so is their automation [9].

New games are challenging because the complexity of game development increases with their scopes. Bigger games need more work, time, people, and funding. Also, users (players) expect a bigger and better game with every new release. Yet, to achieve success in the saturated video game market, games must also be of quality. Thus, to succeed commercially, game developers must reduce expenses while keeping the high-quality levels in their games.

Different from traditional software, games rely on first impressions and first reviews. Many game development companies (also known as studios), after developing a game for years, receive a poor reception on release[21] from which they cannot recover because, no matter how many updates follow, the bad reviews will stay as so will the damage.

To avoid such a catastrophic scenario, game developers must polish and extensively test their games to improve their quality. No matter how fun and advanced a game is, bugs and balance issues can easily affect the users' experience. Therefore, similar to traditional software, testing should be an essential task in a successful game project. Yet, game development lacks traditional software testing practices as game development relies on manual human labour to assess the games[22].

Manual video game testing (commonly named playtesting) is the way developers control the quality of the game [10]. It consists of game testers interacting with the game during the development cycle to gauge the users' engagement and discover states that result in undesirable outcomes [11]. Game developers prefer playtesting over other techniques [12, 13] as it gives feedback to the team in form of "enjoyment heuristics" [14].

Usually, large companies have in-house Quality Assurance (QA) teams that perform game testing. Smaller companies either use outsourcing or let their developers playtest their games, like most independent developers (indie). No matter the company size, manual game testing does not scale to the size of the game. The bigger the game, the more testers are needed.

Playtesting also serves different purposes, including assessing game performance, verifying game completion, finding bugs, and *balancing the game*. Testing the game for balance is a subjective task, crucial for the user experience. It is adjusting the elements of the game until they deliver the experience that the developers want to offer to the players [1]. Simply put, it involves figuring out what numbers (parameters) to use in the game [15]. It demands understanding the subtle nuances between the game attributes and knowing which ones to alter, how much to alter

[21] https://tinyurl.com/357943t5
[22] https://tinyurl.com/yckpx4kb

them, and which ones to leave alone [1]. Balancing is hard because no two games are alike, and every game has many different attributes (for example, the strength of the attack, high of the jump, etc) that must be in balance. Finally, balancing testing is an ad-hoc manual testing process that is slow, hard to reproduce, and inefficient [16].

To ease and improve game testing, developers and researchers could use some form of automation for their tests. Recently, with the success of Machine Learning (ML) models to train autonomous agents to master video games, researchers and game development companies also began to use these models for game testing [17]. In a recent study [18], discussed in Chapter 5, we identified that, despite the advances in this field, most of the current solutions for game testing are lacking. For example, (1) the solutions focused on creating the agents to play the game but were oblivious to engineering aspects, like integrating these solutions with development tools; (2) the testing objectives are not clearly defined, sometimes with phrases like "it can be used to test the game"; (3) there is no oracle or it is made manually *after* the autonomous agents play the games; and (4) the source code (replication packages) are often not available.

Moreover, within our literature review on game testing (also in Chapter 5), we identify a few papers that work with balancing the game. Besides sharing the same problems discussed above, their solutions do not try to incorporate the agents into the game development process. Most game studios, especially the small ones, do not have the time or the budget to adopt complex and costly solutions like those, but they can benefit from a more feasible approach.

Thesis Statement

In this thesis, we research, design, and implement an approach to **enhance game testing to balance video games**. Instead of manually testing games (playtesting), we present an automated approach with autonomous agents to aid game developers assess the game's balance.

We do not want to replace human testers entirely because intelligent agents are not capable, yet, to perceive subtle details that professional game testers can. Thus, developers and testers could focus on the game features and enjoyment, resulting in games with overall better quality. This work is a step toward improving video game testing and, in the long term, we hope these ideas help disrupt the game development state-of-practice.

1.3 Thesis Outline

To answer our statement, we performed a series of studies. They are separated into chapters according to the Figure 1 and Table 1. Chapter 2 has the background information needed to better understand the rest of the thesis. Chapter 3 describes our first exploratory work where we searched

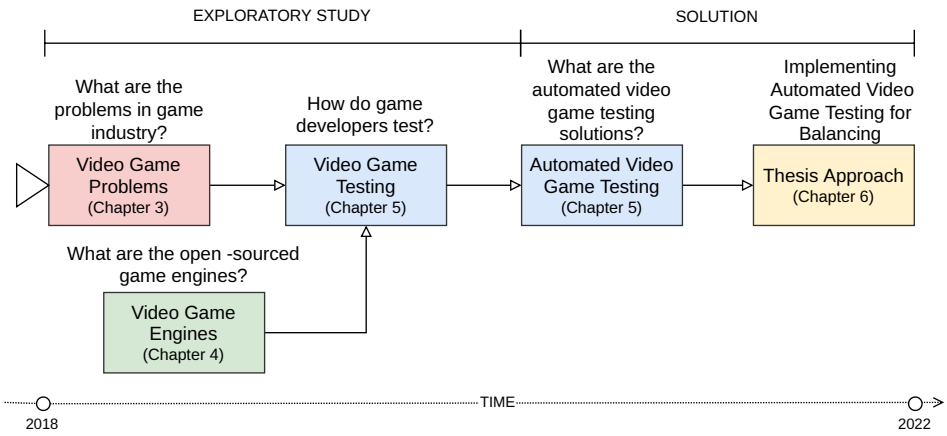


Figure 1: The Thesis Outline

for gaps in the game industry problems using gray literature (postmortems). Chapter 4 describes our second exploratory work where we analyzed the game engines, the main tool used by game developers. Chapter 5 narrows down the testing subject and investigated the game testing literature. Chapter 6 describes our approach and implementation of automated testing for balancing video games. Chapter 7 summarize and conclude the thesis.

Table 1: Contributions done during the Ph.D..

Subject	ID	Venue	Year	Paper title
Game Problems (Chapter 3)	[19]	Conference	2020	Dataset of Video Game Development Problems
	[20]	Journal	2021	Game Industry Problems: An Extensive Analysis of the Gray Literature
	[21]	Workshop	2022	Video Game Project Management Anti-patterns
Game Engines (Chapter 4)	[22]	Journal	2021	Are Game Engines Software Frameworks? A Three-Perspective Study
Game Testing (Chapter 5)	[23]	Conference	2021	A Survey of Video Game Testing
	[18]	Workshop	2022	Towards Automated Video Game Testing: Still a Long Way to Go
Automated Testing Games Balance (Chapter 6)	–	–	2022	Improving video game balance testing using autonomous agents
Misc.	[24]	Journal	2020	A Large Scale Empirical Study of the Impact of Spaghetti Code and Blob Anti-Patterns on Program Comprehension
	[25]	Journal	2021	What Skills Do IT Companies Look for in New Developers? A Study with Stack Overflow Jobs
	[21]	Workshop	2022	Video Game Project Management Anti-patterns

Chapter 2

Background

"A delayed game is eventually good, but a rushed game is forever bad."

SHIGERU MIYAMOTO, CREATOR OF SUPER MARIO AND LEGEND OF ZELDA

In this section we mainly describe the concepts of *software testing* and *video game testing*. We also discussed other key concepts necessary to better understand the thesis, like *postmortems*, *game engines*, and *software frameworks*.

2.1 Testing

There is no clear definition of what is video game testing and what it implies. Therefore we include technical testing, Quality Assurance (QA), and Game User Research (GUR) under the "umbrella" of video game testing. The Figure 2 summarize the discussion in this section.

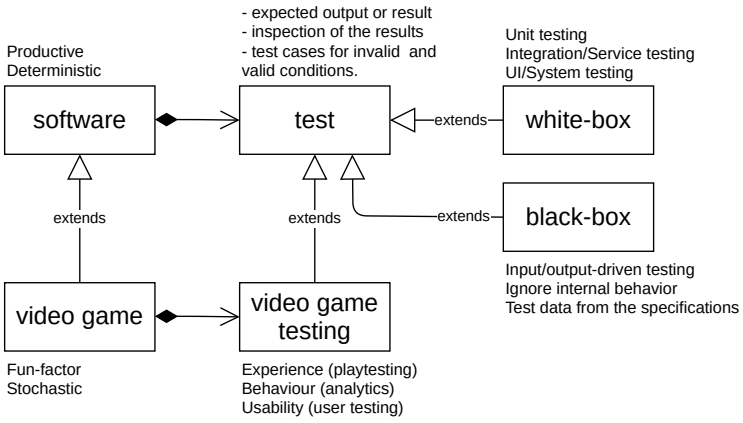


Figure 2: Software Testing & Video Game Testing summary.

2.1.1 Software Testing

Software testing is a domain that gains more importance as the software industry evolves. Faults and their corrections are among the main factors leading to budget overruns [26]. Testing accounts for more than 50% of the total costs of software development [27].

Software testing is part of the process to verify and improve the quality of a software [28], the System Under Test (SUT) [26]. Testing analysis can be static (based on source code) or dynamic (using SUT executions). The objectives of testing are to check if the SUT works; if not, to find the faults. Fault (or defects) refers to the *cause* of an error, which is the problematic *state* of a SUT that might cause it to not *behave* according to its specification, which may lead to a failure [29].

A test case must consist of two components [30]: (1) a description of the *input* data to the SUT and (2) a description of the expected output of the SUT for that set of input data—an *oracle*. The oracle is used to verify the correctness of the outputs produced by the SUT. It is usually performed by testers but an oracle can be a specification or even another program [26].

The test oracle may either be automated or manual; in both cases, the output is compared to a correct output [31]. In unit tests, for example, the developer explicitly asserts an oracle for each of the test cases. However, the most common is the *manual oracle* where the tester manually interacts with the SUT visually checking for errors.

Tests are commonly divided into three levels [28, 29, 32]: *Unit testing* focusing on the smallest components of a SUT; *Integration testing* for complex integration of classes/procedures; and *System testing* for the main (or risky) flow of the application. Also, *regression testing* corresponds to a subset of the previous tests to ensure that changes do not break previously-working code while *acceptance testing* is used by clients to assess the final product.

Software tests can be performed by the development team, for which the internals of the SUT are known (white-box testing [32]), or by an external party focusing on the SUT visible, external functionalities (black-box testing [33]) [28]. Black-box testing only uses the SUT specification to describe and verify test cases, not the internal structure of the SUT.

2.1.2 Automated Software Testing

As the size of software systems keeps increasing, the testing also becomes complex. The classic answer of software engineers to reduce cost and complexity is automation [26]. Test automation reduces the cost and time used during the testing process, improves efficiency, and reduces human errors [34]. It is "the use of special software (separate from the SUT) to control the execution of tests and the comparison of actual outcomes with predicted outcomes" [27]. Testing scripts are the common method of automated software tests. They consist of a pre-defined sequence of actions (as inputs) compared with manually defined oracles [35].

2.1.3 Video Game Testing

There is a blurry line between software testing and video game testing (sometimes referred to as "playtesting"). We choose to consider video games as software with a different purpose: to provide an experience (engagement) to the player[23]. Video games, aside from having code, also integrate artistic elements (sound, 2D/3D graphics, narrative, etc.). Also, in video games, aside from testing techniques like white-box and black-box testing, developers must assess other attributes, like experience and usability [23].

Video game testing is a dynamic analysis of the SUT (a version of the game or a Game Under Test), it is usually performed manually by video game testers in playtesting sessions. Besides checking if the game works and finding faults, testers also assess if the game version is fun or engaging (among many other things). Yet, the fun factor is the main specification of a game: the game, more than anything, must be fun.

Usually, the process to reproduce a problem is described in a report (bug report) written by testers, detailing all the necessary steps to reach the point where the problem occurred. The report consists of text documents with a title, a short description, and the steps [36]. Often, it is hard to replicate the steps (inputs) made by the tester. This situation is even more cumbersome if the game is not deterministic, where the same input does not produce the same output. As randomness is considered a desired feature in games allowing players to keep engaged with the same game (replayability), reproducibility becomes a challenge.

2.1.4 Game User Research (GUR)

Game User Research (GUR) is the research field that focus on *usability* and *user experience* (UX) in video games. This involves any aspect of a video game with which players interact, like menus, audio, artwork, underlying game mechanics, etc. Testing video games involve trying to answer *why* the player is doing something [37].

In practice, GUR involves many fields, like human-computer interaction, psychology, graphic design, marketing, computer science, analytics, etc. Different than QA and technical game testing, GUR methods focus on evaluating players by observing them interact with the game. The goal is to improve the game using empirical evidence from experimentation and testing [37].

There are different methods to assess the players, which differ across development phases. For example, during pre-production, the main concern is to test the core game loop using the prototypes; while in later phases, the focus is on balancing and tuning. Finally, as for the GUR methods of testing, refer to [37].

[23] Here we are referring to games that provide only entertainment, not educational or training games.

There is a difference between GUR and video game testing. Both use playtesting sessions, but, GUR research deals with subjective aspects of the game while video game testing focuses on finding bugs and other technical aspects.

2.2 Postmortem

Contrary to other software industries, game developers *do* share information about their games projects in the form of postmortems, which are also called "war stories". These war stories are informal texts that summarise the developers' experiences with their games projects, often written by managers or senior developers [38] right after their games launched [4]. They often include sections about "What went right" and "What went wrong" during the game development:

- *"What went right"* discusses the best practices adopted by the game developers, solutions, improvements, and project-management decisions that helped the project.
- *"What went wrong"* discusses difficulties, pitfalls, and mistakes experienced by the development team in the project, both technical and managerial.

2.3 Software Framework

Pree [39] defined frameworks as having *frozen* and *hot* spots: code blocks that remain unchanged and others that receive user code to build the product. Larman [40] observed that frameworks use the *Hollywood Principle*, "Don't call us, we'll call you.": user code is called by the framework. Taylor [41] sees a framework as a programmatic bridge between concepts (such as "window" or "image") and lower-level implementations. Frameworks can map architectural styles into implementation and–or provide a foundation for architecture.

GitHub uses a set of "topics"[24] to classify projects. It defines the topic "framework" as *"a reusable set of libraries or classes in software. In an effort to help developers focus their work on higher-level tasks, a framework provides a functional solution for lower-level elements of coding. While a framework might add more code than is necessary, they also provide a reusable pattern to speed up development."*

[24] https://github.com/topics/framework

2.4 Game Engine

The studio id Software[25] introduced the concept of a video-game engine in 1993 to refer to the technology "behind the game" when they announced the game DOOM [42, 43]. In fact, they invented the game engine around 1991 and revealed the concept around the DOOM press release in early 1993 [44]. They created the first game engine to *separate the concerns* between the game code and its assets and to *work collaboratively* on the game as a team [43, 45]. Also, they "lent" their engines to other game companies to allow other developers to focus only on game design. John Carmack[26], and to a less degree John Romero[27], are credited for the creation and adoption of the term game engine.

The invention of this game technology was a discrete historical event in the early 1990s but it established MS-DOS 3.3 as a relevant gaming platform, mostly because of the NES-like horizontal scrolling emulation, allowing developers to create games similar to the ones on the Nintendo console. It also introduced the separation of the game engine from "assets" accessible to players and thereby revealed a new paradigm for game design on the PC platform [44], allowing players to modify their games and create new experiences. This concept has since evolved into the "fundamental software components of a computer game", comprising its core functions, e.g., graphics rendering, audio, physics, AI [43].

In theory, game engines and frameworks have similar objectives: they are *modular platforms* for *reuse* that provide a *standard* way to develop a product, lowering the barrier of entry for developers by *abstracting* implementation details.

We could classify frameworks into different categories, according to their domains, e.g., Web apps, mobile apps, AI, etc. In the same category and across categories, the two frameworks are not the same. They provide their functionalities in different ways. Similarly, game engines also belong to different categories and are different from one another. For example, 3D or 2D and specific for game genres, like *platformer*, *shooter*, *racing*, etc.

Traditional frameworks provide business services while game engines support entertaining games [46]. The process of finding the "fun factor" is exclusive to game development [47, 48] but does not exempt developers from using traditional software-engineering practices [3, 49]. Game engines are tools that help game developers to build games and, therefore, are not directly concerned with non-functional requirements of games, such as "being fun".

[25] https://www.idsoftware.com

[26] John Carmack was the lead programmer and co-founder of id Software in 1991.

[27] John Romero was the designer, programmer and also co-founder of id Software in 1991.

Chapter 3

Video Game Problems

"(...) we had the development of [the game] ironed out to five full-time developers working for six months. Fact: [the game] took eight full-time and between two and four part-time developers 24 months to barely finish. Our initial estimate was off by more than 700 percent."

ARROWHEAD GAME STUDIOS CEO JOHAN PILESTEDT

In this chapter, we present an exploratory study investigating problems in the video-game industry and the anti-patterns that pertain to this industry. We use this study to obtain a detailed overview of the problems in video game development as a whole. The results of this study show gaps in the practice and research regarding video-game development, in particular technical problems in the production phase, which we explore in more detail in the following chapters.

3.1 Context

The game industry is known for its problems. They range from technical problems, e.g., 80% of the games on Steam require critical updates [50], to management problems, e.g., crunch time [51] and unrealistic scopes [3]. The problems in the game industry also include mistreatment of employees[28] and harassment[29]. Yet, the game industry continues to make profits[30] as players keep on buying its games, reinforcing a cycle of bad practices.

In this study, we provide a state of the problems of video-game development, in particular the

[28] https://bit.ly/3h6ZKer
[29] https://bit.ly/391zf7B
[30] https://bit.ly/3haHukG

problems faced by game developers, their evolution in time, and their root causes. We analyze 200 postmortems written between 1997 and 2019 available in our public dataset [19] of grey literature related to game development. These postmortems include 927 problems that we categorized into 20 types. Through our analysis, we draw a landscape of game-industry problems in the past 23 years and how these problems evolved over the years. The content of this chapter is based on Politowski et al. [20].

3.2 Related Work

Callele et al. [38] analyzed 50 postmortems from the Game Developer Magazine, written between 1999 and June 2004, and investigated how requirements engineering was applied to game development. They reported that internal problems are 300% more prevalent than that in other categories. Most internal problems related to project management: missing tasks and poor task estimation.

Petrillo et al. [3] analyzed 20 postmortems published on the Gamasutra Website to identify recurring problems and compare them with traditional software-engineering problems. They concluded that (1) video-game development suffers more from management problems rather than technical ones; (2) problems in video-game development are also found in traditional software development; and, (3) common problems are *Scope*, *Feature Creep*, and *Cutting Features*. They also reported that multidisciplinary teams in large game studios are also a source of problems:

Kanode and Haddad [52] used postmortems to discuss the challenges of adapting traditional software engineering to video-game development. They reported differences between game development and traditional development, specially regarding *Asset Diversity*, *Project Scope*, *Game Publishing*, *Project Management & Team Organization*, *Development Process*, and *Third-Party Technology*.

Lewis and Whitehead [47] used two previous papers [53, 54] to identify problems in game development and assess whether/why these could be of interest to software-engineering researchers. They highlighted some differences between games and traditional software. They reported that, in large game studios, *teams* are multidisciplinary and tightly coupled and that they suffer from tight budgets and deadlines. They also wrote that larger teams require strong leadership due to constant developer turnover. Finally, the authors stated that documenting a game upfront is pointless as new features are added regularly, making documentation obsolete. We observed only 2% of documentation problems in our dataset. Some developers stated the need for a clear vision, but not game-design documents. Developers want a clear vision more than documentation.

Washburn et al. [4] analyzed 155 postmortems, written over 16 years. They identified some characteristics and pitfalls of game development and suggested good practices. The authors reported that the most common problem relates to *Teams*, similar to our observations in which *Teams*

problems are the third most common (8%), e.g., lack of communication and disagreement among developers. They also reported that scheduling and process are recurring problems, which we also support with our findings from the dataset: underestimation and management are reported as the main causes of planning problems.

Edholm et al. [51] conducted interviews at four different game studios and reviewed 78 postmortems to investigate the culture of crunch-time in the game industry. According to their interviewees, crunch time is common within the game industry as the majority of game studios applied such practice. From their postmortem data, 45% mentioned crunch-time. Also, crunch-time has been within the game industry from early 2000 to the current date (2014). Moreover, small studios are more prone to crunch (54% crunch) than both micro-(33%) and medium-sized (36%) studios.

Table 2 summarises these previous works, their methods and goals. These previous works used postmortems to discuss video-game development problems. They used *ad-hoc* classifications for the problems.

Table 2: Summary of the related works on video game problems.

Paper	Postmortems'		Study goal
	Analysis	#	
2005 Callele et al. [38]	Yes	50	Requirements
2009 Petrillo et al. [3]	Yes	20	Problems
2009 Kanode and Haddad [52]	Yes	?	Challenges
2011 Lewis and Whitehead [47]	No	–	Problems
2016 Washburn et al. [4]	Yes	155	Characteristics
2017 Edholm et al. [51]	Yes	78	Crunch-time
Our study	Yes	200	Problems

3.3 Method: Creating the Dataset from Postmortems

For the video game problems, we extended the dataset defined in the previous work [19]. We started with each author randomly picking one postmortem from the Gamasutra Website between the years 1997 to 2019. Each author read the postmortem, focusing on the "What went wrong" section. Using the coding technique from Grounded Theory [55], we identified the problems reported by the game developers, extracting *quotes*, and grouping similar *problem types*.

As a starting point, we created a list of *problem types* based on the previous literature definitions of Petrillo et al. [3] and Washburn et al. [4]. Thus, we discussed the findings, and reached a consensus about the *problem types* (Table 3). Any change resulted in updates in the *dataset* and the list of *problem types*. This process continued until we reached 200 postmortems. To have a

better macro idea of the problems, we decreased the granularity of the problem types by clustering them into four groups: *Production*, *People Management*, *Feature Management*, and *Business*.

Table 3: *List of problem types* of video-game development problems identified through the post-mortem analysis. The *types* that are also used by Petrillo et al. [3] or Washburn et al. [4] are described *P*, and *W*, respectively.

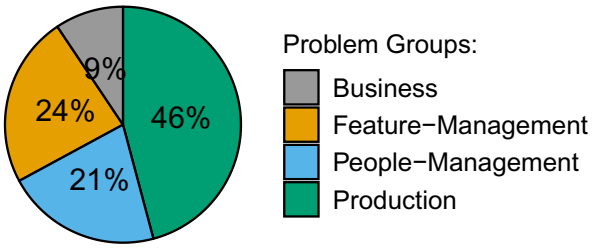
Problem Type	Description
Bugs[P]	Bugs or failures that compromise the game development or its reception.
Game Design[PW]	Game design problems, like balancing the gameplay, creating fun mechanics, etc.
Documentation[PW]	Not documenting the code, artifacts or game plan.
Prototyping	Lack of or no prototyping phase nor validation of the gameplay/feature.
Technical[P]	Problems with code or assets, infra-structure, network, hardware, etc.
Testing[PW]	Any problem regarding testing the game, like unit tests, playtesting, QA, etc.
Tools[PW]	Problems with tools like Game Engines, libraries, etc.
Communication[P]	Problems communicating with any stakeholder, team, publisher, audience, etc.
Crunch Time[P]	When developers continuously spent extra hours working in the project.
Delays	Problems regarding any delay in the project.
Team[PW]	Problems in setting up the team, loss of professionals or outsourcing.
Cutting Features[P]	Cutting features previously planned due to other factors like time or budget.
Feature Creep[P]	Adding non-planned new features to the game during its production.
Multiple Projects	When there is more than one project being developed at the same time.
Budget[PW]	Lack of budget, funding, and any financial difficulties.
Planning[W]	Problems involving planning and schedule, or lack of either.
Security	Problems regarding leaked assets or information about the project.
Scope[PW]	When the project is has too many features that end up impossible to implement it.
Marketing[W]	Problems regarding marketing and advertising.
Monetization	Problems with the process used to generate revenue from a video game product.

The dataset is available in an open repository on GitHub[31] so that researchers and practitioners can access and contribute through *pull requests*. We choose this approach to curate contributions before inclusion. Contributors can also add problem types and other metadata, e.g., genres, to the list.

