INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI®

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

A CASE STUDY IN DOCUMENTING AND DEVELOPING FRAMEWORKS

Pierre Dénommée

A Thesis
in
The Department
of
Computer Science

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Computer Science
Concordia University
Montréal, Québec, Canada

April 1998

Abstract

A Case Study in Documenting and Developing Frameworks

Pierre Dénommée

A framework is a collection of abstract classes that provides an infrastructure common to a family of applications. The design of the framework fixes certain roles and responsibilities amongst the classes, as well as standard protocols for their collaboration. The variability within the family of applications is factored into so-called "hotspots", and the framework provides simple mechanisms to customize each hotspot. Customizing is typically done by subclassing an existing class of the framework and overriding a small number of methods. Sometimes, however, the framework insists that the customization preserves a protocol of collaboration between several subclasses, so customization requires the parallel development of these subclasses and certain of their methods.

A framework exists to support the development of a family of applications. Typically a framework is developed by expert designers who have a deep knowledge of the application domain and long experience of software design. On the other hand, a typical application developer who reuses the framework is less experienced and less knowledgeable of the domain. However, a framework is not an easy thing to understand when one first uses it: the design is very abstract, to factor out commonality; the design is incomplete, requiring additional subclasses to create an application; the design provides flexibility for several hotspots, not all of which are needed in the application at hand; and the collaborations and the resulting dependencies between classes can be indirect and obscure.

The large learning curve faced by the first-time user of a framework is a serious impediment to successfully reaping the benefits of reuse. How can an organization address this problem?

This thesis presents a simple game framework, called SUB, as a case study for the documentation of frameworks and for the development of frameworks in C++.

Acknowledgements

I would like to thank my supervisor, Dr. Gregory Butler, for his patience and valuable guidance.

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada, and *Fonds pour la Formation de Chercheurs et l'Aide a la Recherche.*

Contents

List of Tables viii

List of Figures ix

1 Introduction 1

1.1 Why Did We Choose to Build a Framework? 1
1.2 What is this Framework? 2
1.3 The Hypothesis 2
1.4 Layout of the Thesis 3

2 Background 4

2.1 Frameworks 4
2.1.1 Developing a Framework 6
2.2 Kinds of Framework Reuse 7
2.3 Types of Documentation 9
2.4 New C++ Features Used in the Code 13

3 Framework Overview 16

3.1 Name of the Framework 16
3.2 Number of Players 16
3.3 Objective of the Game 16
3.4 Playing the Game 16
3.4.1 The Ocean 17
3.4.2 Description of the Ships 17
3.4.3 The Submarine 18
3.4.4 The Beginning of the Game 18
3.4.5 The Game Turn 19

3.4.6 Firing on the Submarine . . . 19
3.4.7 Possible Results of Firing . . . 20
3.4.8 The Next Round . . . 20
3.4.9 End of the Game . . . 20
3.5 User Interface . . . 21
3.5.1 The Abstract User Interface . . . 21
3.5.2 The Implemented Interface . . . 22

4 Framework Cookbook **24**
4.1 Recipe 1: Overview for Creating Ships . . . 24
4.1.1 Template for Parameter Description . . . 24
4.2 Recipe 2: Customizing an Existing Type of Ship . . . 25
4.3 Recipe 3: Creating a New Type of Ship . . . 28

5 Framework Design **31**
5.1 Overview . . . 31
5.2 Overview of the Design Process . . . 31
5.2.1 Application Domain Classes . . . 32
5.2.2 User Interface Classes . . . 32
5.2.3 Error Handling Classes . . . 34
5.3 Description of the Principal Classes . . . 34
5.3.1 The COORDINATE Class . . . 34
5.3.2 The OCEAN Class . . . 34
5.3.3 The SHIP Class . . . 36
5.3.4 The SUBMARINE Class . . . 36
5.3.5 The GAME Class . . . 37
5.3.6 The RANK Class and PROMOTIONPATH Class . . . 37
5.3.7 The SCENARIO Class . . . 38
5.3.8 The REFEREE Class . . . 38
5.4 Dynamic Behaviour . . . 39
5.4.1 Use Cases . . . 39
5.4.2 Collaborations . . . 42
5.4.3 Dynamics of the GAME Class . . . 42
5.5 The Referee Design Pattern . . . 44

5.5.1 *Name* : Referee Object Behavioral 44
5.5.2 *Intent* 44
5.5.3 *Motivation* 44
5.5.4 *Applicability* 46
5.5.5 *Structure* 46
5.5.6 *Participants* 47
5.5.7 *Collaborations* 47
5.5.8 *Consequences* 48
5.5.9 *Code example* 48

6 Conclusion **52**
6.1 Guidelines for Framework Documentation 52
6.2 Lessons Learnt from Case Study 53
6.3 Conclusion 54

Appendices **58**

A Source Code **58**

List of Tables

List of Figures

1 A Simple User Interface for Windows Platform 21
2 A Simple User Interface for Windows Platform 33
3 Class Diagram Showing Central Classes 35
4 Class Diagram Showing Role of User Interface Classes 40
5 Overview State Diagram for GAME Class 43
6 State Diagram for Play a Round 43
7 State Diagram for Play a Turn 44
8 Object Diagram for Referee Design Pattern 46

Chapter 1

Introduction

A framework is a collection of abstract classes that provides an infrastructure common to a family of applications. Typically a framework is developed by expert designers who have a deep knowledge of the application domain and long experience of software design. On the other hand, a typical application developer who reuses the framework is less experienced and less knowledgeable of the domain. However, a framework is not an easy thing to understand when one first uses it. The large learning curve faced by the first-time user of a framework is a serious impediment to successfully reaping the benefits of reuse.

In this thesis we will describe how to document an object-oriented application framework to encourage its reuse. In order to achieve this goal, we need a framework to document. Two options are open to us: we write a new framework or document an existing one. We choose the former option.

1.1 Why Did We Choose to Build a Framework?

In order to document framework, one must first have a deep understanding of it. Using anything that I have not created myself means that I first need to understand the framework by using the supplied author documentation, if any, and then re-describe it: not a very enlightening task considering the risk that I might not understand correctly the original documentation in which case all my work will be wrong.

Almost all existing frameworks for which the source code is available (a mandatory requirement for this work) have very scarce documentation: in order to understand

them, we must reverse engineer under-commented source code. (Some automated tools could help performing this task but none of them were available to the author.)

We need to isolate the documentation problem from other problems such as understanding legacy code, maintaining existing framework, understanding existing framework. Failure to isolate the problem would result in results which would be fallacious.

We believe that by constructing a framework that is simple enough to implement in the time available but not totally trivial we can perform some worthwhile learning.

1.2 What is this Framework?

This framework implements a simple *simulation game*, that we call SUB. This is short for "Sink that U-Boat", and is inspired by an existing game [25]. In SUB four surface ships try to sink an invisible submarine using only information about the target range. The game is designed to be played by children, so it will emphasize ease of play over realism and historical reality. The educative goals are to introduce the user to the Cartesian plane, to learn the name of the four cardinal points, to learn where north, east, south and west are on a map, to practice arithmetic skills and to build deductive skills.

The basic game is not enjoyable to play for experienced gamers but the framework is designed for reuse, so it should be quite painless to extend it into something that is both challenging and enjoyable. The framework has an abstract class for the user interface. For compiling the basic game, we have created a text mode concrete interface. This concrete interface is not part of the framework and is included only to permit the compilation and testing of the framework.

1.3 The Hypothesis

The literature review lead to the hypothesis that an application developer needs only three items of documentation of a framework in order to successfully reuse a framework:

1. The application developers will need a context for the framework, so an *overview of the framework* should be prepared, both as a live presentation and as the first recipe in the cookbook.

2. A *set of example applications* that have been specifically designed as documentation tools is required. The examples should be graded from simple through to advanced.

3. A *cookbook* of recipes should be written, and organized along the lines of Johnson's pattern language. The recipes should use the example applications to make their discussion concrete.

Furthermore, a developer who wishes to evolve the framework needs additional information about the architecture and micro-architecture. Micro-architectures are often described as design patterns, and architectures as a combination of design patterns.

This thesis tests part of the hypothesis by developing and documenting a simple framework. The chapters of this thesis follow the guidelines above, and provide a framework overview, a cookbook of recipes, and a design overview incorporating a new design pattern.

In the future, a set of graded applications should be built using the framework and experiments to verify the effectiveness of the documentation should be performed.

1.4 Layout of the Thesis

First we will present some background on frameworks and their documentation. We will then describe the game and its user interface, followed by a cookbook. We will continue with a description of the framework itself introducing a new design pattern: the Referee pattern. After that we will conclude.

We assume the reader is familiar with the *Unified Modeling Language* (UML) [27], and with the features of the draft ANSI standard C++ programming language [30].

Chapter 2

Background

2.1 Frameworks

A framework is a collection of abstract classes that provides an infrastructure common to a family of applications. The design of the framework fixes certain roles and responsibilities amongst the classes, as well as standard protocols for their collaboration. The variability within the family of applications is factored into so-called "hotspots", and the framework provides simple mechanisms to customize each hotspot. Customizing is typically done by subclassing an existing class of the framework and overriding a small number of methods. Sometimes, however, the framework insists that the customization preserves a protocol of collaboration between several subclasses, so customization requires the parallel development of these subclasses and certain of their methods.

> "a collection of abstract classes, and their associated algorithms, constitute a kind of *framework* into which particular applications can insert their own specialized code by constructing concrete subclasses that work together. The framework consists of the abstract classes, the operations they implement, and the expectations placed on the concrete subclasses" [11, p.92]

A framework exists to support the development of a family of applications. Typically a framework is developed by expert designers who have a deep knowledge of the application domain and long experience of software design. On the other hand,

a typical application developer who reuses the framework is less experienced and less knowledgeable of the domain.

> "A framework is an abstract design for a particular kind of application, and usually consists of a number of classes. These classes can be taken from a class library, or can be application-specific." [17, p.25]

A framework allows the user to reuse abstract designs, and pre-fabricated components in order to develop a system in the domain. A user may also customize existing components by subclassing. The design of the framework incorporates decisions about the distribution of control and responsibility, the protocols followed by components when communicating, and implementations for each of the major algorithms. Often the implementations are template methods that embody the overall structure of a computation and that call user's classes to perform sub-steps of the algorithm. Default implementations of each user class may be provided, and the user will subclass in order to override or specialize the operation which implements the sub-step.

A good characterization of the relationship between a framework and user's classes is "Don't call us, we'll call you." So the classes defined in the framework call the user's code, whereas the traditional use of class libraries is for the user's code to call the library classes.

However, a framework is not an easy thing to understand when one first uses it: the design is very abstract, to factor out commonality; the design is incomplete, requiring additional subclasses to create an application; the design provides flexibility for several hotspots, not all of which are needed in the application at hand; and the collaborations and the resulting dependencies between classes can be indirect and obscure.

The large learning curve faced by the first-time user of a framework is a serious impediment to successfully reaping the benefits of reuse.

Early examples of application frameworks were for *graphical user interfaces* (GUI), including MACAPP [2], and INTERVIEWS [22]. There is now an abundance of GUI frameworks that have been used successfully on many platforms ranging from DOS to UNIX, such as MACAPP for MacIntosh, OWL-WINDOWS for DOS/WINDOWS, and MOTIF for UNIX. Frameworks now exist for a broad range of application domains such as ET++, an *editor* toolkit [32] which has recently been used in MET++ which is a

framework for *multimedia* applications; RTL framework [18] for code optimization in compiler s; CHOICES for object-oriented operating systems [9]; BEE++ for analyzing and monitoring distributed programs [7]; and others for network management and telecommunications [5], and financial engineering [6].

2.1.1 Developing a Framework

An application framework evolves in response to feedback from reusers. An initial framework is based on past experience or by careful construction of one or two applications, keeping in mind the need for flexibility, reusability and clarity of concepts. Each consequent reuse points out shortfalls in these qualities in the existing framework as one stretches the architecture to accommodate the new application. By addressing the issues raised, the framework evolves, gaining flexibility, coverage of domain concepts, and clarity of the concepts and the dimensions along which they vary.

The major steps in developing an application framework can be summarized as follows [17, 31]:

1. Identify and analyze the application domain and identify the framework. If the application domain is large, it should be decomposed into a set of possible frameworks that can be used to build a solution. Analyze existing software solutions to identify their commonalities and the differences.