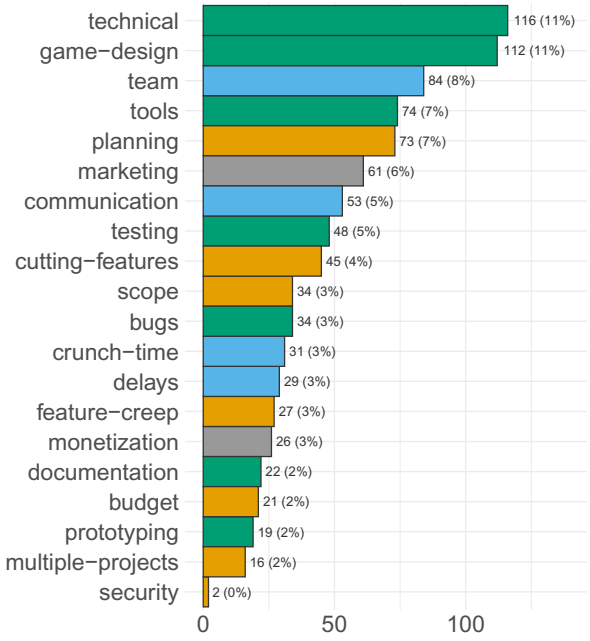
[31] https://github.com/game-dev-database/postmortem-problems

3.4 Results: Showing the Problems in Game Industry

The dataset contains 200 video-game projects from 1997 to 2019, describing 927 problems. On average, there are five problems by game and 40 by year. Figure 3a shows the problems by groups: 46% of the problems relate to production, 45% to management, and 9% to business.



(a) Number of problems related to each Group.



(b) Number of problems related to each Type.

Figure 3: Overall dataset results for problem groups and types.

Figure 3b shows the distribution of the problems by types. *Game design*, *technical*, and *team* problems are the most frequent, with 30% overall. Although management and production problems have close percentages, the two most common problems types, *technical* and *game design*, with 11% each, are related to *production*. *Management* problems are spread among problem types.

Figure 4 shows the normalised number of problems per group or per year. For example, in 2018, there were five business problems among 16 problems. *Production* problems remain constant. *Management* problems peaked in 1998 and are less frequent now. *Business* problems increased over the years.

Figure 5 shows the four different patterns in the dataset. To normalize the number of problems each year, we divide their numbers by the total number of problems that year. The red line (curved line) is a second-degree polynomial function. The grey area represents the confidence interval (0.95 by default) of the function. Figure 5a shows that *Marketing* problems increase over the years. *Monetization* and *Bugs* are also problems that follow this trend, but to a lesser degree. On

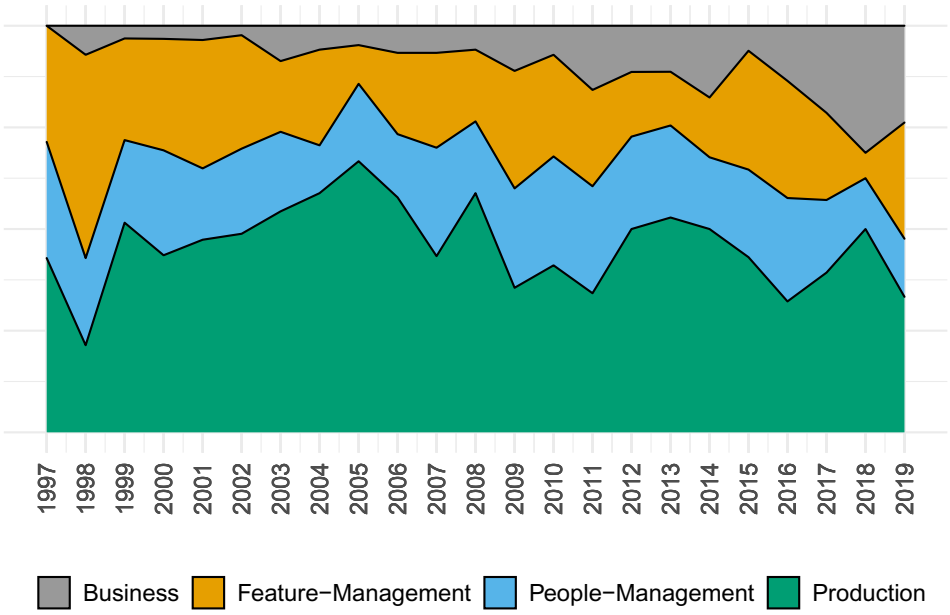


Figure 4: Problems over the years by groups.

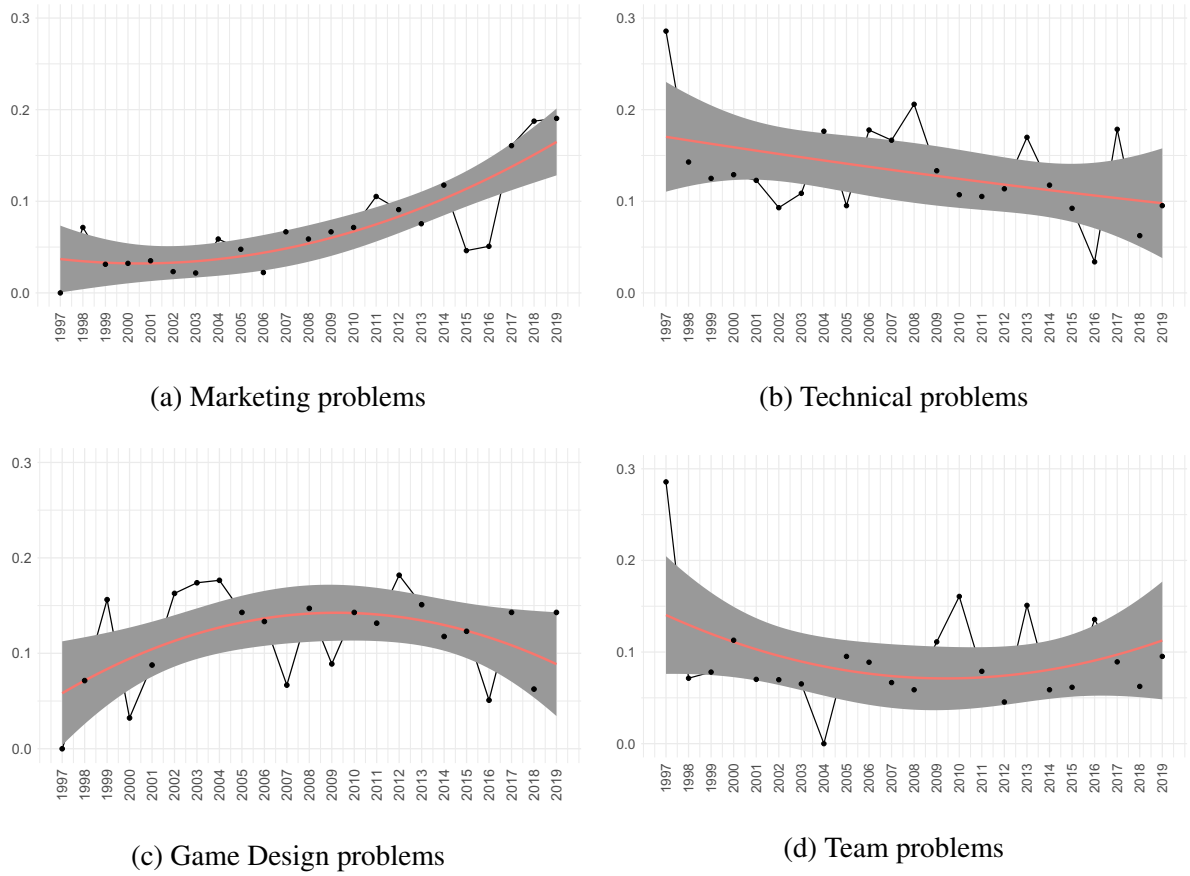
the contrary, Figure 5b shows the decrease of *Technical* problems over the years. Other problem types also follow this trend: *Documentation*, *Testing*, *Cutting Features*, and to a lesser degree *Feature Creep* and *Communication*. We also observe problem types whose trends changed in the last decade. For example, Figure 5c shows that *Game Design* problems, the most notorious case of a problems, decreased in the last decade. To a lesser degree, this pattern is followed by the problem types: *Tools*, *Delays*, *Crunch Time*, *Budget*, *Planning*, and *Prototyping*. However, some problems increased in the last decade. Figure 5d shows the most evident example of problems related to development *Teams*. Problems related to project *Scope* also follow this trend.

We further investigate the problems and identify the root causes of each problem type. We read all the problems again classifying the types into sub-types. We found a total of 105 different sub-types. Table 4 describes the top 10 sub-type problems.

3.5 Discussing the Problems

Production Problems: *Production* problems remain constant to today. The clearest spike in the data occurred in 2005, which might be related to the arrival of a new console generation that year: the seventh generation, e.g., Sony Playstation 3, was released between 2005 and 2006. The Playstation 3, with its new architecture[32], was notoriously difficult to program and Sony shared

[32] https://venturebeat.com/2014/07/06/last-gen-development/



(a) Marketing problems

(b) Technical problems

(c) Game Design problems

(d) Team problems

Figure 5: Four common patterns of the importance of the problems over the years. (a) Shows the Marketing problem that increased since 1997. (b) Shows Technical problems that decreased since 1997. (c) Shows Game Design problems that decreased in the last decade. (d) Shows Team problems that increased in the last decade.

Table 4: The top 10 most common sub-type problems.

Type	SubType (root cause)	N
Team	Insufficient workforce	49
Team	Environment problems	48
Marketing	Wrong marketing strategy	35
Planning	Underestimation	34
Game Design	Unclear game design vision	28
Game Design	Lack of fun	27
Technical	Platform and technology constraints	24
Game Design	Game design complexity	23
Tools	Inadequate or missing tools	22
Communication	Misaligned teams	22

some information only with first-party studios[33].

[33] https://cnet.co/3haIEfN

Management Problems: *Management* problems peaked in 1998 and are less frequent now. One factor that helped decrease management problems might be the adoption of agile methods. The game industry, even today, often works with old development methods, e.g., Waterfall [56], yet agile methods, born around the 2000s, are being adopted gradually. Pre-production and Production may not be both amenable to agile methods. However, the concrete development of the game, during production, could benefit from using agile methods.

Business Problems: The problems with *Business* increased over the years. Our hypothesis is that the rise of Indie developers, in particular, the "one-man-army" teams in which one developer does all the tasks[34], contributed to increasing this problem. Indie developers do not have publishers or colleagues to deal with marketing and often perform related tasks poorly. Their business knowledge is often limited.

Developer Turnover: "The game industry is cyclical, constantly churning employees in and out depending on the needs of a project"[35] and, thus, game developers often change companies. Therefore, teams are also constantly changing, having to adapt to newcomers. This turnover happens during all the development phases because game projects are long-duration projects. Even with a clearly-defined process for newcomers, with mentoring from senior developers, their productivity will be low at first, yet micromanagement must be avoided[36].

Insufficient Workforce: We observed that *Insufficient workforce* is mainly caused by poor management of the project *Scope* (requirements), which leads to other problems like *Cutting Features* and *Crunch Time*. Game developers report that pair programming [57] and code reviews [58] are not common in the game industry. Similar practices could be adapted in the game industry, even pairing technical and non-technical developers.

Wrong Marketing Strategy: *Marketing* is the problem type that increased the most in the study period. We observed that its main causes are threefold: new audience acquisition (need for new strategies), lack of expertise in promoting games (especially in indie companies), and saturation of the game market (need to stand out). The way developers communicate about their games evolved from magazines in 1997, through forums, social media, and online stores, to today's streamers and independent reviewers[37]. Taking advantage of new media is difficult. Indie developers, with

[34]Some examples of (successful) games written by only one developer are "Stardew Valley" and "Dust: An Elysian Tail".

[35]https://bit.ly/2WmLWET

[36]The director of Final Fantasy 14 had to micromanage the team to keep the production pace, but he advises not to do it in https://youtu.be/Xs0yQKI7Yw4.

[37]https://bit.ly/2Zyu77Q

low marketing budgets, often fail by trying to reach too many "influencers". A lack of marketing expertise and a crowded game market[38] make it difficult to be noticed.

Underestimation: Game *estimation*, like for any other software project, varies across game projects. Teams move from one game to another and must adapt to technological advances and different requirements, which make estimation difficult. They must invest in long pre-production phases to research and understand new technologies, tools, and game designs. We believe that better estimation comes with better information about the previous projects. However, the closed nature of most games makes sharing information difficult.

Unclear Game Design Vision: An *unclear game-design vision* impacts the entire game project, including management and testing. Although related to game design and art, the game-design vision must be embraced by the whole team and, thus, is also a management problem. Teams need to understand the project vision to avoid wasted work. They should spend less time defining static documents, which become quickly obsolete, and more time in pre-production until the core mechanics and the fun factor is clear. They must prototype and playtest. Finally, they should keep creative control over the projects, to the greatest extent feasible.

Platform and Technology Constraints: The gaming market is spread across different platforms and developers must publish on different platforms to reach more players and sell more games. Platform constraints stem from the differences among/within consoles, mobiles, and PCs. Platform constraints are often defined by the lowest common denominator in consoles, mobiles, and PCs. They include slow read-and-seek times on hard drives, CPUs with low clock speeds and numbers of cores, and old graphics cards. Developers should assess the viability of their games on the technical specifications of the target platforms (for which they should reserve time for experimentation).They should also *gracefully degrade* their games on lower-end devices or *progressively enhance* them on more capable platforms.

Inadequate or Missing Tools: Tools often frustrate developers, in particular game engines. For example, in two-game projects, EA[39] forced their developers to use their proprietary Frostbite engine, causing delays and reworks for the game "Dragon Age 3" [59] and the failed project "Anthem"[40]. Game engines can speed up game development but also constrain game designs. They are

[38] https://bit.ly/30fTUAI
[39] Electronic Arts is a publisher and owner of many video game studios, see https://www.ea.com/en-ca.
[40] https://bit.ly/39fTnTt

few, including Unity and Unreal, and proprietary, closed-source engines in large companies. Although open-source, Godot[41] is not yet as mature as its proprietary counterparts. Game developers should carefully choose their game engines according to: (1) the project goal – Can we implement the game using this engine? (2) the team experience – Is the team comfortable with this engine? (3) the development schedule – Should we build, extend, or use a third-party engine? and, (4) the game budget – What is the trade-off between licensing and supporting our own game engine?

Game Testing: Given the importance and emphasis given to software testing in traditional software development, we were surprised to find little information about testing in-game projects in the postmortems (only 5% of all problems we found). One hypothesis could be that testing is largely successful and therefore does not need mentioning in the postmortems. However, it is well known that games often suffer from low quality and that game projects often overrun their schedules, hinting that testing is probably problematic. Therefore, our next hypothesis is that testing, in particular software-engineering testing, is under-performed by game developers. Indeed, postmortems mention playtesting but do not mention unit testing or integration testing. This lack of mention is interesting and calls for more research on game developers' testing habits (or lack thereof) and the reasons for these habits.

Chapter 3 Summary

In this chapter, we reported the main problems in the game industry, their evolution over the years, and their relationship with anti-patterns in traditional software. We identified several gaps in the research and practice related to video game development. One of these gaps concerns the tools used by developers to develop games. For example, the learning curve of game engines, their complexity, and lack of features. Game engines are at the heart of all games but are currently difficult to master due to their sizes and complexities and, therefore, poorly understood in practice and in research. In the next chapter, we focus on game engines and explore their architecture, compare their quality with traditional software systems and assess their limitations. Another gap is game testing, which is also connected to game engines. Therefore, in the next chapter, we study game engines while in the following ones, we study testing.

[41] https://godotengine.org/

Chapter 4

Video Game Engines

"It's hard enough to make a game (...). It's really hard to make a game where you have to fight your own toolset all the time."

SCHREIER [60]

In this chapter, to better understand the main tool used by game developers, we investigate open-source game engines and compare them with traditional open-source frameworks. Frameworks are used by developers to ease software development and to focus on their products rather than on implementation details. Similarly, game engines help developers create video games and avoid duplication of code and effort. Yet, video-game engines often frustrate developers. For example, for two games, Electronic Arts forced their developers to use their proprietary Frostbite engine, causing delays and reworks for "Dragon Age 3" [59] and leading to the failure of "Anthem"[a]. Yet, we do not know why game engines are a source of these development issues nor how can we properly deal with it.

[a] https://bit.ly/39fTnTt

4.1 Context

During game development, developers use specialized software infrastructures to develop their games; chief among which are *game engines*. Game engines encompass a myriad of resources and tools [42, 61–63]. They can be built from scratch during game development, reused from previous games, extended from open-source ones, or bought off the shelves. They are essential to game development but misunderstood and misrepresented by the media[42] and developers due to the lack

[42] https://tinyurl.com/bdf4r9fb

of clear definitions, architectural references [64], and academic studies. They are also the source of problems, especially between design and technical teams [45, 59].

To address these problems, some researchers suggest the use of software-engineering techniques [3, 46, 65] while others consider game development as a special kind of software and propose new engineering practices or extensions to classical ones [10, 12, 47, 48, 52, 66, 67]. However, they did not study a large number of game engines, either proprietary, because only 13% of all the games on Steam describe their engines [68], or open-source. They also did not survey game engine developers.

Here, we study open-source game engines from the perspectives of *code* (Section 4.3) and *human* (Section 4.4). We aim to provide a global view of the state of the art and practice on game engines. We explore academic and gray literature on game engines; compare the characteristics of the 282 most popular engines and the 282 most popular frameworks on GitHub; and, survey 124 engine developers about their experience with the development of their engines.

4.2 Related Works

There are few academic papers on game engines. Most recently and most complete, Toftedahl and Engström [68] analyzed the engines of games on the Steam and Itch.io platforms to create a taxonomy of game engines. They highlighted the lack of information regarding the engines used in mainstream games with only 13% of all games reporting information about their engines. On Steam, they reported Unreal (25.6%), Unity (13.2%), and Source (4%) as the main engines. On Itch.io, they observed that Unity alone has 47.3% of adoption among independent developers.

Messaoudi et al. [64] investigated the performance of the Unity engine in-depth and reported issues with CPU and GPU consumption and modules related to rendering. Cowan and Kapralos [69] in 2014 and 2016 [70] analyzed the game engines used for the development of serious games. They identified few academic sources about tools used to develop serious games. They showed that "Second Life"[43] is the most mentioned game engine for serious games, followed by Unity and Unreal. They considered game engines as parts of larger infrastructures, which they call frameworks and which contain scripting modules, assets, level editors as well as the engines responsible for sound, graphics, physics, and networking. They ranked Unity, Flash, Second Life, Unreal, and XNA as the most-used engines.

Neto and Brega [71] conducted a systematic literature review of game engines in the context of immersive applications for multi-projection systems, aiming at proposing a generic game engine for this purpose. Wang and Nordmark [72] assumed that game development is different from

[43] Second Life is not a game engine per se but a game that can be extended by adding new "things" through "mod" or "modding".

traditional software development and investigated how architecture influences the creative process. They reported that the game genre significantly influences the choice of an engine. They also showed that game-engine development is driven by the creative team, which requests features from the development team until the game is completed. They observed that adding scripting capability eases game-engine development through testing and prototyping.

Anderson et al. [73] raised issues and questions regarding game engines, among which the need for a unified language of game development, the identification of software components within games, the definition of clear boundaries between game engines and games, the links between game genres and game engines, the creation of best practices for the development of game engines.

4.3 The Code Perspective

With respect to the design and implementation of game engines and traditional frameworks, we study their static, historical, and community characteristics. From a code perspective, we investigate the static attributes of the projects, like their size, complexity of the functions, programming languages and licenses used. For the historical characteristics of the projects, we compare the life cycles of game engines and traditional frameworks. We analyze the tags released (versions), projects' lifespan and commits. Finally, to investigate the interactions of the OSS community on the projects, we analyze the popularity of the projects, the number of issues reported in these projects, and the truck-factor measure [74].

We gathered the top 1,000 projects in GitHub related to the *game-engine* and *framework* topics. We filtered these projects using the following criteria to remove "noise". We obtained 458 engines and 743 frameworks: We manually analyzed the remaining $458 + 743 = 1,201$ projects to remove those that are neither game engines nor frameworks. We kept 282 game engines and 282 frameworks. The dataset, scripts and all the material from this study are in its replication package[44].

As for the analysis, we used the statistical-analysis workflow model for empirical software-engineering research [75] to test statistically the differences between engines and frameworks. For each continuous variable, we used descriptive statistics in the form of tables with mean, median, min, and max values, together with boxplots. For the boxplots, to better show the distributions, we removed outliers using the standard coefficient of *1.5* ($Q3 + 1.5 \times IQR$). We observed outliers for all the measures, with medians skewed towards the upper quartile (Q3). To check for normality, we applied the Shapiro test [76] and checked visually using Q–Q plots. Normality < 0.05 means the data is not normally distributed. Finally, given the data distribution, we applied the appropriate statistical tests and computed their effect sizes.

[44] https://doi.org/10.5281/zenodo.3606899.

4.3.1 Static Characteristics

Table 5 shows the results of Wilcoxon tests. The p-values < 0.01 indicate that the distributions are not equal and there is a significant difference between engines and frameworks, although this difference is *small*. The biggest effects are related to source code metrics, i.e., *nloc_mean* and *cc_mean*.

Table 5: Statistical Tests for Static Characteristics.

Variable	P-value	Estimate	Effect
main_language_size	<0.01	0.28	0.189 (small)
total_size	<0.01	0.34	0.188 (small)
n_file	<0.01	45.00	0.155 (small)
n_func	<0.01	769.00	0.211 (small)
nloc_mean	<0.01	2.12	0.297 (small)
func_per_file_mean	<0.01	3.13	0.208 (small)
cc_mean	<0.01	0.53	0.356 (small)

The implementation of game engines and traditional frameworks are different but without statistical significance. Engines are bigger and more complex than frameworks. They use mostly compiled programming languages vs. interpreted ones for frameworks. They both often use the MIT license.

Differences in Programming Languages

There is a discrepancy between the languages used in the game engines (Table 6), which belong mostly to the C family, and frameworks, developed mostly with interpreted languages. We explain this difference as follows: engines must work close to the hardware and manage memory for performance. Low-level, compiled languages allow developers to control fully the hardware and memory. Frameworks use languages providing higher-level abstractions, allowing developers to focus on features. Frameworks and engines are tools on which developers build their products, and choose the most effective language for their needs.

We explain the predominance of C++ for engines by a set of features of this language: abstraction, performance, memory management, platforms support, existing libraries, and community. These features together make C++ a good choice for game developers.

Engines are usually written (or extended) via their main programming language. However, to ease the design, implementation, and test workflow during production, game developers often add scripting capabilities to their engines. Therefore, when writing a game, game developers may not code directly with low-level languages but use scripts; sometimes within a specific domain-specific

Table 6: Popularity of programming languages among engines and frameworks.

	Engine		Framework		Total	
	N	%	N	%	N	%
C++	107	37.94%	10	3.55%	117	20.74%
JavaScript	28	9.93%	71	25.18%	99	17.55%
Python	14	4.96%	45	15.96%	59	10.46%
C	41	14.54%	11	3.90%	52	9.22%
PHP	3	1.06%	46	16.31%	49	8.69%
C#	33	11.70%	15	5.32%	48	8.51%
Java	27	9.57%	19	6.74%	46	8.16%
Go	14	4.96%	21	7.45%	35	6.21%
TypeScript	7	2.48%	18	6.38%	25	4.43%
Swift	2	0.71%	13	4.61%	15	2.66%
Scala	1	0.35%	5	1.77%	6	1.06%
Objective-C	1	0.35%	4	1.42%	5	0.89%
Lua	4	1.42%	0	0.00%	4	0.71%
Ruby	0	0.00%	4	1.42%	4	0.71%

language. For example, Unity, although written in C++, offers scripting capabilities in C#[45] for game developers to build their games. Furthermore, the developer can, possibly, finish its game just by using this high-level language. For any further extension in the game engine, they will need to deal with the low-level language. On the other hand, frameworks rarely offer scripting capabilities: their products are often written in the same programming languages.