2. Identify the primary abstractions. Clarify the role and responsibility of each abstraction. Design the main communication protocols between the primary abstractions. Document them clearly and precisely.

3. Design how a user interacts with the framework. Provide concrete examples of the user interaction, and provide a main program illustrating how the abstract classes are related to each other and to the classes for user interaction.

4. Implement, test, and maintain the design.

5. Iterate with new applications in the same domain.

The design and implementation of frameworks relies heavily on abstract classes, polymorphism (both parametric and inclusion polymorphism), and inheritance.

When analyzing existing applications to determine reusable components and abstractions, one might re-structure the classes in order to separate what is common across applications from what is unique to one application. Johnson and Foote [17] provide a set of rules to this end: designing reusable classes.

2.2 Kinds of Framework Reuse

It is important to realize that the styles of documentation for frameworks discussed in the literature really address different audiences, or sometimes a combination of audiences. This web of conflicting aims needs to be unraveled, so here we look at the different kinds of people who reuse a framework.

Reuse by an Application Developer An application developer wants to know how to customize the framework to produce the desired application. This is a very goal-directed activity, where the main priority is to know how to do something, rather than to understand why it is done that way. The application developer needs to know the relevant hotspots, and how to customize them: that is, which classes to subclass, which methods to override, and whether combinations of classes and methods need to be specialized in unison to maintain a protocol of collaboration amongst the classes.

The need is for *prescriptive* documentation.

The application developer may not be a domain expert, nor an experienced software developer.

Reuse by a Framework Maintainer A developer responsible for the maintenance and evolution of a framework must understand the design of the framework. The internal framework design and the design rationale must be known, as well as the application domain and the required flexibility of the framework.

The are many aspects of the framework that need to be grasped: the application domain; the overall architecture and its rationale; the reasoning behind the selection of the hotspots; which design pattern provides flexibility at each hotspot; and why each design pattern was selected. Furthermore, there are the usual requirements to understand the responsibility of each class, their interface contracts, and the shared responsibility (or collaboration) of classes. Information is needed at both a high level of abstraction, and at a concrete level of detail.

The documentation must be descriptive; it cannot be prescriptive, since the original designers can rarely predict how a framework might be extended through additional flexibility at hotspots, or additional hotspots.

The developers are both domain experts and software experts.

Reuse by a Developer of Another Framework Framework developers seek ideas from existing frameworks, even if the framework is for another domain. Of particular interest are the design patterns that provide flexibility at hotspots. The developers require information primarily at a high level of abstraction, though the kinds of information needed is similar to that for framework maintainers.

The developers are expert software designers but not necessarily domain experts for the framework they are reusing.

Reuse by a Verifier Some application developers and framework developers may be concerned with the rigor of their system. They may have a need to verify certain properties of the system in order to satisfy stringent customer requirements. This requires formal methods of specification and verification.

Specification for reuse is generally more descriptive than prescriptive: the reuser is left to figure out the implications of the specification in terms of the desired customization. The main concerns are to clearly specify the obligations on a subclass and its methods that a developer may write; to specify any protocols that the developer can customize; and to specify the collaborations must be supported by the developers' new subclasses.

Summary The majority of framework reusers are application developers, and verifiers run a distant last.

What conclusions can we draw from this perspective of the many kinds of reusers? Different audiences require different information: different in the kind of information; different in the level of abstraction or detail; and different at the level of focus, either global or local. Our concern in this paper is *documentation for application developers.*

2.3 Types of Documentation

A growing body of work has been done on documenting, specifying, and reasoning about frameworks. The frameworks under consideration are often chosen from toolkits for user interfaces and drawing programs. The emphasis is on documentation rather than specification, and, with the exception of the Contracts paper [13], there is no concern for verification of correctness. Unfortunately, there is often little, or only anecdotal evidence of the impact of the style of documentation on actual reuse of the framework.

In this section we discuss the various styles of documentation used.

Examples The source code of *example applications* that have been constructed using the framework is often the first and only documentation provided to application developers. This documentation comes for free, since the example applications are created during the development process of the framework — a framework design may begin as an application that evolves into a framework, and then other applications are developed to confirm the reusability of the framework before the framework is rolled-out for general use. Such examples however are complete applications often selected for their elaborate use of features and functionality. Sparks et al [29, page 60] found that, by themselves, these examples are too difficult for novice application developers, and that the introduction to framework hotspots needs to be more incremental, gradually going from the simplest forms of reuse to more advanced forms.

Documentation requires a graded set of training examples. Each should illustrate a single new hotspot, starting with the simplest and most common form of reuse for that hotspot, and eventually providing a complete coverage. Linn and Clancy [21] offer valuable advise on designing and using examples. The ET++ framework comes with an extensive set of example applications. Most cookbooks (see the next section) revolve around a small number of simple example applications.

Cookbooks and Recipes A *recipe* describes how to perform a typical example of reuse during application development. The information is presented in informal natural language, perhaps with some pictures, and usually with sample source code. Although informal, a recipe often follows a structure, such as sections on purpose, steps of the recipe, cross references to other recipes, and source code examples.

A *cookbook* is a collection of recipes. A guide to the contents of the recipes is generally provided, either as a table of contents, or by the first recipe acting as an overview for the cookbook.

Patterns provide a *format* for each recipe, and an *organization*. The organization follows a spiral approach where recipes for the most frequent forms of reuse are presented early, and where concepts and details are delayed as long as possible. The first recipe is an overview of the framework concepts and the other recipes. Johnson [16] introduced an informal *pattern language* that can be used for documenting a framework in a natural language. The documentation of a framework consists of a set of patterns where each pattern describes a problem that occurs repeatedly in the problem domain of the framework, and also describes how to solve that problem. Each pattern possesses the same format. The elements of a pattern are: description of its purposes, explanation of how to use it, description of its design, and some examples.

Lajoie and Keller [19] introduce the term *motif* for Johnson's patterns in order to avoid confusion with design patterns. They use a template for a motif description that has a name and intent, a description of the reuse situation, the steps involved in customization, and cross references to motifs, design patterns, and contracts. The design patterns provide information about the internal architecture, and the contracts provide more rigorous description of the collaborations relevant to the motif.

Active cookbooks [28] support the developer by combining the cookbook recipes with a visual design and development environment.

Cookbooks have been used with several frameworks, such as MVC (Model-View-Controller), MACAPP [20], HOTDRAW [16], ET++ , MET++ , and Taligent's CommonPoint framework [10]. Many application developers have successfully learned a framework from a cookbook and the framework source code. Johnson [16, page 67] states that his cookbook is " the only documentation for a version of HotDraw that has been distributed since early 1992, and users say they are satisfied with it. "

Contracts A *contract* is a specification of obligations and collaborations. While the traditional *interface contract* [24] of a class provides a specification of the class interface and class invariants in isolation, an *interaction contract* [13, 14] deals with the co-operative behavior of several participants that interact to achieve a joint goal. A contract specifies a set of communicating participants and their contractual obligations: the type constraints given by the signature of a method, the interface semantics

of the method, and constraints on behavior that capture the behavioral dependencies between objects. A contract specifies preconditions on participants required to establish the contract, and the invariant to be maintained by these participants.

Originally framework contracts were viewed as a mechanism to compose behavioral descriptions given by subcontracts. Contracts can be refined by either specializing the type of a participant, extending its actions, or deriving a new invariant which implies the old. Consequently, the refinement of a contract specifies a more specialized behavioral composition.

The role of contracts in Lajoie and Keller's [19] documentation of ET++ is much more pragmatic than the original intent of Helm et al. A contract supports a cookbook recipe with additional rigor in case a developer needs to consult a specification of collaborative behavior of classes.

Design Patterns A *design pattern* presents a solution to a design problem that might arise in a given context [12]. A design pattern provides an abstraction above the level of classes and objects. Design patterns capture design experience at the micro-architecture level, by specifying the relationship between classes and objects involved in a particular design problem. A design pattern is meta-knowledge about how to incorporate flexibility into a framework.

The description of a design pattern explains the problem and its context, the solution, and a discussion of the consequences of adopting the solution. The problem might be illustrated by a concrete example. The solution describes the objects and classes that participate in the design, and their responsibilities and collaborations. A collaboration diagram may be used to represent the same information. Examples of the solution being applied in concrete situations may be provided. The analysis of benefits and trade-offs of applying the pattern is an important part of the design pattern description.

Beck and Johnson [4] illustrate the use of design patterns in developing the architecture for HOTDRAW , a framework for drawing editors. Design patterns are good at describing architectures, but a close look at the patterns used by Beck and Johnson for HOTDRAW show that they are of little use to an application developer, since they deal with internal structure of the framework.

Lajoie and Keller [19] relegate design patterns to a support role for recipes. A design pattern illustrates relevant architectural issues, in case the application developer

needs a deeper understanding of a recipe.

Framework Overview Setting the context of a framework is a first step in helping an application developer reuse a framework. The jargon of the domain can be defined and the scope of the framework delineated: just what is covered by the framework and what is not, as well as what is fixed and what is flexible in the framework. A simple application can be reviewed, and an overview of the documentation can be presented.

Such an overview is often the first recipe in a cookbook, though, in the case of a framework developed in-house, a live presentation by the framework developers offers an opportunity to field questions from the application developers.

Reference Manual A *reference manual* for an object-oriented system consists of a description of each class, together with descriptions of global variables, constants, and types. Typically, a class description presents the purpose or responsibility of the class, the role of each data member, and some information about each method. A method description presents the functionality of the method, its pre- and post-condition, and an indication of which data members it affects or uses. The description of a class may be organized as a Unix `man` page.

For framework documentation, the descriptions can include additional material concerning the role of a class or method in providing flexibility for a hotspot, particularly whether a class is intended to be subclassed or a method to be overridden.

Traditional techniques for modules, such as the LARCH family of interface languages can be used for describing class interfaces and extended to include the obligations on subclasses as in LARCH /C++ . The EIFFEL language supports design-by-contract [24] through the declaration of assertions, preconditions, and postconditions.

Reference manuals by themselves are not a very useful way to learn a framework.

Design Notebooks A *design notebook* collects together information related to the design of hardware. The information will include background theory, analyses of situations, and a discussion of engineering trade-offs. While not specifically intended for frameworks, Schlumberger [3] has adopted this approach with issue-driven design (of Potts and Brun) to capture the design rationale of software systems, as well as

hardware systems and combined hardware/software systems. They call them *technology books* and *product books*. The information includes requirements, specifications, architecture, components, design, code, history and the relationships between this information. Background theory or domain information and analyses of trade-offs are crucial information. A hypermedia system supports access and navigation of the books.

Other Recipes describe how to adapt the functionality of the framework. As such they may refer to, or be documented in terms of, *use cases* or *scenarios* [15] that describe the intended functionality. Similarly, a *time thread* [8] for a scenario can depict when and where the scenario involves the framework and when and where it involves the customized code.

2.4 New C++ Features Used in the Code

The C++ language has evolved to the extent that now there is a draft ISO C++ standard. Several of the new features may be unfamiliar to even experienced C++ programmers, and are used in our code, so a brief discussion is given here. Unfortunately, not all compilers support all the features of the draft standard.

Exception handling The `try`, `throw`, and `catch` statements, together with exception classes, allows programmers to respond appropriately to abnormal or atypical situations during a computation.

Runtime type identification (RTTI) A program can query an object about its dynamic type, in order to distinguish between objects of different subclasses. The `type_info` class provides the following methods for this purpose:

```
namespace std {
    class type_info {
    public:
       virtual ~type_info();
       bool operator==(const type_info& rhs) const;
       bool operator!=(const type_info& rhs) const;
```

```
        bool before(const type_info& rhs) const;
        const char* name() const;
    private:
        type_info(const type_info& rhs);
        type_info& operator=(const type_info& rhs);
    };
}
```

There are standard exception classes that may be thrown when utilizing the run-time type identification, namely:

```
namespace std {
  class type_info;
  class bad_cast;
  class bad_typeid;
}
```

Standard Template Library (STL) The STL contains generally useful classes and algorithms, such as various container classes — set, multiset, vector, list, queue, etc — implemented in a *generic* fashion using templates.