4.3.2 Historical Characteristics

Table 7 shows the results of Wilcoxon tests, showing large differences for all historical measures except `lines_added`, `lines_removed`, and `code_churn`. Overall, all metrics have similar median values when comparing both groups, except for *tags_releases_count*. In fact, engines release way fewer versions (median is one) than frameworks (median is 32). Versioning does not look like a well-followed practice in engine development, with few versions compared to frameworks. Commits are less frequent and less numerous in engines, which are younger and have shorter lifetimes when compared to frameworks.

Our results showed that 40% of the engines do not have tags, which could mean that they are still under development and no build is available. However, our dataset contains the most important game engines on GitHub, thus there should be other reasons for the lack of engine releases. During our manual analysis, we found engines with warning messages alerting us that they were incomplete, and lacking some essential features. Also, we observed that about one-third of the engines have only two collaborators. This fact combined with the complexity of engines

[45] https://docs.unity3d.com/Manual/ScriptingSection.html

Table 7: Statistical Tests for Historical Characteristics.

Variable	P-value	Estimate	Effect
tags_releases_count	<0.01	-24.00	-0.613 (large)
lifespan	<0.01	-56.29	-0.32 (large)
commits_count	<0.01	-175.00	-0.198 (large)
commits_per_time	<0.01	-0.30	-0.198 (large)
lines_added	<0.01	134.47	0.219 (small)
lines_removed	<0.01	48.83	0.168 (small)
cchurn_delta	<0.01	153.88	0.224 (small)
cchurn_sum	<0.01	222.48	0.212 (small)

could explain the difficulty to release a first feature-complete version.

On the other hand, frameworks are released more often than engines with more commits performed more regularly. There are thus meaningful differences between engines and frameworks, which could be explained by the higher popularity of the frameworks (see next section).

4.3.3 Community Characteristics

Table 8 shows the results of Wilcoxon tests, indicating a large difference in all measures related to the community. The truck-factor shows that the majority of the projects have few contributors. Some uncommon languages, like Go and C#, are popular compared to others in more prevalent projects, e.g., C++ and JavaScript. We observed that, although *static* characteristics of game engines and frameworks are similar, the *community* of these projects differ. Also, *historical* aspects are mixed, as engines have a smaller lifespan and fewer releases, yet similar effort on commits contribution and code churn.

Table 8: Statistical Tests for Community Characteristics.

Variable	P-value	Estimate	Effect
stargazers_count	<0.01	-358.00	-0.511 (large)
contributors_count	<0.01	-9.00	-0.459 (large)
truck_factor	0.01	<0.01	-0.138 (large)
issues_count	-139.00	<0.01	-0.451 (large)
closed_issues_count	-122.00	<0.01	-0.459 (large)
closed_issues_rate	-0.05	<0.01	-0.27 (large)

Differences in Truck Factor

The truck-factor is 1 for most of the engines (83%). Lavallée and Robillard [77] considered that, in addition to being a threat to a project survival, a low truck-factor causes also delays, as the knowledge is concentrated in one developer only. This concentration further limits adoption by

new developers. We believe that low truck-factor values are due to the nature of the engines, i.e., side/hobby projects. In contrast, popular frameworks do not have such a dependency on single developers.

Differences in Community Engagement

We assumed that the numbers of stars for projects on GitHub are a good proxy for their popularity [78]. Surprisingly, engines written in Go and frameworks written in C# are the most popular, even though their total numbers are low. JavaScript and C are second and third, respectively. Java is barely present despite its age and general popularity.

4.4 The Human Perspective

The human perspective pertains to the developers' perception of game engines and of their differences from traditional frameworks. We conducted an online survey with developers of the game engines to understand why they built such engines and their opinions about the differences (if any) between engines and frameworks.

Question 1 contains a predefined set of answers that we compiled from the literature and from the documentation and "readme" files studied during the manual filtering of the datasets. The respondent could choose one or more answers. We also provided a free-form text area for developers to provide a different answers and–or explain their answers. With Question 2, we want to understand whether game engine developers are also traditional software developers. Finally, Question 3 collected the developers' points of view regarding the differences (or lack thereof) between the development of engines and frameworks.

We used an online form to contact developers over a period of three days. We sent e-mails to 400 developers of the game engines in our dataset, using the truck-factor of each project: developers who collaborate(d) most on the projects. We received 124 responses, i.e., 31% of the developers. The survey, answers, and scripts for their analyses are in the replication package[44].

Question 1: Why did you create or collaborated with a video-game engine project? Figure 6 shows the breakdown of the developers' answers. Having access to the source code, freedom to develop, etc., i.e., *control of the environment*, is the developers' major reason for working on a game engine while *learning* to build an engine is the second reason; explaining why many engines have few developers and commits.

The third reason is to build a *game*, confirming the lack of clear separation between developers and game designers. It is indeed common for game developers to act also as game designers,

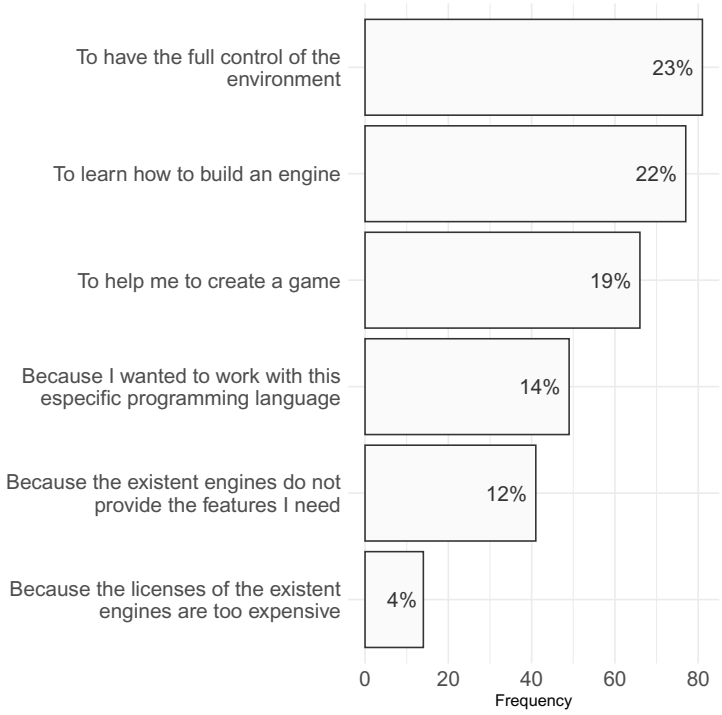


Figure 6: Answers to Question 1.

especially in independent games, e.g., the single developer of Stardew Valley[46].

Question 2: Have you ever written code for a software unrelated to games, like a Web, phone, or desktop app? The great majority of developers, 119 of the 124 respondents (96%), have experience with traditional software. The respondents can be considered general software developers with expertise in engine development.

Question 3: How similar do you think writing a video-game engine is compared to writing a framework for traditional apps? (Like Django, Rails, or Vue) Figure 7 shows that engine developers consider engines different from frameworks: 59% of the respondents believe that engines follow a different process from frameworks. Only 20% believe this process is similar. This is a surprising result as they also have experience in developing traditional software.

The developers' main reasons to work on an engine are (a) having better control over the environment and source code, (b) learning game-engine development, and (c) helping develop a specific game. Almost all engine developers have experience with traditional software. They consider these two types of software as different.

[46] https://www.stardewvalley.net/

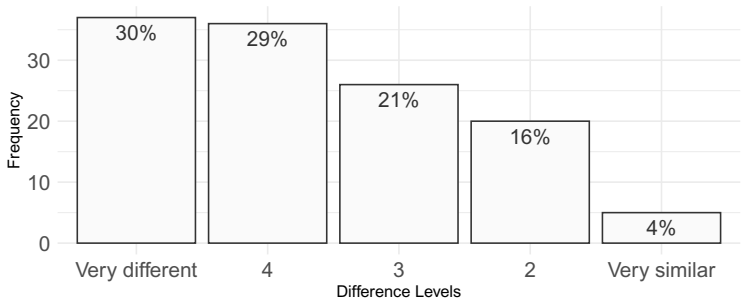


Figure 7: Answers to Question 3.

Chapter 4 Summary

In this chapter, we showed that there is a lack of academic studies about video-game engines, their characteristics and architectures. We also showed that there are qualitative but no quantitative differences between open-source engines and open-source frameworks. We performed a survey that showed that developers' objectives for developing engines are (a) better control of the environment and source code, (b) learning how to develop games and game engines, and (c) developing specific games. We conclude that open-source game engines share similarities with open-source frameworks, mostly regarding their concepts, code characteristics, and contribution effort. Yet, while game engines are mainly personal, the communities around framework projects are larger, with longer lifespans, more releases, better truck-factor, and more popularity. In this study, we did not find testing features in the game engines. Even mature proprietary game engines like Unreal and Unity lack in providing testing features. For example, in Unreal developers can create functional tests, but it requires a big effort, which is, therefore, ignored. In the next chapter, we explore what is video game testing, how developers deal with it, and the academic approaches.

Chapter 5

Video Game Testing

"Even today, at many large video game companies in the United States, QA testers make close to minimum wage and are treated like a lower caste, often given fewer benefits and told not to speak to other developers."

SCHREIER [79]

After exploring the game industry gaps in Chapter 3 we identified that game testing (or lack thereof) is one of the developers' main concerns. Plus, after studying game engines in Chapter 4 we noted the lack of features (or complete engines for that matter), especially testing features. Therefore, in this Chapter, we investigate game testing in the academic and gray literature. We study the field of testing in game development by surveying the processes, techniques, gaps, concerns, and point-of-views using two *academic literature* and *gray literature*. We also perform a literature review on video game automated testing techniques and an online survey with video game developers.

5.1 General Video Game Testing

Video-game projects are notorious for having day-one bugs, no matter how big their budget or team size. The quality of a game is essential for its success. This quality could be assessed and ensured through testing. However, little is known about video-game testing. *We want to understand how game developers perform game testing*. We study the field of testing in game development by surveying the processes, techniques, gaps, concerns, and point-of-views using two different sources: (1) the *academic literature* (journal and conference papers, books, and theses) and (2) the *gray literature* (presentations in well-known conferences, blog posts from game developers, and

video-game postmortems).

5.1.1 Method

Our approach is based on categories of reliability of the literature [80], with *white literature*, peer-reviewed papers, are the most reliable, followed by three gray literature tiers: first-tier, books, books chapters, government reports; *second tier*, annual reports, news articles, videos, presentations, Wiki articles; and, *third tier*, blogs, emails, tweets, letters.

We considered: (1) *academic literature*, journal, conference, and workshop papers, books and theses, and (2) *gray literature*, presentations in well-known conferences, blog posts from game developers, and video-game postmortems. We did not observe any duplication or extension (journal papers extending explicitly conference papers with new material). We performed the search for academic and gray literature on 2019/06/21.

Academic Literature

We used two sources of academic documents: Scopus[47] and Engineering Village[48]. The query was: (test OR testing OR verification OR validation OR qa OR "quality assurance" OR debugging OR prototyping) AND (game OR video-game OR "video game" OR "digital game") AND NOT (gamification OR "serious games" OR education OR teaching) AND LIMIT-TO (LANGUAGE, "English"). After the inclusion and exclusion criteria, the query yields 327 papers. After the title reading, abstract reading, full reading, and snowballing, The final dataset[49] included **96 papers**. The *inclusion* and *exclusion* selection criteria were:

- Inclusion criteria:
  - Papers must be about video game development;
  - Papers must be about or discuss testing;
- Exclusion criteria:
  - Papers not written in English;
  - Papers about gamification;
  - Papers about serious games;
  - Papers about using games for educational purposes;
  - Papers about using games for medical purposes.

[47] www.scopus.com
[48] www.engineeringvillage.com
[49] The dataset is available at https://github.com/game-dev-database/game-testing.

Gray Literature

For the gray literature, we used the same inclusion and exclusion criteria as the academic papers. We used three main sources of gray literature:

- Postmortems of video-game projects
  - We used a database of postmortems [19] and kept only the ones related to testing. We analyzed each one of them and extracted the root cause of the problems.
- Conferences on game development
  - We searched for talks about testing in game-developer conferences. We read/watched presentations and summarized the points related to game testing.

5.1.2 Game Testing Concerns Today

Heavy workload: As the workload for gameplay testing is tiresome[50], the testers often got motion sickness by constant re-playing the games [12]. In this sense, automation can reduce the burden on both developers and human testers, however, it also can introduce cost to game production, and it required programmers with a specialized focus on automated testing [81]. At the same time, the lack of test automation in game development "hurts" the bug fixing process as it becomes harder to reproduce the steps [12].

SE practices: Game developers overlook the SE practices. On testing, this happens because low-level testing is neglected for the sake of gameplay testing [12]. Regarding automated game testing, few studios use test automation alleging causes like development time, staff size, or little knowledge about these tools [13]. Usability tests and frameworks are also used within game development, but they are tailored specifically for the game domain [82]. Even pair and test-first programming, two practices well established in traditional software development, were not largely used by game companies [83].

Functional requirements: In game development, software working according to its functional requirements is not enough; the game must be appealing to the user [84]. This appeal involves balancing the gameplay, which is unique in game development and a challenge for the developers [48].

[50] https://kotaku.com/quality-assured-what-it-s-really-like-to-play-games-fo-1720053842

Proprietary code: Creating video games is a complex task. The game industry is on the edge of technical complexity. On top of that, the field of game development is hard to assess considering the restrictions of proprietary code [47].

5.1.3 Game Testing Challenges for the Future

Add game testing sooner to development pipeline: Game studios start testing for *User Experience* of games too late in the development life cycle, sometimes as late as beta, which means that most of the feedback obtained from the tests is unlikely to have an impact on the final game [85]. Key game concepts should also be testing before release so the reactions of the players can be verified [4]. Every aspect of a game should be tested during the development and production phases. The most important aspect of testing for game developers is to integrate testing as part of the production phase to improve efficiency. To ensure the delivery of quality games to the market, developers must consider different testing options during the production phase [86].

Instrument the game with meaningful logs and create tools to visualize the events: Combine playtesting with play analytics and advanced visualizations (e.g., using synthetic, procedurally generated game worlds to visualize gameplay data sets and temporal relationships) [84].

Add automating testing suitable for game development domain: The use of manual testing and the challenges with automated testing in game systems highlights the need for a new set of methodologies to ease the developers' ability to identify malfunctions and enable automatic testing activities for games. Studies should investigate new record–replay tools, automated test-data generators, etc. [87].

Define design patterns for game development: As the GoF patterns [88] are specific for OO systems, game development patterns are good practices specific for game genres [47]. Aside from game design patterns, which other patterns, more related to source code and more similar to traditional software, could be applied when building a game? How does the quality of code in games influence the game's functional requirements?

Create an universal game design language: The creation of a "universal game design language" could allow the game developers to reuse/extend a feature from game to game [47]. Can developers interchangeably develop a game regardless of the tools (game engine)?

5.1.4 Game Testing Problems on Gray Literature using Postmortems

We filtered all the testing problems discussed in Chapter 3. Table 9 summarizes the problems. Insufficient testing is the most common problem quoted by game developers. Either lack of tests, the need for more tests in the early phases of the development, the lack of unit testing and regression testing were also cited as well as beta testing. The time constraints were the main reason for not testing. Finally, developers also mentioned problems regarding testing-tools setups, like continuous integration, testing systems, and automation.

Table 9: Summary of the testing problems found in the postmortems.

Problem	N
Insufficient testing	22
Process and testing plans issues	18
Specific project requirements	13
Feedback	7
Scope	6
Reproducibility	2
Logging	2
No in-house QA	2
Combinatorial explosion	1
Bug fixing	1

5.1.5 Game Testing on Game Development Conferences

We searched for talks in conferences specialized in game development like GDC[51] and Digital Dragons[52]. Table 10 shows the most relevant talks about game testing. The information about the techniques is scarce, regardless, we summarized the main points below.

Table 10: Conferences on Game development.

	Title	Url
CASE-1	Smart Bots for Better Games: Reinforcement Learning in Production	https://bit.ly/2Hi5lyJ
CASE-2	Automated Testing and Profiling for 'Call of Duty'	https://bit.ly/37hf3w1
CASE-3	Automated Testing and Instant Replays in Retro City Rampage	https://bit.ly/2vuuSlS
CASE-4	It's Raining New Content: Successful Rapid Test Iterations	https://bit.ly/31KNk5I
CASE-5	Automated Testing of Gameplay Features in Sea of Thieves	https://bit.ly/2ONBFh8

[CASE-1] Smart Bots for Better Games: Reinforcement Learning in Production: Ubisoft[53] showcased their use of Reinforcement Learning (RL) in production to test games with bots that

[51] https://www.gdconf.com/
[52] http://digitaldragons.pl/
[53] https://ubisoft.com

can evolve while playing. RL has been applied/studied mainly in two areas of game development: improving AI behaviour and creating test assistants. The former got attention when the OpenAI team trained bots on Dota 2 (OpenAI Five)[54]. The latter approach uses agents to test games, which has been used by Ubisoft in Triple-A games like Far Cry: New Dawn (2019), which has an open world, and Rainbow Six Siege (2015), which is multiplayer.

[CASE-2] Automated Testing and Profiling for Call of Duty: Electronic Arts (EA) is a large company that develops and publishes games. EA has many studios working on different games. The talk was about the franchise Call of Duty, which receives a new iteration almost every year since 2003. The team developed a tool called Compass, designed to keep track of builds/testing. The team workflow with Compass consisted of five different checks during the development of the game: continuous integration (CI) module, "all maps" testing, nightly tests, and maintenance. The tool work with the help of game-play testing. For example, when testers find a place with a low frame rate, the tool record this location, adds more characters and collects metrics automatically.

[CASE-3] Automated Testing and Instant Replays in Retro City Rampage: The author used automated testing techniques to test his indie game, with a small scope. His solution was to record the inputs, in log files, and have the engine re-play them. It allowed him to track down bugs using a simple diff tool on the output files. Advantages of this input–record approach are automated QA, ease to deal with multiple platforms, and ease to narrow down game-play bugs.

[CASE-4] It's Raining New Content: Successful Rapid Test Iterations: Riot is the company developing League of Legends, which had 11 million daily players and 5 million concurrent players at the time of the talk. Thus, one bug, even a small glitch, could affect millions. Thus, they created a process with which testing is carried out carefully in different steps to detect/prevent bugs. Their process consists of daily play-tests with continuous delivery that uses days instead of weeks. They also use automated tests for performance and integration testing, with ad-hoc scripts to test the game loop and front end.

[CASE-5] Automated Testing of Gameplay Features in Sea of Thieves: Rare is the developer of Sea of Thieves, using Unreal Engine. They used the engine features to add test cases at different levels of abstraction. Rare has more than 23,000 tests and 100,000 asset audit checks. With this automated test process, they managed to keep a constant number of bugs and avoid working overtime. They divided the test into:

[54] https://openai.com/projects/five/

- Actor testing (70%): This type of test uses the Actor class of Unreal Engine to have lighter tests compared to integration tests, which have heavy dependencies and do not scale well;
- Unit testing (23%): This is the most basic test unit;
- Map testing (5%): These tests load portions of the game and check a small action, like the interaction between some actors and objects;
- Screenshot testing (1%): This type of test is used to check the rendering output;
- Performance testing (less than 1%): These tests monitor the frame rate of the game and eventual bottlenecks.

5.1.6 Findings and Discussion

The testing strategies should take into account the particularities of game projects wrt. traditional software: The majority of the researchers agree that game development has particularities compared to traditional software development, mostly because of the final product goal: software as a "productivity" product and video game as an "entertainment" product. These differences reflect on the way the process of developing a game is conducted, mainly with requirements and the addition of features along with the development. These particularities of game development should be taken into consideration when devising a testing strategy for a game project. Applying traditional testing strategies, e.g. high emphasis on unit testing using Test Driven Development (TDD), to a game project might not bring the expected benefits. Yet, new testing strategies should be the results of the convergence of well-known software testing strategies, like unit testing, continuous integration (CI), and TDD. Finally, these particularities do not justify using only manual testing.

Game testers should work alongside software testers to complement one another's skills: Game testers are mostly game-play testers, responsible for searching for bugs and assessing the game experience, i.e., the fun factor. Although they are usually not considered part of the development team and have poor working conditions, their feedback is fundamental to the success of games. In game development, testers are professionals with specific skill sets related to game testing, and they are not engineers in general. Therefore, they write, automate, or script source code-related tests. They deal with games as black-boxes. Thus, the test team should include engineers who can automate tests.

Automation in game development is overlooked, as it relies on manual human testers: Automation is often overlooked because of coupling, scope, randomness, cost, requirements, time,

domain restrictions, and non-reusable code. Yet, automation has many benefits: reproducibility, quality, bug reduction, better test focus, and game stability. In game development, the concern (and budget) mostly focuses on play-testing the game. Thanks to deep-learning algorithms, there are initiatives to train agents and playtest the games. It is still too early for proper adoption (i.e., using a standard solution), but it is indeed a step forward to mitigate the burden of the repetitive tasks that game testers have to do.

Search for the "fun-factor" and "balancing" the game are mainly executed by game-play testers: According to the academic papers, there are testing techniques exclusive to game development, mostly related to game-play testing and the search for fun or game balancing. However, these new techniques are poorly explained or too abstract, making them hard to implement and automate. Like regression testing or smoke testing, some techniques are known for their use in traditional software development. Still, in game development, the approaches are different, relying more on game testing. The search for the fun-factor is not the only thing that game testers do. They also use their skills and empirical knowledge to investigate the games intelligently. These techniques are hard to automate.

There is no one-size-fits-all testing process for game projects as the games differ greatly: Big studios with resources can develop testing processes tailored for their needs, well suited for specific games. Small studios rely on creativity and customized techniques to test their games. The lack of well-defined strategies is evident, even when using of-the-shelves engines like Unreal, developers must devise their own testing strategies. How small studios can create a testing process that fits their budget? Big studios can build tailored systems that are well integrated into their development pipelines. As game studios struggle to reach the release date, commonalities among different game types should be investigated, so that working strategies could be reused/tailored for specific games.

The acknowledged importance of testing by game studios should open the door for open-source initiatives: Although not common, there are advanced testing techniques adopted by big companies. All projects described in the gray literature implemented a different testing technique, so it can be better applied to its game type. This fact allows us to believe that developers are aware of the importance of discovering new techniques to test the game and not rely only on play-testing sessions. We observed that different studies apply ad-hoc game testing strategies, varying according to game genres (FPS, RPG, Sports, etc), game types (2D, 3D), revenue models (game-as-a-service), etc. The game industry should learn from traditional software initiatives and invest in sharing knowledge among game developers so that the field can grow faster.

Issues like lack of plan and poor testing coverage call for game testing to be performed early: The most common issues gathered from video-game postmortems indicate that game developers do not plan ahead for testing and, consequently, test coverage is low. Moreover, the specific requirements of the project are listed as the main cause of difficulties in adding proper testing during the development. Other problems are related to feedback from the testers and the project's scope. The non-linearity and randomness of most games nowadays make it hard to cover essential cases of a game. The scope—the world, the variables, the randomness, the paths—are too much to cover. Developers have concerns regarding the lack of testing. Testers need the game artifact in a "playable" version to perform their assessments. However, given the tight deadlines, this playable build is only accessible late in development. Game developers should strive to release incremental builds to game testers.

5.2 Automated Video Game Testing

In the previous section, we discussed what is video game testing. In this section, we discuss automated video game testing and its techniques. As the complexity and scope of game development increase, playtesting remains an essential activity to ensure the quality of video games. Yet, the manual, ad-hoc nature of playtesting gives space for improvements in the process. In this study, we investigate gaps between academic solutions in the literature for automated video game testing and the needs of video game developers in the industry. We conduct this study by (1) performing a *literature review* on video game automated testing techniques and (2) applying an *online survey* with video game developers. We asked them to assess some academic solutions on their desirability, viability, and feasibility.