The STL has the following features:

- It is fast. The STL implementation uses no virtual functions nor inheritance. The C++ standard puts performance constraints on the implementation.
- The STL makes heavy use of iterators, which are a kind of generalized pointers. The iterators are used to access the contents of a container in the same manner that C pointers are used.
- It has consistent syntax across the different container classes, iterators, and algorithms.
- The functions that implement the algorithms that work on the collections are not members of the collection classes. They are defined separately and they use *iterators* to access the members of a container. This indirect approach allows the algorithm to work with regular C arrays as well as STL containers. It also

permits modification of the algorithm without any modification to the container classes.

- The STL does not use the new and delete operator to allocate memory. The STL uses special objects called *allocators* instead. The default implementation of an allocator will allocate on the heap, but a programmer can replace the standard allocator object with their own, thus taking control of the memory allocation procedure without any modification to the container or the algorithm code.

Namespace A namespace encapsulates a scope, or set of declarations, so that the same identifier can be used in separate parts of a program without confusion or ambiguity. For example, the namespace `std` is used for the standard library.

const_cast This language feature is used to add or remove the `const` and `volatile` modifier of pointers or references.

Getting rid of the constantness is not new, Stroustrop has already mentioned it in his (second edition) book. The old way of doing thing was to use a normal type cast but this was making it hard to guess if the constantness was removed: for example if you see `(int *) c` there is no way to know the type of `c` and to know if it is constant without searching for its declaration. The main benefit of `const_cast` is that it makes it clear to both human and computer that constantness has been modified.

Boolean values type bool The built-in type `bool` defines the logical Boolean values `false` and `true`.

Chapter 3

Framework Overview

3.1 Name of the Framework

The name of the framework is SUB, standing for Sink that U-Boat.

3.2 Number of Players

The game can be played by one, two or four human players, but playing alone makes the game far less challenging.

3.3 Objective of the Game

Each player is a member of the same Navy and each commands a *ship*. The goal of each player is to personally sink the enemy *submarine* since only one captain can have this glory. Navy regulation forces each ship to *communicate to the other ships* all the information they have on the submarine range. Nevertheless, commanding officers do not share any insight they may get on the depth of the submarine.

3.4 Playing the Game

The object of the basic rules is to familiarize the beginning player with the system. With the basic rules, a player's ship is never destroyed, it is only disabled for a short time. The submarine will open fire only in reply to an incorrect shot by the players.

The submarine's path and evasive action is quite simple and highly predictable by an experienced player. The players' vessels have an unlimited amount of supplies (fuel, ammunition, food and fresh water for the crew). The ships do not suffer permanent damage when they run aground: instead they come to a full stop at the edge of the ocean even if the player mistakenly gives an order that would result in running aground. All ships running aground will be automatically repaired after one game turn.

3.4.1 The Ocean

The ocean is a flat square area that is represented as a Cartesian plane. Ships are played on the intersections. All coordinates are positive. The x axis coordinate values is called the longitude and the y axis value is called the latitude. Ships can only occupy positions labelled by integer numbers. The minimal size of the ocean is 50 by 50. The minimal playable area is 46 by 46. The two first positions all around the map are sections of ocean that are too shallow to allow a sub to move underwater, but deep enough to allow a surface vessel to cross them. They can offer shelter to damaged ships. Ships are placed on the intersections of the sea and their displacement is measured by the number of intersections they cross. Only eight headings are supported for movement: N, NE, E, SE, S, SW, W, and NW. Firing is supported in sixteen headings: N, NNE, NE, ENE, E, ESE, SE, SSE, S, SSW, SW, WSW, W, WNW, NW, and NNW.

3.4.2 Description of the Ships

A ship is characterised by the following information:

1. Its unique identification number, ranging from 1 to 4.
2. Its *name*, chosen by the player.
3. Its *category*: PT-boat, destroyer, cruiser and battleship.
4. Its *current speed*: measured in the number of intersections it can cross in one turn.
5. Its *current heading*. One of the eight aforementioned ones.

All ships depart from ports located near the south-west corner of the map, at speed 0, and are heading north east.

6. Its turning rate, indicates the maximum amount, in degrees, by which the current heading can be changed in one turn.

7. Its *maximum speed*, in number of intersections.

8. Its *acceleration rate*, the maximum amount by which the ship speed can be modified during one turn.

9. Its *firing range*, which is uniformly two for all ships in the game

3.4.3 The Submarine

The submarine starts from a random allowable (non-shallow) ocean intersection. Thereafter, it moves one square at a time always in the same direction. Upon reaching the edge of the ocean, the submarine will randomly select one of the three possible courses moving away from the boundary. If fired upon, the sub will, randomly, either maintain its current course or make a 45 degree turn to its left or to its right. The same evasive maneuver can be performed even if the attacking ship has not aimed properly at the sub. The sub will *launch torpedoes* at any ship that is attempting to hit the sub. The ships always have the first strike: the sub cannot reply if it has been damaged. Note that the submarine *is a* ship with the following properties: turning rate:45, maximum speed:1, acceleration rate:1 and firing range:2. The submarine has a property that ships do not have: *depth*. Depth is measured in arbitrary units, 3 being the deepest. In the basic game, depth is always 1, 2 or 3 and every (non-shallow) position in the ocean is at least of depth 3.

3.4.4 The Beginning of the Game

Each ship, in the order of their unique numeric identifier, play one turn. This continues until the submarine is sunk, or the human players give-up, in which case they all lose.

3.4.5 The Game Turn

During a turn, a player is given the possibility to *alter course and speed*, *move the ship* and *fire* **one shot** *at the submarine*, if the target is in firing range. Here is the detailed procedure for a turn.

1. The player gets the actual range between its current ship position and the objective. If the range is reported as Fx where x is an integer, it means that the submarine is within firing range.

2. The player *may* fire at the sub if it is in range (see the firing procedure below)

3. The actual course and speed are displayed on the screen and the player *may* set the new heading and course, if he wants. They must be consistent with all constraints imposed by the ship design specification. If they are invalid, the player will receive an error message and he will be continually queried until the answer is acceptable. If the captain's order would result in a *collision* with another friendly ship, then the *first officer* will take command and override your judgement. While you discuss with him, nobody is in command, and the ship changes heading to a new random direction that is consistent with its turning rate. If the new course would result in another collision, your ship will be put in a completely random heading, that *does not* need to be consistent with the ship limitations. You will then lose your next turn because your ship must be repaired before it can move again.

4. An update on the range of the target will be displayed.

5. If you have not fired at step 2, you can do so now, under the same conditions.

6. The submarine moves.

7. The next ship, in numeric order, plays its turn.

3.4.6 Firing on the Submarine

As soon as you are in range, you may fire. You are *never forced to fire.* Because the submarine returns fire if you miss it it, is highly recommended that you do not fire blindly if you have no idea where the target is. If the distance is F0 (you are right

above the objective) you can fire without specifying any heading, if not, the correct heading must be given. Your well-trained crew will not fire at an *out of range target.* The user interface will disable the firing controls until there is a target in range.

3.4.7 Possible Results of Firing

After firing the three possible results are:

1. **The submarine is sunk**, this round is over and the player that has sunken it is promoted, to a more powerfull ship or he wins the game if there is no such ship.

2. *Offn* where n is a positive, non-zero integer. You fired at the right position but the wrong depth. The target depth is the depth at which you fired *plus or minus* n. For example, if you fired at depth 1 and get an Off2 result, then the submarine must be at depth 3 (1+2) as depth 5 is non-existent in the basic game.

3. **SOS** *Your* ship is sending a general distress call because you have fired at the wrong place, in which case the submarine never misses you. You will lose one turn repairing the ship and the naturally occurring ocean current will give your ship a new random heading and speed, that does not need to be consistent with your ship constraints.

3.4.8 The Next Round

If the submarine is sunk by a ship other than a *battleship*, the next round begins by randomly placing a new sub. The ships restart from their original positions. The player who sunk the sub gets the next bigger ship and the round is played exactly like the first one.

3.4.9 End of the Game

The winner is the first player who sinks a submarine with a battleship.

3.5 User Interface

For this project, the user interface is not the most important part of the work, consequently, we do not spend a lot of time developing it. For this reason, a simple text mode interface will be developed, as shown in Figure 1.

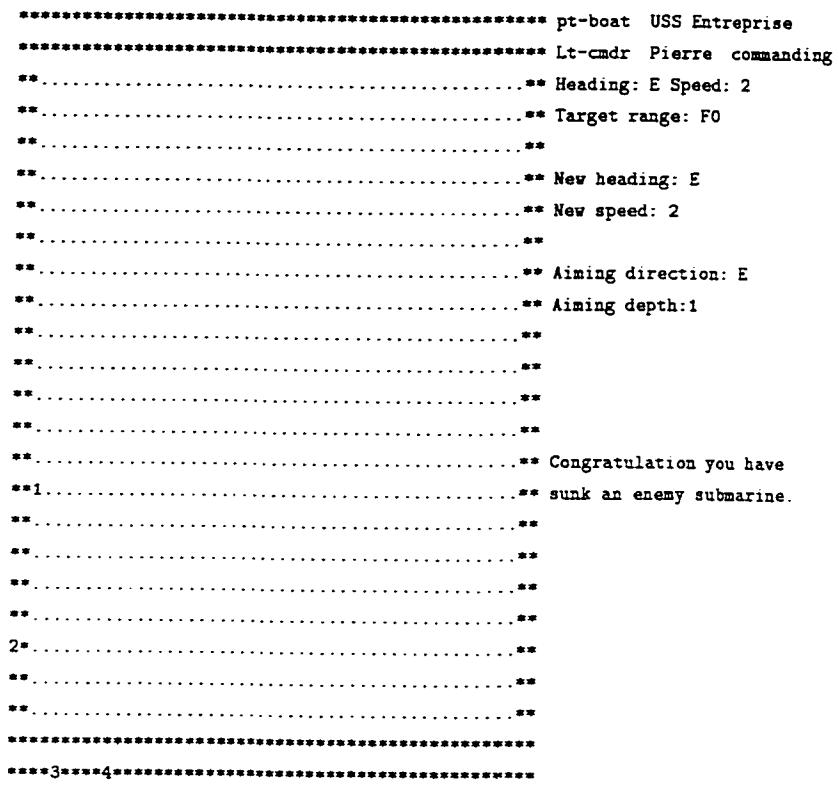

Figure 1: A Simple User Interface for Windows Platform

3.5.1 The Abstract User Interface

The abstract class TScreen is used as an interface between the framework and the user. Except for the constructor and destructor (which cannot be virtual), it is composed solely of pure virtual functions. A concrete subclass of TScreen, TtextScreen, is used to test the framework. The abstract interface divides the screen into four areas.

1. The *ocean area* presents the player with an image of the ocean and the ships. This area might have scroll bar if the whole ocean is too big to fit on this part of the screen. This area is usually the biggest part of the screen.

2. The *ship information area* displays all the useful information about a ship, such as speed, heading, commanding officer, etc. When the framework is extended by adding new features to the ships, this area might need to be updated.

3. The *information area* reports the result of all *legal* player actions.

4. The *infraction area* displays messages about infractions and its background colour indicates whether or not an infraction has occurred. It initially has a *green background.* When a player has committed an *infraction*, such as setting a course that would result in a collision with another ship, the *background of this area will turn red* and an appropriate message will be displayed. The change of color is performed to attract the *attention of the user* and the choice of the red and green color is consistent with the user interface development guidelines. The informationn area and the infraction area may share the same portion of the screen.

3.5.2 The Implemented Interface

The actual interface responds to the followings keystrokes:

- **The arrow keys** The arrow keys are used to set the ship's *future* course and speed. They have no effect on the ship until the user press the "M" key to move the ship.

  - **up arrow** is used to *increase* the ship speed. If the speed is already equal to the ship maximum allowable speed, pressing this key has no effect.
  - **down arrow** is used to *decrease* the ship speed. If the speed is already equal to zero, pressing this key has no effect.
  - **right arrow** is used to *turn the ship 45 degrees to the right.* Repeated used of this key will result in adding an extra 45 degrees to the right. If this key is depressed eight times, the ship heading will return to its previous setting.
  - **left arrow** is used exactly like the right arrow, except that is will execute a *turn of 45 degrees turn to the left.*

- The "M" key, the move key, is used to make the ship move at the speed and heading indicated on the screen. If such a move is illegal, the infraction window will change from green to red and an informative message will appear.