5.2.1 Method

We present the method in two parts: the literature review, where we search for automated video game testing papers, and the survey with video game developers using an online form, where we ask the respondents to assess the solutions we found in the papers. The full list of papers and survey questions is with the support material at https://doi.org/10.5281/zenodo.5854809.

Snowballing the Academic Literature

The goal of the literature review is to search, identify, and catalogue automated video game testing techniques presented in academic papers. Instead of starting from scratch, we used a recent study [17] that already collected works about video game testing. The authors grouped 51 papers

according to the approach: search-based, goal-directed, human-like, scenario-based, and model-based. We decided to extend this work because (1) we found papers about video game testing that were not part of the original dataset, and (2) some of the works in the dataset have solutions too distant from the video game industry reality, making them hard to apply in real-life projects. Thus, to expand it, we performed full snowballing [89] and further exclusion criteria. We used the following criteria for the title and abstract reading:

- Inclusion criteria:
  - Paper must be about automated video game testing;
- Exclusion criteria:
  - Papers about formal or model validation;
  - Papers about gamification;
  - Papers about serious games;
  - Papers about using games for educational purposes;
  - Papers about using games for medical purposes;
  - Papers not written in English;

The final dataset consists of **166 papers** from 2004 to 2021. Among them, 81 are journal articles, 70 are conference papers, 12 are theses (masters and Ph.D.). There are also 1 report, 1 book, and 1 book chapter. We read all papers, considering five variables to include them or not:

- **Study type** (Theoretical/Applied): If the authors produced any practical solution or tool for testing.
- **Testing** (True/False): If the paper is about testing games.
- **Automated** (True/False): If the testing is somehow automated.
- **Machine Learning** (True/False): If the testing uses any machine learning model.
- **Test Objective** (String): The goal of the testing, i.e., *balancing* the game, *finding bugs*, etc.

From the full reading, we found 114 papers that present some sort of applied approach, that is, a solution or a tool for game testing. Among these, 80 used at least one automated step and **53** used some type of machine learning model. We use this filter to exclude papers that use manually written scripts.

Developer Survey

We divided the survey into four sections. The first section discussed the respondent *background*; the second asked them about their *manual playtesting* activities when developing a game; the third asked the participants to assess academic techniques/solutions for *automated playtesting*; and, the fourth contained optional open questions about the *future of game testing*.

Preparing the Questions: To reduce the scope of the survey, we focused on the three most common testing objectives: *balancing*, *exploration*, and *finding bugs*. To choose which paper (solution) we put on the survey, we considered four aspects (see below).

- Include papers that not only propose a solution but actually implemented it;
- Avoid papers that use frameworks or platforms that are deprecated;
- Include papers where the authors validated (or evaluated) their solution;
- Include papers that provide source code (replication package).

Putting the papers' solution in the survey: To explain the paper for the survey participants, we divide the papers' solutions into four parts, with a short description for each: the *GOAL* of the paper; *HOW* the authors accomplished that goal (the method); the role of *AUTOMATION* in the solution; and the final *RESULTS* of the paper. For example, for Paper #3 Gordillo et al. [90], we wrote:

> "GOAL: Testing coverage in complex 3D environments. HOW: Fully traverse the environments using autonomous agents (bots). AUTOMATION: Automate the collection of some playtest data. RESULTS: The agent can reach areas that should not be inaccessible."

For each question, we asked the participants to assess (a) **Desirability** – Is this something you would like to use for game testing? (b) **Viability** – Do you think it can be implemented in your workflow? (c) **Feasibility** – For you, do you believe this idea would bring benefits to game testing?

Sending the Survey: We sent the survey to online communities, groups, and forums for game developers and game testers, including on *Reddit*, *LinkedIn*, *Facebook*, *Discord*, and *Itch.io*.

5.2.2 Literature on Automated Game Testing

Difficulties on Game Testing Automation

There are significant differences between games and non-games regarding game developers' difficulties to write automated tests [87]. There is less automation in game testing, for different reasons shown in Table 11. Despite the difficulties in applying automating tests in game development, there are some benefits of doing it, for example, reproducibility, quality, fewer bugs, better test focus, game stability, and fewer human testers [81].

Table 11: Difficulties in test automation for games.

Difficulty	Description
Coupling	It is hard to write the automation given the coupling between the user interface (UI) and game mechanics [12, 67, 91].
Scope	Trying to cover all "paths" of the game could restrict the game design [12, 92], sometimes seen as contrary to agility preventing the fast pace of changes [81, 91]. Developers believe that they cannot cover everything, due to the large search space of possible game states [91], therefore the effort is not worth [81].
Randomness	The non-determinism in games (multi-threading, distributed computing, artificial intelligence, and randomness) make it hard to hard to assert the correct behaviour [12, 81]. In this case, is a "emergent software" as its randomness is a feature and players' surprise is desirable [47]. You need access to the randomness part of the game, otherwise, you may never reproduce the bug properly [93].
Changes	It is difficult to keep the automation as the game design change too often, even core mechanics [92, 94], making its documentation becoming obsolete too fast [47]. Also, the source code is temporary and highly likely to change as development progresses [81].
Cost	A software engineering is more expensive compared to gameplay testers [12, 81]. The cost of hiring new developers to write additional lines of code into a game to support automation (as well as writing scripts and other test utilities, interfaces, and so on) can be more than it would cost to pay human testers [93]. Also, manual re-test after code changes (new build) are expensive due to reproducibility issues [81].
Time	Programmers usually don't test their games. They usually don't have time to do so [93] or they believe that writing tests take time away from the actual development process [81].
Fun-factor	Capturing the "fun" with an automated test is not possible [81] as game testing is human-centric and human behaviour is difficult to automate [92]. Also, it is hard to automate games where many events are happening simultaneously. Therefore, automation is used mainly in simple tests [92].
Code	Automated testing code may not be bug-free, nor be reusable from one game to another, or from one platform to another [81]. Game developers have more difficulties than other developers when reusing code [87].

Solutions and Techniques for Automated Game Testing from the Literature

From our dataset of 166 papers, 80 of them suggested some types of applied solutions for video game testing. These solutions were either fully-automated (as mentioned by their authors) or

semi-automated. Finally, 53 papers used some types of machine learning models to train agents and playtest games.

In that set of 53 papers (Appendix A), the most discussed test objectives were *balancing the gameplay* (19 papers). The remaining ones were *game exploration* (11 papers), *finding bugs* (6 papers), and *player modeling* (6 papers). Also, testing the *game mechanics*, *UI*, *UX*, *visual correctness*, *collision*, and *visualization*.

According to Figure 8, automated video game testing is an emergent field. Even with a drastic decrease in 2021[55]. The majority of the studies were from the last 2-3 years, especially 2020 with 37 papers.

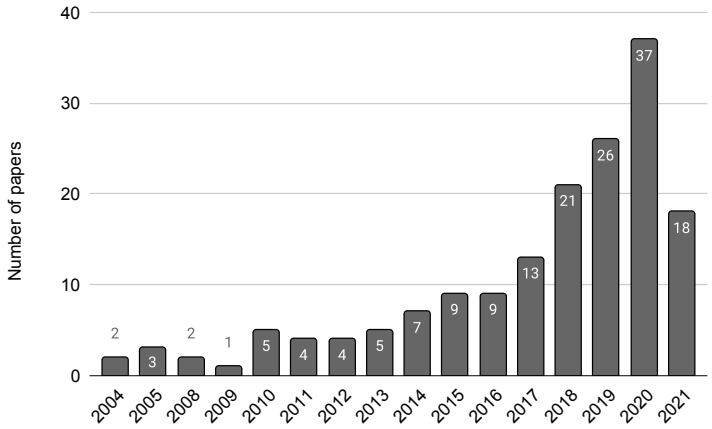


Figure 8: Histogram of all the 166 papers. There is clear rise of the subject "automated video game testing" in recent years.

The academic papers that used ML focused on the model instead of the game testing problem. Even filtering the papers that offer solutions and evaluations, their applicability in real-world scenarios does not seem viable. We identified some main issues with the papers:

- The tools (game engines, games, etc.) used in the experiment are too simple, incomplete, or academic projects.
- The testing objectives are not clearly defined, sometimes with phrases like "it can be used to test the game".
- There is no oracle or it is made manually *after* the autonomous agents play the games.
- The source code (replication packages) is often not available, which makes it impossible to replicate or re-use the proposed solutions.

[55]The study was made in November 2021.

Discussion

The recent rise of papers about automated testing in video games shows a trend in software engineering testing. We identified four main reasons: (1) the rise of machine learning models and solutions applied to every aspect of software development; (2) the achievements of machine learning models in playing video games; (3) all the new tools for writing and deploying machine learning models; and, (4) the need to automate testing games, as the scope, cost, and complexity of games keep on increasing.

Yet, only half of the studies we found offer some kind of applied solution for automated testing. Among them, we perceived a lack of focus on the real problem: the testing part. Few papers present testing oracles or even discuss them, relying on manual assessment. The testing goals are not clear enough, sometimes it is even hard to find in the paper what exactly the authors were trying to test. Also, the lack of replication packages makes the solutions hard to evaluate and reuse.

Finally, there is a segmentation of the video game testing papers. Our dataset contains papers that pertain to Artificial Intelligence, Computer Science, Software Engineering, and Video Game Design. These different fields have different goals. For example, AI papers focus on creating the models, while those on Software Engineering focus on testing concerns, like the oracles. A proper solution for automated video game testing needs all these fields of study working together. A new field of study and–or specific venues for discussing this matter need to arise.

5.2.3 Survey on Automated Game Testing Techniques

We had a total of 12 accepted responses. The majority (58%) of the respondents have more than four years of experience in game development. The same proportion reported working full-time in a game company. Almost all respondents have different roles, from software tester to game designer. All play video games as a hobby and the great majority (92%) have experience developing traditional software.

All respondents use manual playtesting regularly. Some of them test the game within the game engine. None of them use scripts to playtest games. Their testing objectives include *searching for bugs* followed by *exploring the game content* and *balancing the game mechanics*. Finally, *crash* and *stuck* behaviour is what the developers usually try to spot when testing followed by *graphical*, *collision*, and *performance* issues.

Solution #1: The solution #1 by Gudmundsson et al. [95], titled "Human-Like Playtesting with Deep Learning", uses autonomous agents to predict the difficulty of a new game level automatically. To train the agents the authors used Convolutional Neural Networks (CNN) in a grid-like structure (a Match-3 game called Candy Crush), using a discrete actions space. The same method

Table 12: Survey results related to the solutions (paper's ideas).

#	Solution	Test Obj.	Desirability			Viability			Feasibility		
			Yes	Not Sure	No	Yes	Not Sure	No	Yes	Not Sure	No
1	Gudmundsson et al. [95]	Balancing	6	5	1	2	5	5	5	3	4
2	Roohi et al. [96]	Balancing	3	6	3	1	8	3	1	6	5
3	Gordillo et al. [90]	Exploration	8	2	2	4	4	4	6	2	4
4	Ariyurek et al. [97]	Exploration	3	7	2	4	4	4	3	4	5
5	Zheng et al. [2]	Exploration	5	5	2	2	7	3	4	4	4
6	Pfau et al. [98]	Bugs	6	4	2	5	6	1	7	2	3
7	Ariyurek et al. [99]	Bugs	5	5	2	2	7	3	6	3	3

was used by AlphaGO[56]. Table 12 shows that this solution is desired and feasible for the respondents. Yet, it is not viable according to them. Among the reasons are the necessity of "lots" of data and the need for building the testing pipeline from scratch. Another problem is the time needed to train the agents. They also mentioned that Match-3 games are not as random as they seem: designers deliberately make choices to avoid players getting stuck.

Solution #2: The solution #2 by Roohi et al. [96], titled "Predicting Game Difficulty and Churn Without Players", uses gameplay data from autonomous agents and human playtesters to check the pass rate of new levels automatically. To train the agents, the authors used Deep Reinforcement Learning with the Proximal Policy Optimization (PPO). They tested a puzzle game using Unity ML-agents[57]. Respondents were more averse to this idea compared to the previous solution #1. Because the solution used players' churn data, they mentioned that it is hard to spot precisely why (and where) players abandon games. One respondent stated: "It's also something that has more value when captured during soft launch without much effort.".

Solution #3: The solution #3 by Gordillo et al. [90], titled "Improving Playtesting Coverage via Curiosity Driven Reinforcement Learning Agents ", uses autonomous agents to fully traverse complex 3D environments. To train the agents, the authors used Reinforcement Learning with the PPO algorithm. They tested a complex 3D environment using an undisclosed game engine. Compared to all others solutions, this was the most desired, as it deals with a situation that is lengthy to test manually. Some of the answers agreed with the approach for the exploration of edge cases, allowing "more obvious glitches be caught during manual testing".

[56] https://deepmind.com/research/case-studies/alphago-the-story-so-far
[57] https://unity.com/products/machine-learning-agents

Solution #4: The solution #4 by Ariyurek et al. [97], titled "Playtesting: What is Beyond Personas" uses autonomous agents with different personas (killers, explorers, etc.) to discover different paths at the level. To train the agents, the authors used Reinforcement Learning with the PPO algorithm. They used the General Video Game Artificial Intelligence (GVG-AI)[58] and VizDoom[59] frameworks to evaluated their solution. The respondents were not sure if they wanted this solution and were divided about the viability and did not think it is applicable in practice.

Solution #5: The solution #5 by Zheng et al. [2], titled "Wuji: Automatic Online Combat Game Testing Using Evolutionary Deep Reinforcement Learning", uses autonomous agents (bots) with different goals to explore game states and corner cases. The authors developed a testing agent called "Wuji", which uses Evolutionary Algorithms and multi-objective optimization to explore game space. They used Reinforcement Learning to direct the agent while exploring the state space. They evaluated the solution using an undisclosed MMOG. The respondents are not sure about the viability of this solution and are divided about its feasibility.

Solution #6: The paper #6 by Pfau et al. [98], titled "Automated Game Testing with ICARUS: Intelligent Completion of Adventure Riddles via Unsupervised Solving", uses autonomous agents to complete the game like a "speedrun"[60] and spot crashes/freezes and blocker (soft lock). To automate the process, the authors used the script language Lua on top of the Visionary game engine[61]. According to Table 12, this paper idea is the most feasible of all solutions and very desirable and viable.

> *"The biggest bang for the buck would be as a build acceptance test on a CI/CD pipeline, making sure they catch obvious blocking bugs. Otherwise, it drops significantly in usefulness"*. – survey respondent about paper #6 [98].

Solution #7: The solution #7 by Ariyurek et al. [99], titled "Enhancing the Monte Carlo Tree Search Algorithm for Video Game Testing", uses agents to generate sequences that can be replayed, to explore games and spot bugs. The authors modified the Monte Carlo Tree Search policy to use different strategies. They also used the General Video Game Artificial Intelligence (GVG-AI) in a 2D adventure game. The respondents reported viability as a problem. The respondents' concern is that the authors used pre-defined bugs: one respondent stated "You don't want to find the bugs you already know about".

[58] https://gaigresearch.github.io/gvgaibook/
[59] http://vizdoom.cs.put.edu.pl/
[60] A speedrunner aims to complete a game as quickly as possible.
[61] https://www.visionaire-studio.net/

Future of Video Game Testing

We also asked the game developers open questions about the future of video game testing. First, we asked *"What is the most important aspect of video game testing?"*. The answers varied:

- Identifying areas for improvement in the game;
- Helping make the game work as the players expect;
- Making sure everything works is secondary;
- When playing a game, it must feel right;
- Matching specifications (game design requirements) is not enough;
- Testing to check how players perceive the game;
- Testing to check the UX.

We also asked what could help game testers do their jobs: *"Currently, what is lacking in the video game industry that could help video game testers do their jobs?"*. Respondents mentioned easy-to-maintain test automation that is decoupled from the game under test. The lack of testing process and lack of engineer expertise were also mentioned.

> *"I think game testers would greatly benefit from learning standard software testing and engineering from the rest of the industry. A lot of the testing is done manually by non-technical people with no knowledge about game engines, backend services, graphics API and so on. This also applies to developers. While they are good at making games, they are terrible at engineering and don't follow good practices. I have never seen a unit test written in a game, for instance. It would also help if they were actually agile instead of doing waterfall cycles of months if not years."* – survey participant about what could help game testers do their jobs.

When we asked *"In 10 years, how do you think video games will be tested?"*, the respondents were skeptical. They believe the majority of game companies will still be using manual testing. Engineers will still be working mainly on the games instead of building testing tools.

Discussion

In general, the respondents were skeptical about the solutions of the academic papers. The solutions with a more straightforward process were better received, as the Solution #3 by Gordillo et al. [90]. Training agents to test games is seen as a "waste of time and money that can be spent somewhere else", which is a recurrent narrative in the video game industry. Building a complex testing pipeline requires dedicated software engineers, which cost more than manual testers. For small games (indie developers, for example) none of the presented seven solutions are suitable.

A respondent's recurrent concern is the up-front cost of building testing tools and training agents. There is a clear need for open-source, general tools that allow video game developers to test their games. These tools must work with a wide range of game types, without lengthy, complex customization. They should work directly with game engines, which are the main tools for all video game developers. For example, the Unreal engine (version 4) has built-in functionality tests. Yet, to test a feature requires doubling the developer's effort, like traditional unit tests. Game developers need an easier way to automate tests in their games.

Some respondents were concerned about AI replacing game testers. We believe that this is not going to happen soon. Even if better machine learning models could be built, spotting inconsistencies, and odd behaviours, and verifying the "fun" in games, need humans, who can do it trivially.

Chapter 5 Summary

In this chapter, we discussed the field of video game testing. Among the main findings, we found that automation is overlooked and game testing currently relies mostly on human testers. Also, the search for the "fun-factor" and "game balance" is mainly executed by game-play testers. Finally, there is no one-size-fits-all testing process for game projects as the game types differ greatly.

Game studios acknowledge the importance of testing but also have issues like lack of plan and poor testing coverage. The results of the literature review show a rise in research topics related to automated video game testing in recent years. Yet, most testing tools and frameworks are more concerned with the performance of the machine learning models instead of the testing objective. The survey results show that game developers are skeptical about using automated agents to test games.

We conclude that automating the video-game testing process is the natural next step. However, developers and researchers lack processes, frameworks, and tools to help them with test automation. It results in ad-hoc techniques that are hard to generalize on different game

types. Yet, testing is the key to quality games, and automation makes game quality sustainable. In the next chapter, we present our approach to using automated testing video games using machine learning models.

Chapter 6

Automating Video Game Testing to Balance the Game

"The intention behind the high difficulty of the games I direct is to evoke a feeling of joy and accomplishment in the player when they overcome these challenges."

HIDETAKA MIYAZAKI, FROM SOFTWARE GAME DIRECTOR

6.1 Introduction

Game development is an interactive process [56, 100]. Usually, game developers start with a core mechanic, limited in scope, and then iterate, adding new features until the game is "complete". They use experimentation and trial-and-error to find the fun and balance the game mechanics. Thus, developers (and–or game testers) play their game until they "feel" that it is correct. These constant changes are the main reason there are no clear requirements but a "vision" for game projects.

During this interactive process, alongside the creation of the game mechanics, developers must ensure that the changes reflect what they intend. For example, improving the graphics while not degrading the performance. Keeping the game challenging while avoiding boredom requires a testing process called *balancing* [1], which relies on empirical knowledge that is hard to translate into the actual game specification.

Schell [1] list 12 common types of balance in video games (Appendix B). In this thesis, we focus on two balance types: *Challenge vs. Success* is about keeping the player engaged considering the game difficulty and the player's skills (Figure 9); *Skill vs. Chance* refers to games in which success depends more or less on luck instead of the players' skills. Games of skill are more like athletic contests (which player is the best?) while games of chance are more casual, as much of the

outcome is decided by fate. For example, dealing out a hand of cards is a pure chance but choosing how to play them is pure skill.

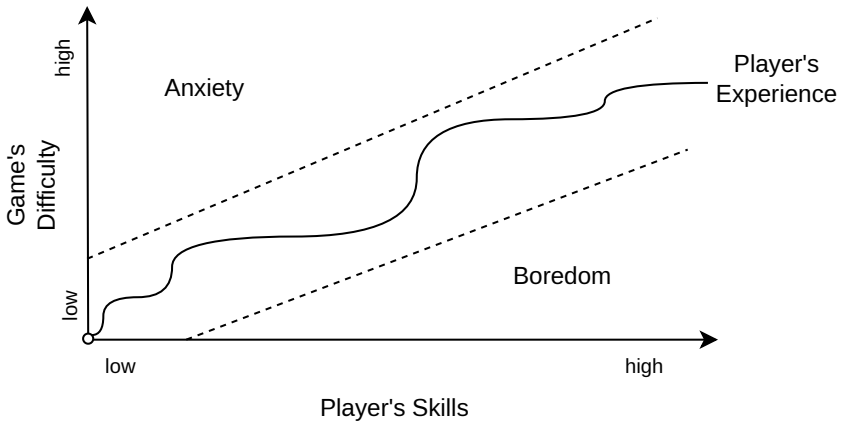


Figure 9: The relation between game difficulty and player's skill. Adapted from Schell [1]. If play is too challenging, the player becomes frustrated. But if the player succeeds too easily, they can become bored. Keeping the player on the middle path means keeping the experiences of challenge and success in proper balance.

For every new feature or change in the game attributes, the game must be tested again to check if it now is too easy, too hard, impossible to complete, etc. This process is manual, slow, not scalable, and costly. As developers perform multiple changes or permutations thereof, it quickly becomes overwhelming to keep track of what works best for the game. In summary, this testing process demands much effort to produce a quality game.

In this chapter, after studying game problems, game engines, and game testing, we implement an approach to enhance game testing to balance video games. Instead of manually testing games (playtesting), we propose an automated approach with autonomous agents to aid game developers assess the game's balance.

Similar to unit tests on continuous integration pipelines in traditional software development, with which a system warns developers when a test fails, we want to provide developers with an automated process that would warn them when a new version of their game is too far from their ideal balance. This automated process must help game developers by (1) providing fast feedback about the balance state of the game and (2) testing multiple game attributes at once.

We believe in treating the game development as an iterative process, similar to the use of Continuous Integration (CI) pipelines. For instance, the developer writes a unit test for each new feature. By adding these tests to the CI system the developer automates its execution. Thus, for each new feature, the CI system warns the developer if there is any failure with the previous tests. Instead, in our case, we write integration tests that warn the game developer if there is an issue with the game balance. To do so, we use autonomous agents to aid the process of testing to balance the

game mechanics. We use autonomous agents because (1) scripts are immutable and keeping them up to date with the game modifications requires much effort; and (2) agents play differently every time, which helps to explore different game states.

We build the autonomous agents with Deep Reinforcement Learning models and train them to master the game. Because games are big in scope, we use a scenario representing a chunk of the game. This scenario has different game attributes. In unit testing, the oracle is well defined and immutable. For balancing the game we need to be less strict because the balance of the game depends on the game developer's vision, which is subjective. Therefore, we use metrics from within the game to assert its balance. For example, we verify if the score is too high or low and if the random agent is performing better than the others.

With this automated process, developers can quickly test different attributes of their game automatically and in parallel. It provides evidence to developers on which changes benefit their game. It thus saves time, reduces manual effort, reduces the cost, and also helps improve the game's quality.

6.2 Related Work

There is a large community focusing on customizing models to master the game (play as good as a human) [101]. As for example, the most recently DeepMind's Agent57, which is a customized model[62]. Yet, for testing the game, the authors focus on more simplistic models, as seen in Table 13, like PPO and A2C.

Proximal Policy Optimization (PPO) is a Reinforcement Learning algorithm "which alternates between sampling data through interaction with the environment and optimizing a 'surrogate' objective function using stochastic gradient ascent" [102]. The algorithm aims to maximize the probability of a set of actions being taken by the agent, given these actions make the agent get rewards above average during its interaction with the environment. PPO is an on-policy algorithm, meaning it learns by comparing the current set of actions taken with the previous one, without using a replay memory.