- The "F" key, the fire key, is used to open fire on the enemy submarine. When the target is within firing range, the range written on the screen will be prefixed by the letter F. Depressing this key when there is *no target within firing range has no effect.*

- Firing interface. In order to sink the sub, a player must first enter the depth at which the explosives will be detonated, this depth is either 1, 2 or 3. What happens depends on the target range: if the range is zero, a depth charge is immediately release without any further prompt; otherwise, the aiming direction will be shown. Originally, it will be set to the north but the user can change it by using the right arrow and left arrow key exactly in the same manner used to set the heading of a ship, the only difference being that there will be 16 possible aiming directions, instead of 8 heading directions.

- Any key may be pressed when an on-screen message informs the user to "press any key to continue". This is used to delay the program in order to give the player time to read messages that are written on the screen.

- the "A" key is use to *abandon the ship.* A confirmation will be requested. Abadonning a ship is the equivalent of resigning. The player who abandons his ship loses the game. Abandonning all ships is the only way to end the game before its normal termination.

Chapter 4

Framework Cookbook

This chapter is an example of a cookbook. A cookbook is a collection of recipes, each describing in practical terms how to use a framework.

4.1 Recipe 1: Overview for Creating Ships

There are two basic ways to easily create new ships:

Recipe 2: Customizing an Existing Type of Ship which selects an existing type of ship and customizes the parameters for its construction; and

Recipe 3: Creating a New Type of Ship which defines a new type of ship that fits within the general range of abilities and rules for behaviour of current ships, and then follows Recipe 2.

4.1.1 Template for Parameter Description

The recipes often describe classes, methods, and their parameters. To increase readability we will describe each parameter using the following template:

- <Parameter name> The name of the parameter.
- <Category> A measure of the impact on the game that a modification of this parameter would have. There are two categories

- *Decorative* The value of this parameter does not affect the game in any fundamental manner. Decorative features are the only difference between two ships of the same type. For example, the ship's name is decorative.
- *Fundamental* The value of parameters of this category has a direct impact on the game. All ship of the same type share the same fundamental attributes.

- < Difficulty: hard/medium/easy > The difficulty of the customization: *hard* means that, in order for the framework to remain consistent, changes to *other classes* would be required if the parameter value is changed from the default value; *easy* means that no changes in the code are required; *medium* means that change to the implementation of the current class might be required if the parameter is changed, thus *subclassing* is mandatory if the original code is not to be changed.

- <Range> The set of allowable values for a parameter. Note that the range might be dependent on limits set in other classes.

- <Exception list> A list of exceptions that might be thrown if the value of the parameter is out of range or if the class invariant, if any, is violated by the value of the parameter.

4.2 Recipe 2: Customizing an Existing Type of Ship

The simplest way to construct a new ship is to select one of the existing types of ships and make minor customizations by supplying parameter values at the time of construction. For example,

```
TShip* myship = new TShip( destroyer, Player3, "USS Missouri" );
```

selects the type `destroyer` for the new ship, and customizes by indicating which player is the commanding officer, and what is the name of the ship.

The constructor for the Tship class that is used in the example is

```
TShip(const TShipType AShipType,
            TPlayer  *CommandingOfficer,
      const string   &ShipName="",
      const unsigned Longitude=0, const unsigned Latitude=0,
      const TShipAllignment AnAllignment=allied,
      const unsigned Depth=0 );
```

The first three parameters are the ones that must be supplied when creating a new ship, as the remaining parameters have default values that can be used as-is. The three main parameters are:

1. Ship Type

   - <Parameter name: ShipType> The *type* of the ship. The ship's type is used to generate all the vital statistics, such as maximum speed, weapon range, etc for the entire game.
   - <Category: fundamental> Considering the vast differences that exist between the various type of ships, this parameter is very important.
   - <Difficulty: easy> There are two possible customization at this level: it is *easy* to change the value of this parameter to another legal value of type TShipType. It is also *easy* to create and register a new type of ship (see Section 4.3).
   - <Range: TShip > ShipType is a member of an enumerated type declared inside the TShip class

     ```
     typedef enum {pt_boat, destroyer, cruiser,
                                battleship, submarine} TShipType;
     ```

   - <Exception list> TshipTypeIsntRegistered will be thrown if no information about this type of ship exists in the registration database.

2. Commanding Officer

   - <Parameter name: CommandingOfficer> It is a pointer to the player that is commanding this ship. It must not be NULL.

- <Difficulty: Hard> Changing the controlling player has subtle effects on other classes: the position of the player on the *promotion path* is modified, and the class of ship that a commanding officer is allowed to command will change.
- <Category: fundamental> In the basic game the commanding officer has *no direct effect whatsoever* on the ship's behavior. This might not be the case in advanced versions of the game.
- < Range: Not Applicable> A pointer can have any address.
- <Exception list>: TAShipMustHaveACommandingOfficer is thrown if the parameter is NULL.

3. Ship Name

- <Parameter name: ShipName> The ship's name, such as HMCS Huron, or USS Enterprise.
- <Category: decorative > The ship's name is displayed on the screen when it is that ship's turn to move.
- <Difficulty: easy > Just set to any value in the acceptable range and the job is done.
- <Range> The string may contain any printable character.
- <Exception list> Not Applicable.

Some words of caution on how Ship objects may be constructed, or modified, are in order: (1) A default constructor has not been defined so there will be no way of creating an array or collection of TShip objects. This is not an oversight, when an array of this type would be required, an Array of Tship* should be used. (2) Not all attributes have access functions defined. This is not an oversight but a way to ensure respect of the rules. For example, there is no SetCommandingOfficer function, thus ensuring that the attribute is not modified after a ship has been sunk to unfairly give the player a second chance.

4.3 Recipe 3: Creating a New Type of Ship

To create a new type of ship, say a frigate, then there are two steps that must be carried out. First, the enumerated type TShipType must be modified to include the new type `frigate` in an approriate place in the order, for example,

```
typedef enum {pt_boat, frigate,  destroyer, cruiser,
                        battleship, submarine} TShipType;
```

Secondly, the performance characteristics of the new type of ship must be registered using the static function RegisterANewShip. For example,

```
RegisterANewShip( frigate, 4, 90, 2, 2 );
```

specifies that a frigate has a maximum speed of 4, a maximum turning rate of 90 degrees, a maximum acceleration rate of 2, and a weapons range of 2.