A2C is a synchronous variant of Asynchronous Advantage Actor-Critic (A3C) that uses agents running in parallel to explore different parts of the environment [103]. By doing so, the algorithm does not need to use a replay memory. Similar to PPO, after reaching a terminal state (game over) or a maximum number of actions, the algorithm updates its policy, which is the function that generates the set of actions to be taken by the agent. This update is done to make the policy more likely to generate actions that will lead to high rewards.

[62] https://deepmind.com/blog/article/Agent57-Outperforming-the-human-Atari-benchmark

Table 13 shows all the papers that, to some degree, talk about video game testing, automation, and balancing. These papers lack validation of the proposed approaches, i.e., they only propose theoretical solutions or the solutions focus on the modelling and training of autonomous agents, leaving the training/assessment of the game with few details. None of the papers provided the source code. Also, only a few of them explain the process of training the agents in some detail.

Among the game balancing papers, we found some that deal with the difficulty of the game, the balance between multiplayer, Bugs, and exploration of the game states. There are three main related works that deal with the game difficulty. Isaksen et al. [104] verify how different attributes of the same game (without changing the rules) affect the game difficulty. The authors use metrics to predict the score. Gudmundsson et al. [95] test the difficulty of a game level using autonomous agents (bots) to simulate gameplay and the "success rate" with human players. They want to predict the difficulty of a new game level automatically. They claim the difficulty of new levels could be tested automatically. Roohi et al. [96] predict if the player can win and abandon the game in new levels. They use gameplay data from autonomous agents (bots) and playtesters to train agents to play the game autonomously.

Other three important studies discuss the balancing in multiplayer games. DeLaurentis et al. [16] define a framework that predicts the game balance using data from autonomous agents playing against each other. Pfau et al. [105, 106] use data from real players, playing against computer enemies, to replicate human play behaviour.

> The previous works mainly aim to create autonomous agents based on ML/AI models to master the game but do not try to incorporate the agents into the game development process. Most game studios, especially the small ones, do not have the time or the budget to adopt complex and costly solutions like these, but they can benefit from a more feasible approach. Therefore, instead of thinking of game testing as an isolated process, we aim to the process of finding the right balance of the game.

6.3 Approach

Figure 10 shows the feedback loop of manual game testing and how our approach aims to automate it. The game developer starts the process of checking the new *Game Attributes* by making major or minor modifications to the game. Thus, producing a new build (version) of the game. It can be, for example, introducing a new game mechanic, like "the ability to fly", or simply tuning the strength of the attack or character speed.

The game is then divided into scenarios, which are isolated chunks of the game. The standard way to test the game is by performing it manually and receiving the feedback in free-form text.

Table 13: Works that deal with the automation of game balancing. *PvC* and *PvP* mean player versus computer and player versus another player, respectively.

Approach	Model	Engine	Game	Oracle	Test Goal	Mode
Isaksen et al. [104]	Custom	NA	Platformer (Flappy Bird)	Survival Analisys (hazard rate)	Difficulty	PvC
Gudmundsson et al. [95]	CNN	NA	Puzzle - Match-3 (Candy Crush)	Human and Agent success rate	Difficulty	PvC
Roohi et al. [96]	PPO	Unity engine	Puzzle - Match-3	Pass rate and churn rate	Difficulty	PvC
DeLaurentis et al. [16]	CNN and MCTS	microRTS	RTS multi-player	Win/Lose rate	Balance	PvP
Pfau et al. [105]	ANN	NA	MMO (Aion)	Win/Lose, Fight duration, remaining Health	Balance	PvC and PvP
Pfau et al. [106]	ANN	NA	MMO (Aion)	Win/Lose, Fight duration, remaining Health	Balance	PvC and PvP
Garca-Snchez et al. [107]	Evolutionary algorithm	NA	CCG (Heartstone)	Recommend decks	Exploration	PvP
Aponte et al. [108]	Q-learning	NA	Pacman clone	NA	Difficulty	PvC
de Mesentier Silva et al. [109]	Custom	Boardgame	Ticket to Ride Europe	NA	NA	PvP
de Mesentier Silva et al. [110]	Custom	Boardgame	Ticket to Ride USA	NA	NA	PvP
Guerrero-Romero et al. [111]	Not clear	GVGAI framework	NA	NA	NA	NA
Mugrai et al. [112]	Genetic algorithm and MCTS	NA	Puzzle - Match-3	Max and Min scores	Difficulty	PvC
John and Gow [113]	NEAT	Unity engine	Fighting (Divekick)	Log player actions and survey	Bugs	PvP
Sriram [114]	Unity-ML Agents	Unity engine	Platformer	NA	NA	PVC
Shin et al. [115]	MCTS and CNN	NA	Puzzle - Match-3	NA	Difficulty	PvC

In our approach, we start a parallel process that trains the agents to autonomously play the game while measuring game metrics related to its balance, that is, *Balance Metrics*. These metrics are variables that work as a proxy to the agent's performance and, therefore, the game balance. They vary for each game but, in general, are related to the score of the game. For example, how many monsters were killed, the time to complete the level, the number of lives, etc.

To summarize, *Game Attributes* are the dependent variables that modify the builds of the game, while *Balance Metrics* are independent variables that "warn" the developer about the game balance.

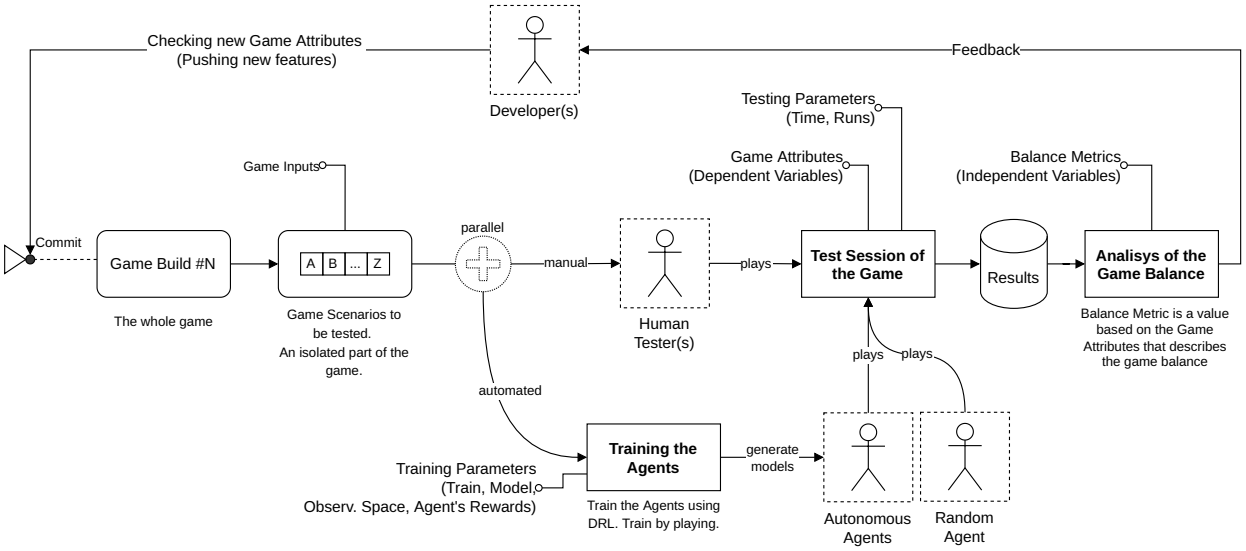


Figure 10: Our method to automate the game testing and feedback loop during the development.

Depending on the modification of the game, it is necessary to re-train the agents for each new build. For example, when a new game mechanic heavily modifies the gameplay. The training process varies according to the game. We also train the agents with different skill levels, so we can better assess the game. Aside from the agents, we add a random agent and humans with different skill levels.

Finally, the game developer quickly receives the feedback containing this information about the agent's performance. With this data, the game developer can (1) check the balance of the game between all the builds, (2) check the difficulty among the player skill level, and (3) check game design inconsistencies using the random agents.

6.3.1 Game Balance

For each game, we define one or more *balance metrics*. This value represents how well the agent (and human) plays the game, that is, their performance in the game. To balance the *Challenge vs. Success* we check spikes on the *balance metrics* when the agent plays in each game version. Also,

we compare the performance of novice and professional skill levels.

To balance the *Skill vs. Chance* we want the game to reward the player's choices rather than reward random inputs. To do so, we compare the random agent with the other trained agents. If the random agent is performing better, the game has a balance problem.

We use different game attributes to modify the game difficulty. To do so, we use the *Doubling and Halving* balancing methodology [1]. It says the developer should change the game attributes by high amounts: doubling or halving them. The objective of this method is to change something so that you can actually feel the difference right away.

6.3.2 Testing Scenario

Video games have a bigger scope, either in length, quantity, or time demanded to complete the game. For example, the game *Elden Ring* has hundreds of monsters and dozens of main "bosses" that are enemies of the player, all spread out in an open world[63]. All of them must be assessed to provide a proper experience to the player. However, developers can divide the game into chunks, where they are isolated and tested. This strategy is similar to what Rare does with the game Sea of Thieves (Section 5.1.5). They use single scripted actions to verify the object's behaviour, like opening a door, for example. In our case studies, we use a scenario representing a chunk of the game. This scenario has different attributes according to the game and game versions.

6.3.3 Training the Agents

Training agents to play games with Deep Learning (DL) or Deep Reinforcement Learning (DRL) is an ever-increasing research area. From the seminal work by Mnih et al. [116] to the recent improvements of Badia et al. [117], machine learning models allow autonomous agents to master games. Video games offer an environment with reduced scope (compared to real life) that suits the training of autonomous agents. These approaches show great success in mastering simple [101] and complex[64] games.

Playing the game is a sequential decision-making process, where the players continuously make decisions and take actions based on received observations. This problem can be modeled as a Markov Decision Process (MDP) [2] (Figure 11). MDP consists of five elements. The *agent*, which interacts with the *environment* using a *policy*. The *state*, which is the observation (representation) of the environment. The *action*, a set of possible decisions (move, attack, jump, etc). Finally, a *reward*, is the feedback we use to measure the success or failure of the agents' actions in achieving some goal (winning, surviving, etc)

[63] https://en.bandainamcoent.eu/elden-ring/elden-ring
[64] https://openai.com/projects/five/

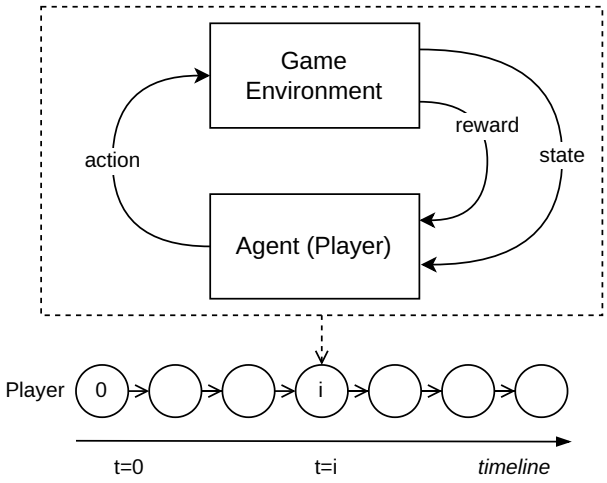


Figure 11: Markov Decision Process (MDP) adapted from Zheng et al. [2].

6.4 Implementation

One of the issues we saw in the related works was how to separate the concerns in the testing architecture. We usually noticed that the game code was mixed with the training code (Machine Learning code). Plus, the code for the AI model was also the testing code. The first thing we did was to separate each class and make them responsible for only one job. Figure 12 shows the UML-like diagram of our architecture.

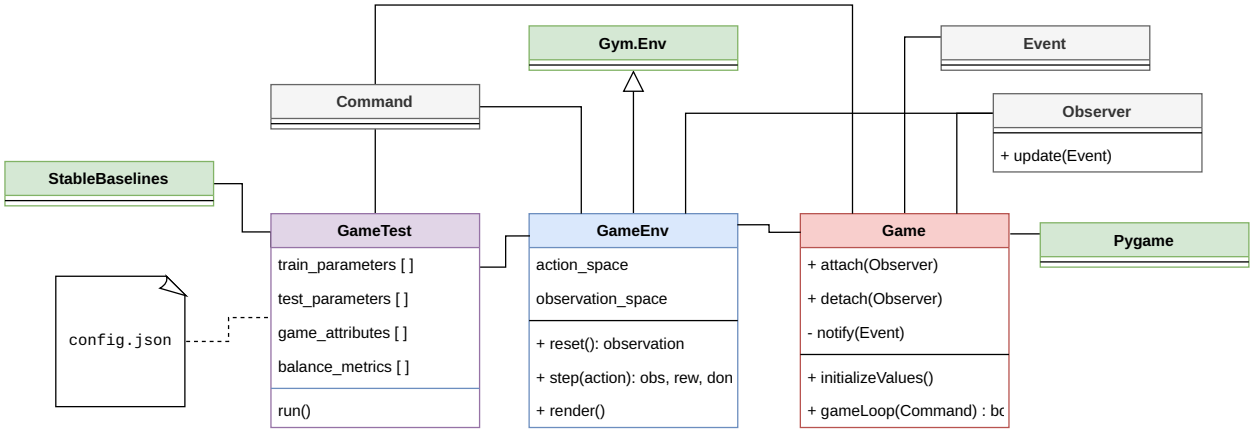


Figure 12: The Game Testing architecture.

The `Game` class uses *Pygame*[65] as a game engine, which is a Python framework that provides basic functions to help game development. Pygame uses the game loop structure: starting dealing with the *inputs* from the player, *updating* on the game entities that changes the game state, and *rendering* the new frame of the game.

[65] https://www.pygame.org

To train the agents we use a Python library called *Gym* which provides a framework to create machine learning environments. It also provides a number of already defined environments, usually used to assess new machine learning models. To train the agents for the games that we chose we needed to define a custom environment from scratch. This means we have to define an *action space* with the commands that are possible within the game, and a *observation space*, which is a set of variables representing the game state. The latter depends on the *reward functions* (see `GameEnv` class). Finally, we also use *Stable Baselines* library, which works on top of Gym, proving a selection of machine learning models, like *PPO*, *A2C*, and *DQN*, to train the agents to play the game.

The `GameTest` class is responsible for training the agents, running the game, and logging the results. The `config.json` file defines how the test will run. There we put the *testing parameters*, the *training parameters*, the *game attributes*, and the *balance metrics*.

6.4.1 Type of the Games

For this thesis, we decided to focus on one type of game: *Platformers*. Platformers are a video game genre where players control a game character to jump between platforms while avoiding obstacles and enemies that can kill them. They are notorious for being difficult and their designs can sometimes infuriate players rather than provide fun [118]. We believe this type of game fits well in our purpose of assessing the game balance.

6.4.2 Testing Scenario

Table 14 shows the *Testing Parameters* used in the game test scenarios. The player plays the game for *N* seconds, *M* times. Either a human or an agent plays the game. Each one is divided by skill level, with the exception of the random agent. The human professional is someone with gaming experience while the novice is not a regular game player.

Table 14: Testing parameters used to define the testing scenarios.

Parameter	Type/Value	Description
`time`	Integer	The time to be played in seconds
`run`	Integer	Number of runs
`session`	{human, ai-play, random}	The player of the session
`skill`	{novice, professional}	The skill level of the player
`build`	String	The game builds (versions)

6.4.3 Training the Agents

We divided the skill level of the agents by training time, that is, the *novice* is trained with *100K steps* while the *professional* with *1M steps*. The machine used to run the training has a CPU Core i7 2.6 GHz, A GPU NVIDIA GeForce RTX3070, 32 GB DDR4 or RAM, and an SSD hard drive. As for the software, we use the Python language with the Stable Baseline library running on Windows with NVIDIA CUDA.

Table 15 shows the *Training Parameters* we use to train the autonomous agents. For our experiments we chose to use the model-free DRL `models` because (1) in our game environment, we cannot predict state transitions and rewards, and (2) the training cost (computation time) is lower. Thus, we chose two different models to train the agents: *PPO* and *A2C*. The `action space`, `reward function`, and `observation space` vary from game to game.

Table 15: The training parameters used to train the autonomous agents.

Params	Type/Value	Description
`train`	Boolean	Re-train or not the agents
`model`	String	The machine learning model used to train the agents (PPO, A2C, etc)
`action space`	Array	String indicating the action of the game (LEFT, ATTACK, etc)
`reward function`	Float	Values, positives and negatives, defined by an heuristic
`observation space`	Array	The state of the game, what the agent knows and "see"

6.5 Case Study A. Batkill

6.5.1 About the Game

According to our architecture (Figure 2), we modified an open-source game[66], a 2D action platformer called Batkill[67]. We did not touch the rules of the game. It consists of a single screen, where the character tries to stay alive while bats fly towards him. The goal is to kill as many bats as possible without being hit. For each bat killed, the player gets one point (`+1 score`). For each hit taken, the player losses one life (`-1 life`). The bats spawn faster as they are killed by the players. The bats spawn in random locations. The actions are `LEFT, RIGHT, ATTACK, JUMP`. Figure 13 shows the game and its actions.

[66] https://github.com/polako/batkill
[67] https://github.com/python-aficionado/batkill

Figure 13: The character actions of the game Batkill. From bottom to top: *standing*, *attacking*, *running*, and *jumping*.

6.5.2 Test Scenario and Balance Metrics

Below we list the game attributes and balance metrics we use to create five different builds for the game Batkill. The player (or agent) plays the game for 180 seconds, two times. Either a human or an agent plays the game, divided by skill level: *novice* or *professional*, with the exception of the random agent. We use the median of the two runs to calculate the score.

- `bats`: {2,3} - Number of bats on the screen
- `bat_speed`: {3,6} - Movement speed of the bat
- `attack_cooldown`: {10,15} - Time between the character's attacks
- `jump` - Boolean - If the character can jump or not

To measure the balance of the game Batkill we use the balance metric `score`, which is `bats_killed - hits_taken`. To balance the *Challenge vs. Success* we check spikes of the `score` from the agents in each game version. Also, comparing their performance on novice and professional skill levels. To balance the *Skill vs. Chance:* we use the results from the random agent and compared them with the `score` of the other agents. If there is any build where the former has better performance, it is a hint that the game has balance issues.

- `bats_killed`: {0,1,2...} - Number of the bats killed

- `hits_taken`: {0,1,2...} - Number of the hits taken
- `score`: `bats_killed` - `hits_taken` - Rate between kills and hits

We simulate the development process by considering five different versions of the game (Table 16). They are also incremental, i.e., the changes in build#1 are carried to the other versions. For some of them, we re-trained the agents because the gameplay changes were significant, like adding a new bat on build#2 and adding a jump on build#5.

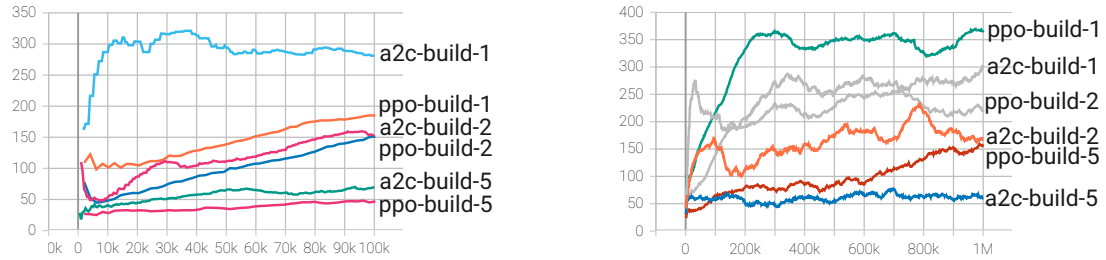
For every run we measure the `bats_killed` and the `hits_taken` by the character. In this test scenario, it is expected the difficulty of the game to increase over the builds. Also, the jump on build#5 is expected to add one more way to avoid getting hit.

Table 16: The Game Builds (versions) - Batkill

Build	Bats	Bats' speed	Attack time	Jump?	Train?
#1	2	3	10	FALSE	TRUE
#2	3	6	10	FALSE	TRUE
#3	3	6	10	FALSE	FALSE
#4	3	6	15	FALSE	FALSE
#5	3	6	15	TRUE	TRUE

6.5.3 Training the Agents

The Figure 14 shows the reward graphs for the PPO and A2C. The former has better performance (bigger reward) in all builds when the steps are more than 100K. For the PPO model, training a novice agent takes around six minutes while the professional takes around one hour. A2C takes about 10% more time to do the same training process.



(a) Skill level: Novice (100K steps of training). (b) Skill level: Professional (1M steps of training).

Figure 14: Training results for the game Batkill.

To train the game Batkill we use the following set of rewards:

- `BAT_KILLED +5`: the biggest reward when the agent kills a bat.

- `HIT_TAKEN -5`: the biggest negative reward when the agent gets hit.
- `ATTACK -0.1`: we use this small negative reward to suggest the agent using the attack too often.
- `JUMP -0.2`: after seeing humans playing the game, we verify that they do not jump often. That is why we penalize the agent if he jumps. The idea is to keep the character on the ground more time.
- `MOVING_TOWARDS +0.1`: we give a small reward so the player moves towards the bat.
- `FACING_NEAREST_BAT +0.2`: we give a small reward if the agent is facing the nearest bat. This is a movement humans do naturally and we try to hint here.

For the observation space, the state of the game, that is, what the agent knows and "see", we defined this set of variables:

- `player_x`: the character horizontal position.
- `player_y`: the character vertical position.
- `player_direction`: the direction the character is facing (left or right).
- `player_facing_bat`: if the player is facing the bat.
- `player_attack`: if the player attack (True or False).
- `player_cooldown`: the time between the attacks.
- `bat_alive`: if the bat is alive.
- `bat_direction`: the direction the bat is facing (left or right)
- `bat_x`: the bat horizontal position.
- `bat_speed`: the bat speed.
- `bat_distance_to_player`: the bat distance to the player.
- `bat_in_attack_range`: if the bat is in attack range.

6.5.4 Results

In this section, we describe the results of the agents and humans playing the game on different versions. We analyze the balance metrics `score`, `bats_killed`, and `hits_taken`. We compare them between the builds, agents and players, and skill level.

Skill Level: Professional

Figure 15 shows the results of the human professional playing the game. Build#1 is the easiest because the agents and human players had the best score. The random agent did surprisingly well, performing better than the models on build#3, close to the human level. This means that the game, in that state (3 bats with more speed), does not rely on the player's skill but on "button-mashing" techniques. Therefore, builds#3 seems to be the worst game state in terms of balancing. On build#4, the agent cannot spam attacks because the time between attacks is bigger. Thus the random agent performs worst than on build#3. On build#5, however, when the jump is added, PPO and A2C agents surpass the random agent. This means that the new feature is rewarding more the player's skill.

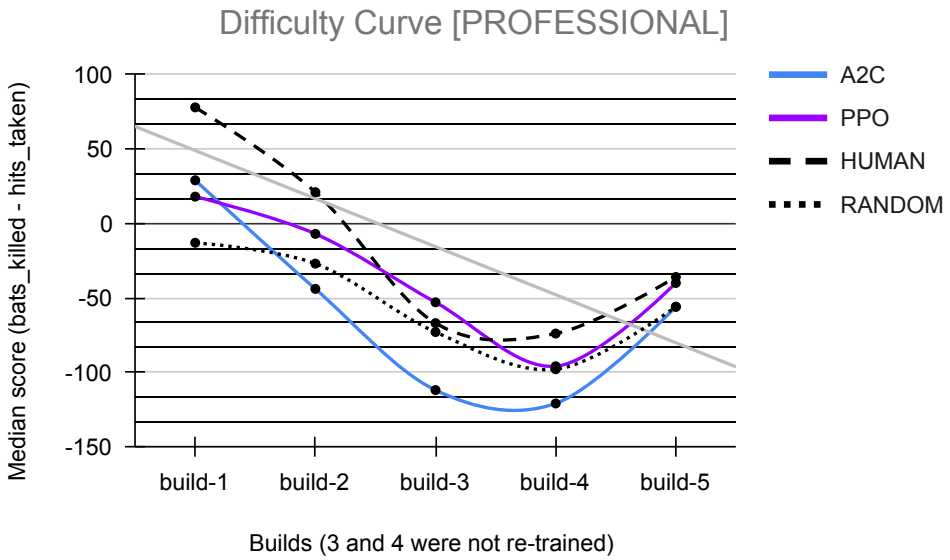


Figure 15: Difficulty Curve between all Professional agents on Case Study A. Batkill.

PPO agent on build#3 outperforms the human. On all the other builds the human performs better than any other agent. The A2C agent had the worst score, even if compared with the random agent. PPO agents follow a similar pattern to the human player across all the builds. This means that the agents perform similar to a human player, although not as good in terms of score.

Skill Level: Novice

Figure 16 shows the results for the novice agents and human players. Considering the human novice, the difficulty curve follows the same pattern as the professional version. The build#1 is the easier. The game gets harder on builds #2, #3 and #4. On build#5, when we introduce the jump, there is a significant improvement in the score. On build#3, humans had the worst score compared

to the random agent and a similar score on build#4. In those builds, the game is too hard and does not favour the player's skill. PPO model performed well with low training, surpassing the novice human on builds #2, #3 and #4. A2C had the worst results, even if compared with the random agent except on build#1.

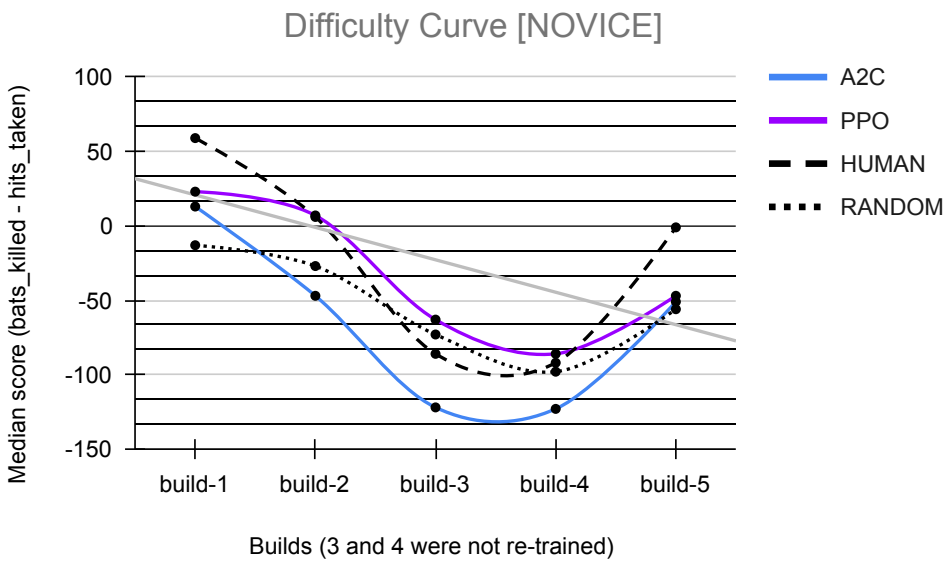


Figure 16: Difficulty Curve between all Novice agents on Case Study A. Batkill.

Novice vs. Professional

On the build#2, the human player killed more, because of the one extra bat added. However, there is a big spike in hits taken. When we increase the bat speed on build#3, the player had another spike on hits taken and fewer bats killed. The worst performance was in build#4 when we increase the time between the character's attacks. Finally, when we add jump on build#5 the player could avoid damage, similar to what he had on build#2, but killed fewer bats. This is because of the play style, either you play aggressively attacking more than avoiding damage, or you chose to jump and attack when it is safe.

The human novice got better results on build#5 when we compare it with the human professional. However, as Figure 17 shows, the novice had fewer kills and fewer hits on all builds. On build#5, the novice human killed less but took much fewer hits too. This is because the jump allows the player to choose between playing carefully and more aggressively.

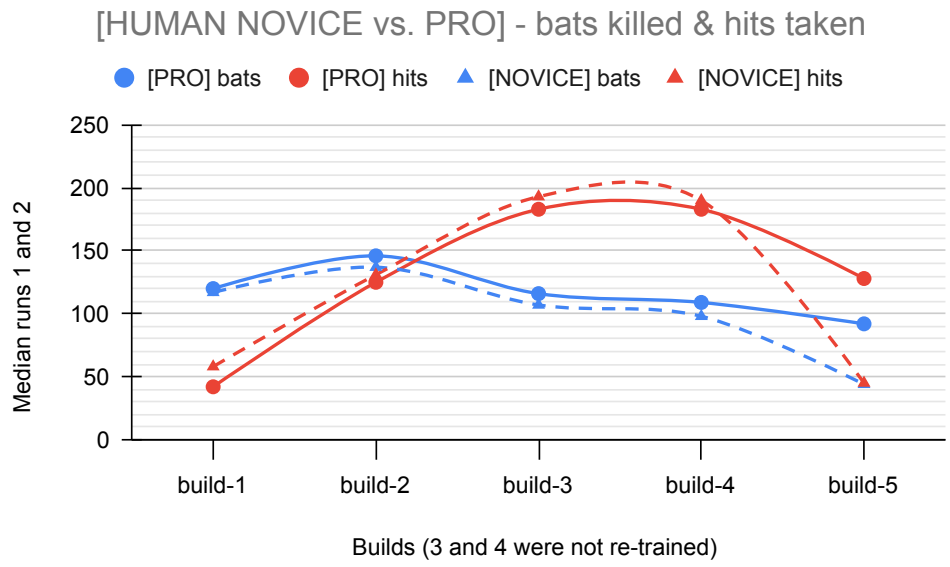


Figure 17: Bats killed and Hits taken for Human novice and professional on Case Study A. Batkill.

Summarizing the Results - Case Study A. Batkill

- The PPO agent's performance follows a similar pattern as the human players.
- The build#1 is the easiest while the build#4 is the hardest, followed by builds #3, #5, and #2. There is a big spike in difficulty on builds #3 and #4.
- The good performance of the random agent on builds#3 and #4 shows that these builds do not favour the player's skill. The introduction of jumping helped make the game more skill-based and reduce its difficulty.
- The addition of a new gameplay feature (the ability to jump), shows that humans can take different strategies when playing (aggressive or caution in our case). This hints to the importance of using multiple balance metrics to measure the game.
- Compared to PPO, A2C agents got better rewards during the novice training (100K steps) but worst results when the agent plays the game.

6.6 Case Study B. Jungle Climb

6.6.1 About the Game

We modified an open-source, 2D "infinite-runner" platform game[68] called Jungle Climb[69]. It consists of a single screen, where several platforms are drawn. Each platform has gaps randomly generated. The player character is initially placed below the platforms and has to climb them by jumping between gaps. After passing through the second platform, the screen will start scrolling upwards. The objective of the player is to keep the character below the bottom line of the screen as long as possible, not letting the player be "scrolled down" for too long with the screen.

For each step, the player is able to survive while above the second platform, it gets one point (`+1 point`). The scrolling speed of the platform increases as time passes. The actions are `LEFT`, `RIGHT` and `JUMP`. Figure 18 shows the game and its actions.

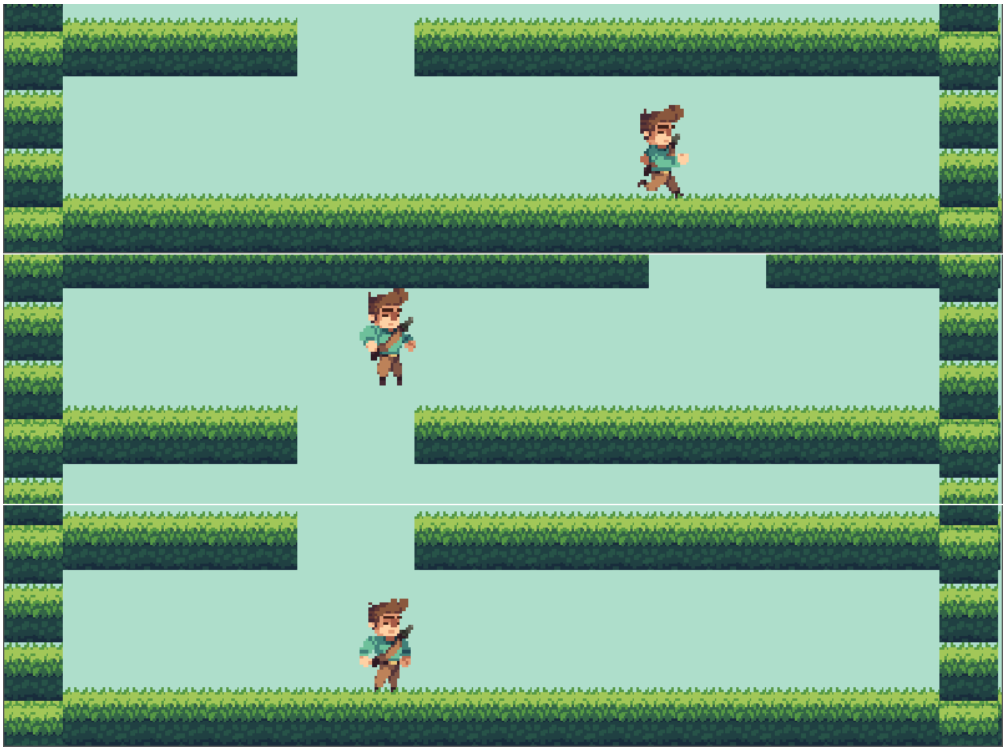


Figure 18: The character actions of the game Jungle Climb. From the bottom to up: *standing*, *jumping*, and *running*.

[68] https://github.com/gamedev-studies/jungle-climb
[69] https://github.com/elibroftw/jungle-climb

6.6.2 Test Scenario and Balance Metrics

Below we list the game attributes and balance metrics we use to create three different builds for the game Jungle Climb. The player (or agent) plays the game for 180 seconds, two times. Either a human or an agent plays the game, divided by skill level: *novice* or *professional*, with the exception of the random agent. We use the median of the two runs to calculate the score.

- `shift_speed`: {1,2} - The rate the screen scrolls up.
- `max_gaps`: {1,2} - The maximum number of gaps in each platform.

To measure the balance of the game Jungle Climb we use the balance metric `score`, which is `max_points + (max_correct_jumps * 100)`. To balance the *Challenge vs. Success* we check spikes of the `score` from the agents in each game version. Also, comparing their performance on novice and professional skill levels. To balance the *Skill vs. Chance:* we use the results from the random agent and compared them with the `score` of the other agents. If there is any build where the former has better performance, it is a hint that the game has balance issues.

- `max_points`: {0,1,2...} - Maximum number of points the while playing, that is, show how long the character was alive.
- `max_correct_jumps`: {0,1,2...} - The maximum of correct jumps, that is, when the jump connects to the next platform.
- `max_correct_jumps`: `max_points + (max_correct_jumps * 100)` - Rate between points and correct jumps.

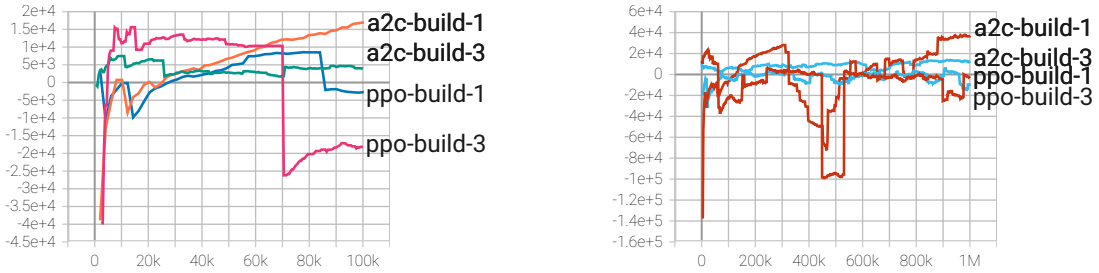
We simulate the development process by considering three different versions of the game (Table 17). They are also incremental, i.e., the changes in build#1 are carried to the other versions. For some of them, we re-trained the agents because the gameplay changes were significant, like adding a new gap for every platform on build#3. For every run we measure the `max_points` and the `max_correct_jumps` by the character. In this test scenario, it is expected the difficulty of the game to increase over the builds. Also, the new gaps on build#3 are expected to ease the climb.

Table 17: The Game Builds (versions) - Jungle Climb

Build	Shift Speed	Max Gaps	Train?
#1	1	1	TRUE
#2	2	1	FALSE
#3	2	2	TRUE

6.6.3 Training the Agents

Figure 19 shows the reward graphs for the PPO and A2C. Different than case study A, A2C got more rewards in both, novice and professional training sessions.



(a) Skill level: Novice (100K steps of training). (b) Skill level: Professional (1M steps of training).

Figure 19: Training results for the game Jungle Climb.

To train the game Jungle Climb we use a set of rewards. At the beginning of every step, we compute the `Below Threshold Reward (BTR)`. If the character has not passed the threshold of the screen where it starts to shift, the score will always be zero and the BTR will be greater than zero. We use this value so we can give the agent different motivations so it can exit this initial state more quickly: `BTR = time_elapsed * 5 if score == 0 else 0`.

We check if the character is not under the gap and, if the distance between the character and the gap is decreasing, we give `+100` and `-100` if increasing.

If the character is under the first gap, we check first if he is facing the second gap (in the upper row), and give `+100`. If the character's position on Y-axis is decreasing, the reward is `+100 + BTR`, otherwise, `-100 - BTR`. Finally, we give a penalty to the agent so he can get out of the first platform. Thus, if the `BTR` is bigger than zero and the character repeats the previous step, we give `-100`.

6.6.4 Results

In this section, we describe the results of the agents and humans playing the game on different versions of the game Jungle Climb. We analyze the balance metrics `score`, `max_points`, and `max_correct_jumps`. We compare them between the builds, agents and players, and skill level.

Skill Level: Professional

Figure 20 shows the difficulty curve considering the *score* for *professional* skill level. Considering the evolution across the builds, the build#1 seems to be the easiest to play. There is a spike of

difficulty on build#2. The build#3, which has more gaps in the platforms, does not reduce the difficulty of the game, as the score remains the same as the build#2.

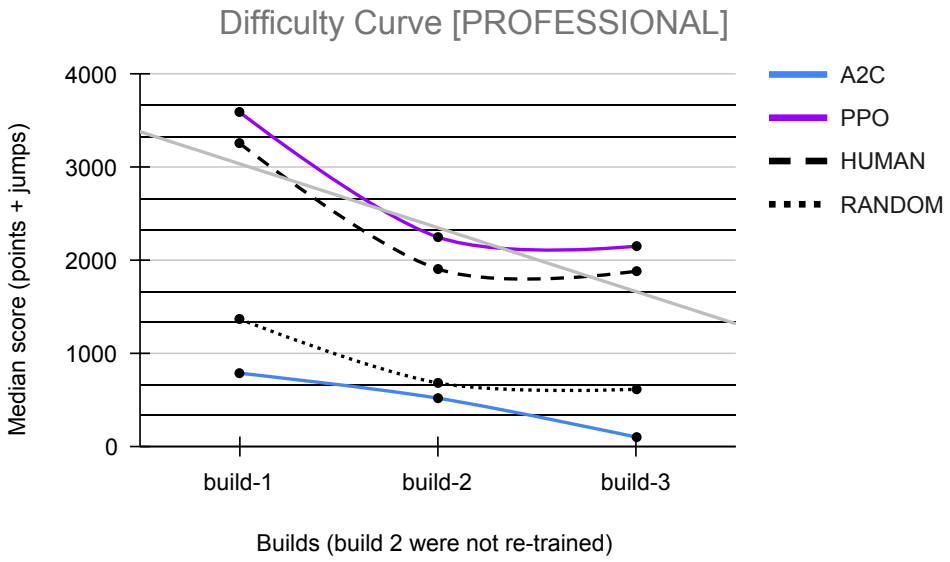


Figure 20: Difficulty Curve between all Professional agents on Case Study B. Jungle Climb.

The PPO agent outperforms, in all three builds, all other agents including the human players. On the other hand, the A2C agent performs poorly, even when compared to the random agent. The random agent has a similar curve to the human but a very low score. Indicating that the game does reward the player's skill in all builds. Also, the curve for the PPO agent and Human players are very similar, indicating that this agent is a good proxy to mimic human players.

Skill Level: Novice

Figure 21 shows the difficulty curve considering the *score* for *novice* skill level. Similar to the *professional* agents, the difficulty across the builds drops on build#2 and maintains on build#3. The exception here is that the PPO novice agent performs much worse than the human novice. Yet, the random agent still outperforms the A2C agent.

Summarizing the Results - Case Study B. Jungle Climb

- There is a spike in difficulty on build#2 that is not balanced on build#3 by adding more gaps jump.
- The performance pattern of the PPO agent is similar to the human players, in novice and professional skill levels.

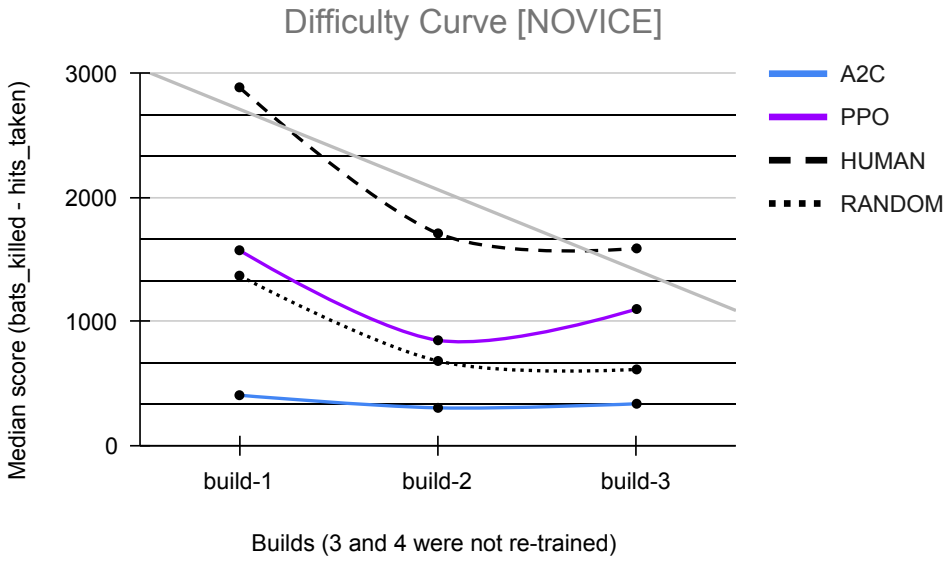


Figure 21: Difficulty Curve between all Novice agents on Case Study B. Jungle Climb.

- The A2C agent performs poorly, even having the biggest rewards during the training.
- The random agent does not outperform the human nor the PPO, showing that the game is skill-based.

6.7 Discussion

Balancing the "Challenge vs. Success": To deliver a good experience to the players, keeping them away from boredom and anxiety (Figure 9), game development requires trial-and-error and much experimentation. Our results show that it is possible to automate a big portion of this process. Within our test scenarios, we identified that the agents could mimic the player's achievements and struggles, even without re-training the models. This shows that is possible to rely on autonomous agents to proxy the human's skill levels. More so, these agents clarify the spikes in difficulties among the builds. This setup provides quick feedback so the game developer can use it right away to promote changes in the game design.

Balancing the "Skill vs. Chance": Games are complex systems that mix mechanics that demand both skill and luck (chance). Finding the "sweet spot" requires a subjective aspect that only experience developers have. Using the random agents proved to be useful in spotting when the game is rewarding, or not, the player's skills. When compared to the other agents, either human

or AI, we noticed that, in certain builds, the random agents were outperforming the others. With a closer look at the metrics, these exact same builds were the most difficult to play. The game developer can use the information to tweak the game accordingly, that is, make the game more skill-based or chance-based.

Checking the game balancing hypothesis: For case study A we confirmed our hypothesis that adding the jump ability would reduce the game difficulty, as it would add a new way to avoid getting hit. However, for case study B, our hypothesis that adding a new gap in the platforms would facilitate the climbing did not confirm. This shows that the game developer's assumptions must be properly tested in practice. Our approach helps this process.

Training the agents already hints for balance issues: Training the agents demands an initial effort that pays off later in development. As each game has different mechanics, it takes time to discover which rewards result in an agent playing well (close to the human performance). While trying different rewards, the game shows some of its balance issues. For example, even simple mechanics like in the *Case Study B. Jungle Climb* require an effort to make the agents play reasonably well.

Players got tired of testing: In our experiment, we ask two people to play the game for three minutes each build, twice. Even with these two simple game scenarios and a few minutes of gameplay, both of the players reported tiredness after playing each game. This adds to the fact that manual testing on video games is tiresome and demands full concentration. In the game industry, playtesters work for hours every day on the same game checking the same issues multiple times. The automation in the game testing aid these testers as well, so they can focus on subjective details of the game.

Deterministic vs. Stochastic: The two games used in our experiments are stochastic, that is, they contain random gameplay elements that are not possible to predict. For example, the bats in case study A and the gaps in case study B. This poses difficulties when training the agents to play the game properly. Whoever, as randomness is a feature in the games, automating the testing is even more valuable for these cases.

6.7.1 Threats to Validity

We had some issues during the implementation of our approach that can become obstacles to adopting it. For example, creating the testing architecture (Figure 12) demands an initial effort to set the environment up. Although this investment pays up in a long run, we understand that

spending engineering time with tooling instead of the game itself might not be welcomed by the developers. However, there are many libraries that allow a rapid configuration of the environment. In our cases, we used Python libraries (Gym and Stable Baselines) to speed up the process of training.

Another issue is to define reward functions so that the agents can play the game properly. Games have different mechanics and even games within the same genre, platformers in our case, do not have common rewards. On the other hand, scenarios within the same game can share the same rewards. For example, in larger games, we can split the chunks of the scenarios the developer wants to test and reuse the rewards with fewer modifications.

Chapter 6 Summary

In this Chapter, we show our approach to automating game testing to balance video games. We described the process of training, playing, and assessing the game. We validated our testing process with two platform games.

Game development often relies more on the feeling of the developer rather than on the process. The result is an empirical manual cycle of development and testing. Although replacing the manual testing is not possible, we believe that game developers can adopt a development pipeline with automated testing that provides quick feedback about the game balance state. Also, developers can submit multiple builds with different configurations and verify the results later. For example, they can leave computer testing overnight and check the feedback the next day.

Chapter 7

Conclusion

"I remember taking the mouse, and I clicked on the mouse, and the warrior walked over and and smacked the skeleton down, and I was like 'Oh my god! That was awesome!'."

DAVID BREVIK, ONE OF THE CREATORS OF THE ACTION RPG GAME GENRE.

In this thesis, we show how we explored and developed a solution to automate video game testing. We started investigating the gaps in the game industry problems. We also analyzed the main tool used to develop games: the game engine. After that, we focused our attention on the game testing problems and solutions. Finally, we proposed and implemented our approach to help game developers and testers to automate the game testing.

In Chapter 3, we reported the main problems in the game industry, their evolution over the years, and their relationship with anti-patterns in traditional software. We identified several gaps in the research and practice related to video game development. Like concerns with the tools used by developers to develop games and game testing, or lack thereof.

In Chapter 4, we showed that there is a lack of academic studies about video-game engines. We also showed that there are qualitative but no quantitative differences between open-source engines and open-source frameworks. We performed a survey that showed that developers' objectives for developing engines are mainly to better control the environment. We did not find testing features in the game engines. Even mature proprietary game engines like Unreal and Unity lack in providing testing features.

In Chapter 5, we discussed the field of video game testing. We found that automation is overlooked and game testing currently relies mostly on human playtesters. Also, the search for the "fun-factor" and "game balance" is mainly executed by game-play testers. Moreover, there is no one-size-fits-all testing process for game projects as the game types differ greatly. Game studios

acknowledge the importance of testing but also have issues like lack of plan and poor testing coverage. The results of the literature review show a rise in research topics related to automated video game testing in recent years. Yet, most testing tools and frameworks are more concerned with the performance of the machine learning models instead of the testing objective. Finally, the survey results show that game developers are skeptical about using automated agents to test games.

In Chapter 6 we implemented our approach to automate game testing to balance video games using autonomous agents. We described the process of training the agents, playing the game, and assessing the game balance. We validated our testing process with two platform games. Game development often relies more on the "feeling" of the developer rather than on the process. The result is an empirical manual cycle of development and testing. Although replacing manual testing is not possible, game developers can adopt a development pipeline with automated testing that provides quick feedback about the game balance state.

Final Words

We conclude that the use of autonomous to test games is faster than the manual feedback loop and provides a viable solution for game balancing, showing spikes in difficulty between game versions and issues with the game design. The flexibility of the metrics, the low cost (human and computer time), and quick feedback of our approach are a step toward providing steady game development and games with better quality.

7.1 Future Work

7.1.1 Short Term (6 months)

There are much more things to be done in the game testing automation domain. First, we want to expand the scope of the experiments. For example, investigate other DRL algorithms to find easier ways to train the agents. But also other heuristics like Monte Carlo Tree Search [16]. With that, try to achieve a super agent that can master the game and demand less effort from the developer. Also add more skill levels for the agents aside from different ways to play the game, that is, a player profile. For example, creating an agent that only focuses on finishing the game while others explore the states of the game. Moreover, add different game genres (racing, fighting, sports, etc) and game types (3D, isometric, etc).

In this thesis we only explore two types of balance problems, there are others that also require the developer's attention (see Appendix B). For example, the *Fairness* in asymmetrical games or *Simple vs Complex* game mechanics. To test these subjective items it might be necessary to create a more complex testing pipeline and more "intelligent" agents.

By creating an ecosystem for automated game testing, it is possible to add other testing objectives to the balancing test, like an exploration of the game and checking its performance (in terms of frames-per-second). These metrics, together with the existent balancing ones, can provide better feedback to the game developer.

7.1.2 Medium Term (1 year)

As seen in Chapter 3, game development problems have been an object of study for several researchers over the last few years. They represent a valuable piece of information. However, datasets on this subject are still small and mostly originate from the same sources, like Gamedeveloper[70]. Also, the writers normally refrain from disclosing the full history.

Indie projects, however, do not have this constraint. Indie developers usually share their unfinished games in search of feedback. They do that by releasing different versions of the game and writing development logs (postmortems).

In this project, we expand an existing postmortem dataset by automating the text analysis so that we can quickly extract the relevant pieces of information from each postmortem document. Instead of just adding new information to our dataset, we want to create a system that automatically fetches each new postmortem, analyze it and add it to the dataset. With that, we can display the information on an online platform, like a service. We believe the game community can benefit from this curated, always up-to-date, dataset.

We want to include postmortems from a different source: `itch.io`. This website built a community that become important for aspiring game developers. Based on this information from the postmortems, we create a profile of the average *itch.io* developer. Showing, for example, which are the most frequent problems they face and what kind of tools they use. To validate our data mining technique and platform we can survey itch.io developers. Our method consists of the following steps:

- Create a crawler to extract postmortem content from itch.io.
- Build a dataset similar to the one built in [20], containing information such as game name, year of release, genre, etc.
- Develop an automated way to search for problems inside of the postmortems' texts. This will require a further study on text mining techniques [119].
- Compare the problems on *itch.io* with the dataset with in Politowski et al. [20].

[70] `gamedeveloper.com`

- Explore other characteristics of the games, like the tools used, the team size, demography of the developers, platform and language they develop with, the development time, how often they publish or abandon the game, and the evolution between the postmortems (same mistakes happening again, problems solved, etc), and similarities between the problems/solutions among different developers.

7.1.3 Long Term (2 years)

We understand that setting up the environment for game testing is not trivial. It demands an effort that could be put into the game development itself. For this reason, we plan to design and create a tool (plugin), so that game developers can use it from within the game engine.

This is a hard problem because we need to define abstraction between the elements of the game engine and the training frameworks. The issue is that we have to comply with the rules of the game engine (physics, collision, attributes of the elements, etc), which might restrict what can be done.

Also, there are a few game engines to choose from. The market is mainly divided into three: *Unreal* engine used in medium to big studios, *Unity* engine used in small to medium studios, and also the custom in-house game engines, mostly from the big studios.

There are also open-source engines. Yet, they still lack features compared to off-the-shelf ones. Godot is one open-source engine that is being adopted by developers. Godot is written in C++ and its version 4 (as of today still in alpha version) provides a better way to be extended. As of now, extending this game engine is the right direction for this project. The roadmap for this project is the following:

- Chose one open-source game engine and study its architecture. Mainly what elements define the game, the game loop, and how to extend its functionalities.
- Embed an AI training framework (like Gym or Stable Baselines) into the game engine. In this step, we will research the literature on other methods to "teach" agents how to play and check which best fits the scenario. For example, in this thesis, we used DRL with PPO/A2C and reward functions to train the agents.
- Abstract the way we train the agents to ease their use for game developers and game designers. In this step, we use different games with different genres so we can generalize the solution.

Bibliography

[1] J. Schell, *The art of game design: a book of lenses*. Amsterdam ; Boston: Elsevier/Morgan Kaufmann, 2008. (Cited on pages ix, xi, 3, 4, 49, 50, 55, and 92.)

[2] Y. Zheng, X. Xie, T. Su, L. Ma, J. Hao, Z. Meng, Y. Liu, R. Shen, Y. Chen, and C. Fan, "Wuji: Automatic Online Combat Game Testing Using Evolutionary Deep Reinforcement Learning," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Nov. 2019, pp. 772–784. (Cited on pages ix, 44, 45, 55, 56, and 90.)

[3] F. Petrillo, M. Pimenta, F. Trindade, and C. Dietrich, "What went wrong? a survey of problems in game development," *Computers in Entertainment*, vol. 7, no. 1, p. 1, Feb. 2009. (Cited on pages xi, 10, 11, 12, 13, 14, and 22.)

[4] M. Washburn, P. Sathiyanarayanan, M. Nagappan, T. Zimmermann, and C. Bird, "What went right and what went wrong: An analysis of 155 postmortems from game development," in *Proceedings of the 38th International Conference on Software Engineering Companion - ICSE 16*. ACM Press, 2016. (Cited on pages xi, 9, 12, 13, 14, and 33.)

[5] Entertainment Software Association - ESA, "Esa's essential facts about the computer and video game industry report," https://www.theesa.com/resource/2021-essential-facts-about-the-video-game-industry/, 2021, [Online; accessed 22-Mar-2022]. (Cited on page 1.)

[6] Newzoo, "Newzoo's 2022 trends to watch in games, esports, cloud, and the metaverse," https://newzoo.com/insights/trend-reports/gaming-industry-trends-2022-report-games-esports-gamestreaming-cloud-metaverse/, 2022, [Online; accessed 22-Mar-2022]. (Cited on page 1.)

[7] L. Chien, L. Xiaozhou, N. Timo, Z. Zheying, and P. Jaakko, "Patches and player community perceptions: Analysis of no mans sky steam reviews," in *DiGRAs 20 Proceedings of the 2020 DiGRA International Conference: Play Everywhere*, 2020. (Cited on page 2.)

[8] B. Keogh and D. Jayemanne, ""game over, man. game over": Looking at the alien in film and videogames," *Arts*, vol. 7, no. 3, p. 43, Aug. 2018. (Cited on page 2.)

[9] D. Spinellis, "State-of-the-Art Software Testing," *IEEE Software*, vol. 34, no. 5, 2017. (Cited on page 3.)

[10] R. Ramadan and B. Hendradjaya, "Development of game testing method for measuring game quality," in *2014 International Conference on Data and Software Engineering (ICODSE)*. IEEE, Nov. 2014, pp. 1–6. (Cited on pages 3 and 22.)

[11] Y. Zhao, I. Borovikov, F. de Mesentier Silva, A. Beirami, J. Rupert, C. Somers, J. Harder, J. Kolen, J. Pinto, R. Pourabolghasem, J. Pestrak, H. Chaput, M. Sardari, L. Lin, S. Narravula, N. Aghdaie, and K. Zaman, "Winning is not everything: Enhancing game development with intelligent agents," *IEEE Transactions on Games*, vol. 12, no. 2, pp. 199–212, 2020. (Cited on pages 3 and 90.)

[12] E. Murphy-Hill, T. Zimmermann, and N. Nagappan, "Cowboys, ankle sprains, and keepers of quality: how is video game development different from software development?" in *Proceedings of the 36th International Conference on Software Engineering - ICSE 2014*. ACM Press, 2014, pp. 1–11. (Cited on pages 3, 22, 32, and 41.)

[13] J. Musil, A. Schweda, D. Winkler, and S. Biffl, "A Survey on the State of the Practice in Video Game Software Development," Institute of Software Technology and Interactive Systems, Tech. Rep., Tech. Rep. March, 2010. (Cited on pages 3 and 32.)

[14] A. Ampatzoglou and I. Stamelos, "Software engineering research for computer games: a systematic review," *Information and Software Technology*, vol. 52, no. 9, pp. 888–901, Sep. 2010. (Cited on page 3.)

[15] M. Morosan and R. Poli, "Lessons from testing an evolutionary automated game balancer in industry," in *2018 IEEE Games, Entertainment, Media Conference (GEM)*, 2018, pp. 263–270. (Cited on page 3.)

[16] D. A. DeLaurentis, J. H. Panchal, A. K. Raz, P. Balasubramani, A. Maheshwari, A. Dachowicz, and K. Mall, "Toward automated game balance: A systematic engineering design approach," in *2021 IEEE Conference on Games (CoG)*, 2021, pp. 1–8. (Cited on pages 4, 52, 53, 73, and 90.)

[17] A. M. Albaghajati and M. A. K. Ahmed, "Video Game Automated Testing Approaches: An Assessment Framework," *IEEE Transactions on Games*, pp. 1–1, 2020. (Cited on pages 4 and 38.)

[18] C. Politowski, Y.-G. Guéhéneuc, and F. Petrillo, "Towards automated video game testing: Still a long way to go," 2022, pre-print at https://arxiv.org/abs/2202.12777. (Cited on pages 4 and 5.)

[19] C. Politowski, F. Petrillo, G. C. Ullmann, J. de Andrade Werly, and Y.-G. Guéhéneuc, "Dataset of Video Game Development Problems," in *Proceedings of the 17th International Conference on Mining Software Repositories*. Seoul Republic of Korea: ACM, Jun. 2020, pp. 553–557. (Cited on pages 5, 12, 13, and 32.)

[20] C. Politowski, F. Petrillo, G. C. Ullmann, and Y.-G. Guéhéneuc, "Game industry problems: An extensive analysis of the gray literature," *Information and Software Technology*, vol. 134, p. 106538, Jun. 2021. (Cited on pages 5, 12, and 74.)

[21] G. C. Ullmann, C. Politowski, Y.-G. Guéhéneuc, F. Petrillo, and J. E. Montandon, "Video game project management anti-patterns," 2022, pre-print at https://arxiv.org/abs/2202.06183. (Cited on page 5.)

[22] C. Politowski, F. Petrillo, J. E. Montandon, M. T. Valente, and Y.-G. Guéhéneuc, "Are game engines software frameworks? A three-perspective study," *Journal of Systems and Software*, vol. 171, p. 110846, Jan. 2021. (Cited on page 5.)

[23] C. Politowski, F. Petrillo, and Y.-G. Gueheneuc, "A Survey of Video Game Testing," in *2021 IEEE/ACM International Conference on Automation of Software Test (AST)*. Madrid, Spain: IEEE, May 2021, pp. 90–99. (Cited on pages 5 and 8.)

[24] C. Politowski, F. Khomh, S. Romano, G. Scanniello, F. Petrillo, Y.-G. Guéhéneuc, and A. Maiga, "A large scale empirical study of the impact of Spaghetti Code and Blob anti-patterns on program comprehension," *Information and Software Technology*, vol. 122, p. 106278, Jun. 2020. (Cited on page 5.)

[25] J. E. Montandon, C. Politowski, L. L. Silva, M. T. Valente, F. Petrillo, and Y.-G. Guéhéneuc, "What skills do IT companies look for in new developers? A study with Stack Overflow jobs," *Information and Software Technology*, vol. 129, p. 106429, Jan. 2021. (Cited on page 5.)

[26] V. H. S. Durelli, R. S. Durelli, S. S. Borges, A. T. Endo, M. M. Eler, D. R. C. Dias, and M. P. Guimarães, "Machine Learning Applied to Software Testing: A Systematic Mapping Study," *IEEE Transactions on Reliability*, vol. 68, no. 3, pp. 1189–1212, Sep. 2019, conference Name: IEEE Transactions on Reliability. (Cited on page 7.)

[27] V. Garousi and M. V. Mäntylä, "When and what to automate in software testing? A multi-vocal literature review," *Information and Software Technology*, vol. 76, pp. 92–117, 2016. (Cited on page 7.)

[28] K. Naik and P. Tripathy, *Software testing and quality assurance: theory and practice*. John Wiley & Sons, 2011. (Cited on page 7.)

[29] P. Bourque and R. Fairley, *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0*, 3rd ed. IEEE Computer Society Press, 2014. (Cited on page 7.)

[30] G. J. Myers, C. Sandler, and T. Badgett, *The Art of Software Testing*. Wiley Publishing, 2011. (Cited on page 7.)

[31] Q. Xie and A. M. Memon, "Designing and comparing automated test oracles for gui-based software applications," *ACM Trans. Softw. Eng. Methodol.*, vol. 16, no. 1, p. 4–es, feb 2007. (Cited on page 7.)

[32] M. Aniche, *Effective Software Testing*. Manning publications, 2021. (Cited on page 7.)

[33] L. Mariani, M. Pezzè, and D. Zuddas, "Recent Advances in Automatic Black-Box Testing," in *Advances in Computers*. Elsevier, 2015, vol. 99, pp. 157–193. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0065245815000315 (Cited on page 7.)

[34] K. Wiklund, S. Eldh, D. Sundmark, and K. Lundqvist, "Impediments for software test automation: A systematic literature review: Impediments for Software Test Automation," *Software Testing, Verification and Reliability*, vol. 27, no. 8, Dec. 2017. (Cited on page 7.)

[35] F. Ricca, A. Marchetto, and A. Stocco, "AI-based Test Automation: A Grey Literature Analysis," in *2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, Apr. 2021, pp. 263–270. (Cited on page 7.)

[36] L. Levy and J. Novak, *Game Development Essentials: Game QA & Testing*. Cengage Learning, 2009. (Cited on page 8.)

[37] A. Drachen, P. Mirza-Babaei, and L. E. Nacke, *Games user research*, 1st ed. Oxford, UK ; New York: Oxford University Press, 2018. (Cited on page 8.)

[38] D. Callele, E. Neufeld, and K. Schneider, "Requirements engineering and the creative process in the video game industry," in *13th IEEE International Conference on Requirements Engineering (RE05)*. IEEE, 2005, pp. 240–250. (Cited on pages 9, 12, and 13.)

[39] W. Pree, "Meta patterns—a means for capturing the essentials of reusable object-oriented design," in *European Conference on Object-Oriented Programming*. Springer, 1994, pp. 150–162. (Cited on page 9.)

[40] C. Larman, *Applying UML and patterns: an introduction to object oriented analysis and design and interative development*. Pearson Education India, 2012. (Cited on page 9.)

[41] R. N. Taylor, "Only the architecture you need," in *The Essence of Software Engineering*, V. Gruhn and R. Striemer, Eds. Cham: Springer International Publishing, 2018, pp. 77–89. (Cited on page 9.)

[42] J. Gregory, *Game Engine Architecture, Second Edition*. Natick, MA, USA: A. K. Peters, Ltd., 2014. (Cited on pages 10 and 21.)

[43] H. Lowood and R. Guins, *Debugging Game History: A Critical Lexicon*. The MIT Press, 2016. (Cited on page 10.)

[44] H. Lowood, "Game Engines and Game History," in *History of Games International Conference Proceedings*, Jan. 2014, pp. 179–98. (Cited on page 10.)

[45] D. Kushner, *Masters of Doom: How Two Guys Created an Empire and Transformed Pop Culture*. Random House Publishing Group, 2003. (Cited on pages 10 and 22.)

[46] J. Kasurinen, M. Palacin-Silva, and E. Vanhala, "What concerns game developers? a study on game development processes, sustainability and metrics," in *2017 IEEE/ACM 8th Workshop on Emerging Trends in Software Metrics (WETSoM)*. IEEE, May 2017, pp. 15–21. (Cited on pages 10 and 22.)

[47] C. Lewis and J. Whitehead, "The whats and the whys of games and software engineering," in *Proceeding of the 1st international workshop on Games and software engineering - GAS 11*. ACM Press, 2011, pp. 1–4. (Cited on pages 10, 12, 13, 22, 33, and 41.)

[48] D. Callele, P. Dueck, K. Wnuk, and P. Hynninen, "Experience requirements in video games definition and testability," in *2015 IEEE 23rd International Requirements Engineering Conference (RE)*, 2015, pp. 324–333. (Cited on pages 10, 22, and 32.)

[49] J. Kasurinen and K. Smolander, "Defining an Iterative ISO/IEC 29110 Deployment Package for Game Developers," *International Journal of Information Technologies and Systems Approach*, vol. 10, no. 1, pp. 107–125, 2016. (Cited on page 10.)

[50] D. Lin, C.-P. Bezemer, and A. E. Hassan, "Studying the urgent updates of popular games on the Steam platform," *Empirical Software Engineering*, vol. 22, no. 4, pp. 2095–2126, 2016. (Cited on page 11.)

[51] H. Edholm, M. Lidstrom, J.-P. Steghofer, and H. Burden, "Crunch Time: The Reasons and Effects of Unpaid Overtime in the Games Industry," in *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*. IEEE, May 2017, pp. 43–52. (Cited on pages 11 and 13.)

[52] C. M. Kanode and H. M. Haddad, "Software engineering challenges in game development," in *2009 Sixth International Conference on Information Technology: New Generations*. IEEE, 2009, pp. 260–265. (Cited on pages 12, 13, and 22.)

[53] J. Blow, "Game development: Harder than you think: Ten or twenty years ago it was all fun and games. now it's blood, sweat, and code." *Queue*, vol. 1, no. 10, p. 28–37, feb 2004. [Online]. Available: https://doi-org.lib-ezproxy.concordia.ca/10.1145/971564.971590 (Cited on page 12.)

[54] T. Tschang, "Videogames as interactive experiential products and their manner of development," *International Journal of Innovation Management*, vol. 9, no. 01, pp. 103–131, 2005. (Cited on page 12.)

[55] K.-J. Stol, P. Ralph, and B. Fitzgerald, "Grounded theory in software engineering research," in *Proceedings of the 38th International Conference on Software Engineering - ICSE '16*. ACM Press, 2016. (Cited on page 13.)

[56] C. Politowski, L. Fontoura, F. Petrillo, and Y.-G. Guéhéneuc, "Are the old days gone?: A survey on actual software engineering processes in video game industry," in *Proceedings of the 5th International Workshop on Games and Software Engineering - GAS '16*. Austin, Texas: ACM Press, 2016, pp. 22–28. (Cited on pages 18 and 49.)

[57] T. Kude, S. Mithas, C. T. Schmidt, and A. Heinzl, "How Pair Programming Influences Team Performance: The Role of Backup Behavior, Shared Mental Models, and Task Novelty," *Inf. Syst. Res.*, 2019. (Cited on page 18.)

[58] A. Alami, M. Leavitt Cohn, and A. Wąsowski, "Why does code review work for open source software communities?" in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, 2019, pp. 1073–1083. (Cited on page 18.)

[59] J. Schreier, *Blood, Sweat, and Pixels: The Triumphant, Turbulent Stories Behind How Video Games Are Made*. HarperCollins, 2017. (Cited on pages 19, 21, and 22.)

[60] ——, "How bioware's anthem went wrong," https://kotaku.com/how-biowares-anthem-went-wrong-1833731964, 2019, accessed: 2019-04-03. (Cited on page 21.)

[61] A. Thorn, *Game Engine Design and Implementation*, ser. Foundations of game development. Jones & Bartlett Learning, 2011. (Cited on page 21.)

[62] J. Hughes, "What to look for when evaluating middleware for integration," in *Game Engine Gems 1*, E. Lengyel, Ed. Jones and Bartlett, 2010, pp. 3–10. (Not cited.)

[63] A. Sherrod, *Ultimate 3D Game Engine Design & Architecture*, ser. Charles River Media game development series. Charles River Media, 2007. (Cited on page 21.)

[64] F. Messaoudi, G. Simon, and A. Ksentini, "Dissecting games engines: The case of Unity3D," *Annual Workshop on Network and Systems Support for Games*, vol. 2016-January, pp. 1–6, 2016. (Cited on page 22.)

[65] F. Petrillo and M. S. Pimenta, "Is agility out there?: agile practices in game development," in *Proceedings of the 28th Annual International Conference on Design of Communication, SIGDOC 2010, São Carlos, São Paulo state, Brazil, September 26-29, 2010*, J. C. Anacleto, R. P. de Mattos Fortes, and C. J. Costa, Eds. none: ACM, 2010, pp. 9–15. (Cited on page 22.)

[66] S. Hyrynsalmi, E. Klotins, M. Unterkalmsteiner, T. Gorschek, N. Tripathi, L. B. Pompermaier, and R. Prikladnicki, "What is a Minimum Viable (Video) Game?" in *17th IFIP WG 6.11 Conference on e-Business, e-Services, and e-Society, I3E 2018*, vol. 11195. Springer-Verlag GmbH, Oct. 2018, pp. 217–231. (Cited on page 22.)

[67] M. Mozgovoy and E. Pyshkin, "A comprehensive approach to quality assurance in a mobile game project," in *Proceedings of the 14th Central and Eastern European Software Engineering Conference Russia on ZZZ - CEE-SECR 18*. ACM Press, 2018, pp. 1–8. (Cited on pages 22 and 41.)

[68] M. Toftedahl and H. Engström, "A Taxonomy of Game Engines and the Tools that Drive the Industry," in *Proceedings of the 2019 DiGRA International Conference: Game, Play and the Emerging Ludo-Mix*, 2019, pp. –. (Cited on page 22.)

[69] B. Cowan and B. Kapralos, "A survey of frameworks and game engines for serious game development," in *2014 IEEE 14th International Conference on Advanced Learning Technologies*. IEEE, Jul. 2014, pp. 662–664. (Cited on page 22.)

[70] ——, "An overview of serious game engines and frameworks," in *Recent Advances in Technologies for Inclusive Well-Being*, A. L. Brooks, S. Brahnam, B. Kapralos, and L. C. Jain, Eds. Springer International Publishing, 2017, pp. 15–38. (Cited on page 22.)

[71] M. P. Neto and J. R. F. Brega, "A survey of solutions for game engines in the development of immersive applications for multi-projection systems as base for a generic solution design," in *2015 XVII Symposium on Virtual and Augmented Reality*. IEEE, May 2015, pp. 61–70. (Cited on page 22.)

[72] A. I. Wang and N. Nordmark, "Software architectures and the creative processes in game development," in *Entertainment Computing - ICEC 2015*, K. Chorianopoulos, M. Divitini,

J. Baalsrud Hauge, L. Jaccheri, and R. Malaka, Eds. Cham: Springer International Publishing, 2015, pp. 272–285. (Cited on page 22.)

[73] E. F. Anderson, S. Engel, P. Comninos, and L. McLoughlin, "The case for research in game engine architecture," in *Proceedings of the 2008 Conference on Future Play Research, Play, Share - Future Play 08*. ACM Press, 2008, p. 228. (Cited on page 23.)

[74] G. Avelino, L. Passos, A. Hora, and M. T. Valente, "A novel approach for estimating truck factors," in *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*, May 2016, pp. 1–10. (Cited on page 23.)

[75] F. G. de Oliveira Neto, R. Torkar, R. Feldt, L. Gren, C. A. Furia, and Z. Huang, "Evolution of statistical analysis in empirical software engineering research: Current state and steps forward," *Journal of Systems and Software*, vol. 156, pp. 246–267, 2019. (Cited on page 23.)

[76] C. Wohlin, P. Runeson, M. Hst, M. C. Ohlsson, B. Regnell, and A. Wessln, *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated, 2012. (Cited on page 23.)

[77] M. Lavallée and P. N. Robillard, "Why good developers write bad code: An observational case study of the impacts of organizational factors on software quality," in *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, ser. ICSE '15. IEEE Press, 2015, pp. 677–687. (Cited on page 26.)

[78] H. Borges and M. T. Valente, "What's in a github star? understanding repository starring practices in a social coding platform," *Journal of Systems and Software*, vol. 146, pp. 112–129, 2018. (Cited on page 27.)

[79] J. Schreier, *Press Reset: Ruin and Recovery in the Video Game Industry*. Grand Central Publishing, 2021. (Cited on page 30.)

[80] R. J. Adams, P. Smart, and A. S. Huff, "Shades of grey: guidelines for working with the grey literature in systematic reviews for management and organizational studies," *International Journal of Management Reviews*, vol. 19, no. 4, pp. 432–454, 2017. (Cited on page 31.)

[81] S. Hooper, "Automated Testing and Validation of Computer Graphics Implementations for Cross-platform Game Development," Master's thesis, Auckland University of Technology, 2017. (Cited on pages 32 and 41.)

[82] A. I. Wang and N. Nordmark, "Software Architectures and the Creative Processes in Game Development," *Entertainment Computing – ICEC 2014*, vol. 8770, pp. 272–285, 2014. (Cited on page 32.)

[83] J. Koutonen and M. Leppänen, "How are agile methods and practices deployed in video game development? a survey into finnish game studios," in *Lecture Notes in Business Information Processing*. Springer Berlin Heidelberg, 2013, vol. 149, pp. 135–149. (Cited on page 32.)

[84] W. Scacchi and K. M. Cooper, "Research Challenges at the Intersection of Computer Games and Software Engineering," *Computer Games and Software ...*, no. Fdg, 2015. (Cited on pages 32 and 33.)

[85] V. Paakkanen, "User Experience of Game Development Environments: Can making games be as fun as playing them?" Master's thesis, Aalto University School of Science, 2014. (Cited on page 33.)

[86] S. Aleem, L. F. Capretz, and F. Ahmed, "Critical Success Factors to Improve the Game Development Process from a Developer's Perspective," *Journal of Computer Science and Technology*, vol. 31, no. 5, pp. 925–950, 2016. (Cited on page 33.)

[87] L. Pascarella, F. Palomba, M. Di Penta, and A. Bacchelli, "How is video game development different from software development in open source?" in *Proceedings - International Conference on Software Engineering*, Gothenburg, Sweden, 2018, pp. 392–402. (Cited on pages 33 and 41.)

[88] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994. (Cited on page 33.)

[89] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, ser. EASE '14. New York, NY, USA: Association for Computing Machinery, 2014. (Cited on page 39.)

[90] C. Gordillo, J. Bergdahl, K. Tollmar, and L. Gisslén, "Improving playtesting coverage via curiosity driven reinforcement learning agents," in *2021 IEEE Conference on Games (CoG)*, 2021, pp. 1–8. (Cited on pages 40, 44, 47, and 90.)

[91] H.-r. Ruonala, "Agile Game Development: A Systematic Literature Review," Master's thesis, University of Helsinki, Faculty of Science, Department of Computer Science, 2016. (Cited on page 41.)

[92] R. E. S. Santos, C. V. C. Magalhaes, L. F. Capretz, J. S. Correia-Neto, F. Q. B. da Silva, and A. Saher, "Computer games are serious business and so is their quality," in *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement - ESEM 18*, no. 33. ACM Press, 2018, pp. 1–10. (Cited on page 41.)

[93] C. P. Schultz, R. Bryant, and T. Langdell, *Game Testing All in One (Game Development Series)*. Course Technology PTR, 2005. (Cited on page 41.)

[94] J. Kasurinen, J.-P. Strandén, and K. Smolander, "What do game developers expect from development and design tools?" in *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering - EASE 13*. ACM Press, 2013, p. 36. (Cited on page 41.)

[95] S. F. Gudmundsson, P. Eisen, E. Poromaa, A. Nodet, S. Purmonen, B. Kozakowski, R. Meurling, and L. Cao, "Human-Like Playtesting with Deep Learning," in *2018 IEEE Conference on Computational Intelligence and Games (CIG)*. Maastricht: IEEE, Aug. 2018, pp. 1–8. (Cited on pages 43, 44, 52, 53, and 90.)

[96] S. Roohi, A. Relas, J. Takatalo, H. Heiskanen, and P. Hämäläinen, "Predicting game difficulty and churn without players," in *Proceedings of the Annual Symposium on Computer-Human Interaction in Play*, 2020, pp. 585–593. (Cited on pages 44, 52, 53, and 90.)

[97] S. Ariyurek, E. Surer, and A. Betin-Can, "Playtesting: What is Beyond Personas," Jul. 2021, arXiv preprint arXiv:2107.11965. (Cited on pages 44, 45, and 90.)

[98] J. Pfau, J. D. Smeddinck, and R. Malaka, "Automated Game Testing with ICARUS: Intelligent Completion of Adventure Riddles via Unsupervised Solving," in *Extended Abstracts Publication of the Annual Symposium on Computer-Human Interaction in Play*, ser. CHI PLAY '17 Extended Abstracts. New York, NY, USA: Association for Computing Machinery, Oct. 2017, pp. 153–164. (Cited on pages 44, 45, and 90.)

[99] S. Ariyurek, A. Betin-Can, and E. Surer, "Enhancing the monte carlo tree search algorithm for video game testing," in *2020 IEEE Conference on Games (CoG)*, 2020, pp. 25–32. (Cited on pages 44, 45, and 90.)

[100] T. McKenzie, M. Morales-Trujillo, S. Lukosch, and S. Hoermann, "Is agile not agile enough? a study on how agile is applied and misapplied in the video game development industry," in *2021 IEEE/ACM Joint 15th International Conference on Software and System Processes (ICSSP) and 16th ACM/IEEE International Conference on Global Software Engineering (ICGSE)*. IEEE, 2021, pp. 94–105. (Cited on page 49.)

[101] N. Justesen, P. Bontrager, J. Togelius, and S. Risi, "Deep Learning for Video Game Playing," *IEEE Transactions on Games*, vol. 12, no. 1, Mar. 2020. (Cited on pages 51 and 55.)

[102] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017. (Cited on page 51.)

[103] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," *CoRR*, vol. abs/1602.01783, 2016. (Cited on page 51.)

[104] A. Isaksen, D. Gopstein, J. Togelius, and A. Nealen, "Exploring Game Space of Minimal Action Games via Parameter Tuning and Survival Analysis," *IEEE Transactions on Games*, vol. 10, no. 2, pp. 182–194, Jun. 2018. (Cited on pages 52, 53, and 90.)

[105] J. Pfau, A. Liapis, G. Volkmar, G. N. Yannakakis, and R. Malaka, "Dungeons Replicants: Automated Game Balancing via Deep Player Behavior Modeling," in *2020 IEEE Conference on Games (CoG)*, Aug. 2020, pp. 431–438. (Cited on pages 52, 53, and 90.)

[106] J. Pfau, A. Liapis, G. N. Yannakakis, and R. Malaka, "Dungeons amp; replicants ii: Automated game balancing across multiple difficulty dimensions via deep player behavior modeling," *IEEE Transactions on Games*, pp. 1–1, 2022. (Cited on pages 52 and 53.)

[107] P. Garca-Snchez, A. Tonda, A. M. Mora, G. Squillero, and J. J. Merelo, "Automated playtesting in collectible card games using evolutionary algorithms," *Knowledge-Based Systems*, vol. 153, no. C, pp. 133–146, Aug. 2018. (Cited on pages 53 and 90.)

[108] M.-V. Aponte, G. Levieux, and S. Natkin, "Measuring the level of difficulty in single player video games," *Entertainment Computing*, vol. 2, no. 4, pp. 205–213, Jan. 2011. (Cited on pages 53 and 90.)

[109] F. de Mesentier Silva, S. Lee, J. Togelius, and A. Nealen, "AI-based playtesting of contemporary board games," in *Proceedings of the 12th International Conference on the Foundations of Digital Games*. Hyannis Massachusetts: ACM, Aug. 2017, pp. 1–10. (Cited on pages 53 and 90.)

[110] F. de Mesentier Silva, S. Lee, J. Togelius, and A. Nealen, "Ai as evaluator: Search driven playtesting of modern board games," in *AAAI Workshops*, 2017. (Cited on pages 53 and 90.)

[111] C. Guerrero-Romero, S. M. Lucas, and D. Perez-Liebana, "Using a Team of General AI Algorithms to Assist Game Design and Testing," in *2018 IEEE Conference on Computational Intelligence and Games (CIG)*. Maastricht: IEEE, Aug. 2018, pp. 1–8. (Cited on pages 53 and 90.)

[112] L. Mugrai, F. Silva, C. Holmgård, and J. Togelius, "Automated playtesting of matching tile games," in *2019 IEEE Conference on Games (CoG)*, 2019, pp. 1–7. (Cited on pages 53 and 90.)

[113] N. John and J. Gow, "Adversarial behaviour debugging in a two button fighting game," in *2021 IEEE Conference on Games (CoG)*, 2021, pp. 01–08. (Cited on pages 53 and 90.)

[114] V. Sriram, "Automated playtesting of platformer games using reinforcement learning," Ph.D. dissertation, Northeastern University, 2019. (Cited on pages 53 and 90.)

[115] Y. Shin, J. Kim, K. Jin, and Y. B. Kim, "Playtesting in Match 3 Game Using Strategic Plays via Reinforcement Learning," *IEEE Access*, vol. 8, pp. 51 593–51 600, 2020. (Cited on pages 53 and 90.)

[116] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013. (Cited on page 55.)

[117] A. P. Badia, B. Piot, S. Kapturowski, P. Sprechmann, A. Vitvitskyi, D. Guo, and C. Blundell, "Agent57: Outperforming the atari human benchmark," 2020. (Cited on page 55.)

[118] B. Bostan, Ed., *Game User Experience And Player-Centered Design*, ser. International Series on Computer Entertainment and Media Technology. Cham: Springer International Publishing, 2020. (Cited on page 57.)

[119] C. Lu, J. Peltonen, and T. Nummenmaa, "Game postmortems vs. developer reddit amas: Computational analysis of developer communication," in *Proceedings of the 14th International Conference on the Foundations of Digital Games*. New York, NY, USA: Association for Computing Machinery, 2019. (Cited on page 74.)

[120] F. Southey, R. Holte, G. Xiao, M. Trommelen, and J. Buchanan, "Machine learning for semi-automated gameplay analysis," in *Proceedings of the 2005 Game Developers Conference (GDC*, 2005. (Cited on page 90.)

[121] T. Kristo and N. U. Maulidevi, "Deduction of fighting game countermeasures using neuroevolution of augmenting topologies," in *2016 international conference on data and software engineering (ICoDSE)*. IEEE, 2016, pp. 1–6. (Cited on page 90.)

[122] O. Keehl and A. M. Smith, "Monster Carlo: An MCTS-based Framework for Machine Playtesting Unity Games," in *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, Aug. 2018, pp. 1–8. (Cited on page 90.)

[123] C. Salge, C. Lipski, T. Mahlmann, and B. Mathiak, "Using genetically optimized artificial intelligence to improve gameplaying fun for strategical games," in *Proceedings of the 2008 ACM SIGGRAPH Symposium on Video Games - Sandbox '08*. Los Angeles, California: ACM Press, 2008, p. 7. (Cited on page 90.)

[124] J. Bergdahl, C. Gordillo, K. Tollmar, and L. Gisslen, "Augmenting Automated Game Testing with Deep Reinforcement Learning," in *2020 IEEE Conference on Games (CoG)*. Osaka, Japan: IEEE, Aug. 2020, pp. 600–603. (Cited on page 90.)

[125] B. Horn, J. A. Miller, G. Smith, and S. Cooper, "A Monte Carlo Approach to Skill-Based Automated Playtesting," in *Proceedings of the Fourteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2019, p. 16. (Cited on page 90.)

[126] T. Machado, D. Gopstein, A. Nealen, O. Nov, and J. Togelius, "AI-Assisted Game Debugging with Cicero," in *2018 IEEE Congress on Evolutionary Computation (CEC)*, Jul. 2018, pp. 1–8. (Cited on page 90.)

[127] F. d. M. Silva, I. Borovikov, J. Kolen, N. Aghdaie, and K. Zaman, "Exploring gameplay with ai agents," in *Proceedings of the Fourteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, ser. AIIDE'18. AAAI Press, 2018. (Cited on page 90.)

[128] S. . Stahlke, A. Nova, and P. Mirza-Babaei, "Artificial Playfulness: A Tool for Automated Agent-Based Playtesting," in *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems*. Glasgow Scotland Uk: ACM, May 2019, pp. 1–6. (Cited on page 90.)

[129] M. C. Green, A. Khalifa, G. A. B. Barros, T. Machado, and J. Togelius, "Automatic critical mechanic discovery using playtraces in video games," in *International Conference on the Foundations of Digital Games*, ser. FDG '20. New York, NY, USA: Association for Computing Machinery, 2020. (Cited on page 90.)

[130] R. Prada, I. S. W. B. Prasetya, F. Kifetew, F. Dignum, T. E. J. Vos, J. Lander, J.-y. Donnart, A. Kazmierowski, J. Davidson, and P. M. Fernandes, "Agent-based Testing of Extended Reality Systems," in *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*, Oct. 2020, pp. 414–417. (Cited on page 90.)

[131] D. R. C. Alves, "Modeling of Synthetic Players As An Instrument For Testing Generative Content," Ph.D. dissertation, Universidade de Coimbra, 2021. (Cited on page 90.)

[132] H. Suetake, T. Fukusato, C. Arzate Cruz, A. Nealen, and T. Igarashi, "Interactive Design Exploration of Game StagesUsing Adjustable Synthetic Testers," in *International Conference on the Foundations of Digital Games*, ser. FDG '20. New York, NY, USA: Association for Computing Machinery, Sep. 2020, pp. 1–4. (Cited on page 90.)

[133] B. Chan, J. Denzinger, D. Gates, K. Loose, and J. Buchanan, "Evolutionary behavior testing of commercial computer games," in *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753)*, vol. 1, Jun. 2004. (Cited on page 90.)

[134] S. Ariyurek, A. Betin-Can, and E. Surer, "Automated Video Game Testing Using Synthetic and Human-Like Agents," *IEEE Transactions on Games*, pp. 1–1, 2019. (Cited on page 90.)

[135] A. Zook, B. Harrison, and M. O. Riedl, "Monte-Carlo Tree Search for Simulation-based Strategy Analysis," in *Proceedings of the 10th International Conference on the Foundations of Digital Games (FDG 2015)*, 2015. (Cited on page 90.)

[136] B. Wilkins, C. Watkins, and K. Stathis, "A Metric Learning Approach to Anomaly Detection in Video Games," in *2020 IEEE Conference on Games (CoG)*. Osaka, Japan: IEEE, Aug. 2020, pp. 604–607. (Cited on page 90.)

[137] D. Loubos, "Automated Testing in Virtual Worlds," Master's thesis, Utrecht University, 2018. (Cited on page 90.)

[138] S. Shirzadehhajimahmood, I. S. W. B. Prasetya, F. Dignum, M. Dastani, and G. Keller, "Using an agent-based approach for robust automated testing of computer games," in *Proceedings of the 12th International Workshop on Automating TEST Case Design, Selection, and Evaluation*. Athens Greece: ACM, Aug. 2021, pp. 1–8. (Cited on page 90.)

[139] N. S. Pereira, "Towards Automated Playtesting in Game Development," in *Proceedings of SBGames 2021*, 2021, p. 6. (Cited on page 90.)

[140] S. Devlin, A. Anspoka, N. Sephton, P. I. Cowling, and J. Rollason, "Combining Gameplay Data with Monte Carlo Tree Search to Emulate Human Play," in *Proceedings of the Twelfth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, ser. AIIDE'16. AAAI Press, 2016, p. 7. (Cited on page 90.)

[141] C. Holmgård, M. C. Green, A. Liapis, and J. Togelius, "Automated playtesting with procedural personas through mcts with evolved heuristics," *IEEE Transactions on Games*, vol. 11, no. 4, pp. 352–362, 2019. (Cited on page 90.)

[142] C. Arzate Cruz and T. Igarashi, "MarioMix: Creating Aligned Playstyles for Bots with Interactive Reinforcement Learning," in *Extended Abstracts of the 2020 Annual Symposium on Computer-Human Interaction in Play*, ser. CHI PLAY '20. New York, NY, USA: Association for Computing Machinery, Nov. 2020, pp. 134–139. (Cited on page 90.)

[143] N. Napolitano, "Testing match-3 video games with Deep Reinforcement Learning," Jun. 2020, arXiv preprint arXiv:2007.01137. (Cited on page 90.)

[144] P. de Woillemont, R. Labory, and V. Corruble, "Configurable Agent With Reward As Input: A Play-Style Continuum Generation," in *3rd IEEE Conference on Games*, 2021. (Cited on page 90.)

[145] S. Roohi, C. Guckelsberger, A. Relas, H. Heiskanen, J. Takatalo, and P. Hämäläinen, "Predicting game difficulty and engagement using ai players," *Proc. ACM Hum.-Comput. Interact.*, vol. 5, no. CHI PLAY, oct 2021. (Cited on page 90.)

[146] Y. Wu, Y. Chen, X. Xie, B. Yu, C. Fan, and L. Ma, "Regression Testing of Massively Multiplayer Online Role-Playing Games," in *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Sep. 2020, pp. 692–696. (Cited on page 90.)

[147] K. Chen, Y. Li, Y. Chen, C. Fan, Z. Hu, and W. Yang, "GLIB: Towards automated test oracle for graphically-rich applications," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. Athens Greece: ACM, Aug. 2021, pp. 1093–1104. (Cited on page 90.)

[148] C. Paduraru, M. Paduraru, and A. Stefanescu, "Automated game testing using computer vision methods," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW)*, 2021, p. 8. (Cited on page 90.)

[149] C. Guerrero-Romero, S. Kumari, D. Perez-Liebana, and S. Deterding, "Studying General Agents in Video Games from the Perspective of Player Experience," in *Proceedings of the Sixteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, ser. AIIDE'20, vol. 16, no. 1. AAAI Press, Oct. 2020, pp. 217–223. (Cited on page 90.)

[150] A. Nantes, R. Brown, and F. Maire, "A framework for the semi-automatic testing of video games," in *Proceedings of the Fourth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, ser. AIIDE'08. AAAI Press, 2008, p. 197–202. (Cited on page 90.)

[151] P. Davarmanesh, K. Jiang, T. Ou, A. Vysogorets, S. Ivashkevich, M. Kiehn, S. H. Joshi, and N. Malaya, "Automating Artifact Detection in Video Games," Nov. 2020, arXiv preprint arXiv:2011.15103. (Cited on page 90.)

[152] S. Agarwal, C. Herrmann, G. Wallner, and F. Beck, "Visualizing AI Playtesting Data of 2D Side-scrolling Games," in *2020 IEEE Conference on Games (CoG)*, Aug. 2020, pp. 572–575. (Cited on page 90.)

Appendix A

Papers on Video Game Testing and Automation

Table A.1: Papers that deal with video game testing, automation, and machine learning models.

Author	Year	Test Objective
Southey et al. [120]	2005	Balancing
Aponte et al. [108]	2011	Balancing
de Mesentier Silva et al. [109]	2017	Balancing
de Mesentier Silva et al. [110]	2017	Balancing
Kristo and Maulidevi [121]	2017	Balancing
Gudmundsson et al. [95]	2018	Balancing
Garca-Snchez et al. [107]	2018	Balancing
Isaksen et al. [104]	2018	Balancing
Keehl and Smith [122]	2018	Balancing
Mugrai et al. [112]	2019	Balancing
Sriram [114]	2019	Balancing
Pfau et al. [105]	2020	Balancing
Shin et al. [115]	2020	Balancing
Roohi et al. [96]	2020	Balancing
Zhao et al. [11]	2020	Balancing
John and Gow [113]	2021	Balancing
DeLaurentis et al. [16]	2021	Balancing
Salge et al. [123]	2008	Balancing
Guerrero-Romero et al. [111]	2018	Balancing
Bergdahl et al. [124]	2020	Collision
Horn et al. [125]	2019	Exploration
Machado et al. [126]	2018	Exploration
Silva et al. [127]	2018	Exploration
Stahlke et al. [128]	2019	Exploration
Green et al. [129]	2020	Exploration
Gordillo et al. [90]	2021	Exploration
Ariyurek et al. [97]	2021	Exploration
Zheng et al. [2]	2019	Exploration
Prada et al. [130]	2020	Exploration
Alves [131]	2021	Exploration
Suetake et al. [132]	2020	Exploration
Chan et al. [133]	2004	Finding bugs
Pfau et al. [98]	2017	Finding bugs
Ariyurek et al. [134]	2019	Finding bugs
Zook et al. [135]	2019	Finding bugs
Wilkins et al. [136]	2020	Finding bugs
Ariyurek et al. [99]	2020	Finding bugs
Loubos [137]	2018	Mechanics
Shirzadehhajimahmood et al. [138]	2021	Mechanics
Pereira [139]	2021	Mechanics
Devlin et al. [140]	2016	Player modeling
Holmgård et al. [141]	2018	Player modeling
Arzate Cruz and Igarashi [142]	2020	Player modeling
Napolitano [143]	2020	Player modeling
de Woillemont et al. [144]	2021	Player modeling
Roohi et al. [145]	2021	Player modeling
Wu et al. [146]	2020	Regression
Chen et al. [147]	2021	UI
Paduraru et al. [148]	2021	UI
Guerrero-Romero et al. [149]	2020	UX
Nantes et al. [150]	2008	Visual correctness
Davarmanesh et al. [151]	2020	Visual correctness
Agarwal et al. [152]	2020	Visualization

Appendix B

Game Balance Types

Table B.2: Game Balance Types [1]

Balance Type	Description
Fairness	Symmetrical: give equal resources to all players, like chess and monopoly. Asymmetrical: give opponents different resources, like Dota, Street Fighter, or most of the online games. A classic way to balance assymmetrical games is using the \textit{rock-paper-scisor} design, where every strength also has a weakness.
Challenge vs. Success	If play is too challenging, the player becomes frustrated. But if the player succeeds too easily, they can become bored. Keeping the player on the middle path means keeping the experiences of challenge and success in proper balance.
Meaningful Choices	Give the player choices that will have a real impact on what happens next. For example, offering 50 vehicles to choose from, but if they all drive the same way does not change anything. For these cases is common to use the concept of Triangularity, that is, offer a choice with high risk and high reward or a low risk and low reward.
Skill vs. Chance	Games of skill are more like athletic contest (which player is the best?) while games of chance are more casual, as much of the outcome is decided by fate. For example, dealing out a hand of cards is pure chance but choosing how to play them is pure skill.
Head vs. Hands	How much of the game should involve doing a challenging physical activity (be it steering, throwing, or pushing buttons dexterously) and how much of it should involve thinking?
Competition vs. Cooperation	Multiplayer games can be either focus on competition (Fighting games), cooperation (Don't Starve Together), or a blend of both (MMOs or Battle Royale games).
Short vs. Long	The length of the gameplay: if too short, players may not get a chance to develop meaningful strategies; if too long, players may grow bored.
Rewards	Rewards are the way the game tells the player have done well. Could be a praise, points, resources, etc.
Punishment	Punishment (loss of points, shortned gameplay, etc) used properly can increase the enjoyment that players get from games. For example, resources in a game are worth more if there is a chance they can be taken away. Also, taking risks is exciting and the punishment increases challenge.
Freedom	Give the player control, or freedom, over the experience. The question is how much control? Giving the player control over everything can be boring for the player.
Simple vs. Complex	It is related to game mechanics. It is a double-edge sword. For example, a game can be too simple it is boring or simple and elegant. Moreover, the game can be overly complex and confusing or have a richly and intricately complex design.
Detail vs. Imagination	Games provide some level of detail, but leave it to the player to fill in the rest. Deciding exactly what details should be provided and which should be left to the player's imagination is a different, but important kind of balance to strike.