The function RegisterANewShip can also be used to change the performance characteristics of existing ship types, such as `destroyer`.

The static function RegisterANewShip updates a map within the TShip class. The map records the characteristics of each ship type. The signature of RegisterANewShip, and an explanation of its parameters, are:

```
bool RegisterANewShipType(
    const TShipType AShipType,
    const unsigned MaxSpeed=DefaultMaximumSpeed,
    const unsigned MaxTurn=180,
    const unsigned MaxAccel=DefaultMaximumSpeed,
    const unsigned WRange=2 );
```

1. Ship Type

   The type of the ship, as described in Section 4.2.

2. Maximum Speed

   - <Parameter name: MaxSpeed>. The maximum speed at which the ship can move.

- <Category: fundamental > This factor is one of those that will set the difficulty level of the game. Note that any ship that is *slower then its objective* is at a great disadvantage, for this reason it is recommended to keep the basic game speed at a minimum of 4, the speed at which the target submarine is moving in a four player game. The speed of the sub is 1, but it moves once before each player's turn.
- <Difficulty: easy > Just set to any value in the acceptable range.
- <Range: [1..9]> The ocean must be such that the ships cannot cross it in one turn. Furthermore, too fast a speed would ruin the pleasure of playing the game.
- <Exception list> Not Applicable. A ship with a speed of zero would be totally useless, so if the user request zero, the value will be silently changed to 1 without any warning nor error message.

3. Maximum Turn Rate

- <Parameter name: MaxTurn>. The maximum turning rate of the ship, expressed in degrees. measures how much the ship can turn during a single game turn.
- <Category: fundamental > This factor is one of those that will set the difficulty level of the game. Note that any ship that is *less maneuverable then its objective* is at a disadvantage.
- <Difficulty: easy > Just set to any value in the acceptable range.
- <Range: [45..180]> For the basic game, movement is possible only along the following compass directions: N, NE, E, SE, S, SW, W and NW. Consequently, the minimum acceptable value is 45 degrees.
- <Exception list> Not Applicable. If the value is less then 45, it will be replaced by 45. A value greater then 180 would indicate the ability to do more then a 180 degree turn in the same turn, such a value would be automatically be reduced to 180.

4. Maximum Acceleration Rate

- <Parameter name: MaxAccel> The maximum acceleration rate of the ship measures how much a ship's speed may be altered during a single game turn. For example, if a ship begins its turn at speed 6 and has a maximum acceleration rate of 3, its speed after this turn could be either 3, 4, 5, 6, 7, 8 or 9 provided that the speed remains below it maximum speed.
- <Category: fundamental > This factor is one of those that will set the difficulty level of the game. Ships that have high speed but feeble acceleration rate are poorly suited for hunting a more mobile target because they will require many game turns to effect any change of speed.
- <Difficulty: easy > Just set to any value in the acceptable range.
- <Range: [1 .. 9]> There is no reason for setting an acceleration rate higher than the maximum speed.
- <Exception list> Not Applicable. If zero is used as the argument, it will be replaced by 1, silently and without any error message.

5. Maximum Weapons Range

- <Parameter name: WRange>. The range of the ship's weapons measures the maximum distance at which the ship weapons can deliver ordinance.
- <Category: fundamental > This factor is one of those that will set the difficulty level of the game.The longer the range, the easier it will be to sink the enemy without taking much risk. If a ship's weapon range is greater than that of its target, then it has a huge advantage. Conversely, it would have a great disadvantage to have a shorter range. When assigning this value, it is useful to remember that the default target submarine has a weapons range of 2.
- <Difficulty: easy > Just set to any value in the acceptable range.
- <Range: [0..2]> If zero is used as the argument, it means that the ship must be located directly above the submarine in order to open fire. Weapon range is limited to 2 for sake of simplicity, as this allows the use of the eight compass points as firing directions: a range over 2 would therefore complicate the interface.
- <Exception list> Not Applicable.

Chapter 5

Framework Design

5.1 Overview

This chapter describes the major design decisions for the SUB framework. An overview of the classes, their associations and collaborations is presented first, using the UML notation [27]. The central design decision is the use of the Referee design pattern to enforce the rules of the game. This original design pattern is described separately in Section 5.5.

5.2 Overview of the Design Process

The design process that we followed was an ad hoc iterative approach, first identifying the obvious entity classes from the application domain, while at the same time realizing the need for a REFEREE class to mediate the players' actions. Other classes are required for the error handling mechanism, which is heavily dependent on C++ exceptions, and for a simple user interface.

At all times, we were concerned with design for reuse, and especially customization of the ships available to players.

We discuss the classes that were identified from the application domain, the user interface, and the error handling mechanism in the following subsections.

5.2.1 Application Domain Classes

An examination of the domain of the SUB game enabled us to find a list of candidate classes: SHIP, SUBMARINE, PLAYER, MILITARYRANK, OCEAN. Background knowledge indicates that we will need a Coordinate class to label each ocean position. The chief arbiter of the game is the REFEREE class. Its duties are: to get the orders from the player, to inform the other classes the consequences of their actions (movement, firing, running aground), to disable incapacitated ships, to declare the winner of each round, to declare the winner of the game, and to maintain the state integrity of the classes under its responsibility. A GAME class will be responsible for ensuring that the sequence of play is properly followed. The GAME class is a minor referee, like a linesman in hockey. In the basic game it can only claim two infractions: that a player is attempting to move for the second time in the same turn, or that a player is attempting to open fire twice during a single game turn. The GAME class also controls the INTERFACE class in order to write to the screen and read from the keyboard. The PLAYER class stores the player name and rank. RANK is a class by itself representing the military ranks, while the PROMOTIONPATH class maintains the military hierarchy as an ordering of the ranks.

Iteration revealed the need for several support classes in the application domain. The SCENARIO class describes the game being played in terms of initial positions, the rules of engagement, and the objectives of players, in order to permit easy modification of the simulated scenario. The SCENARIO class contains three other classes: the INITIALPOSITION class stores the position of all ships at the beginning of the scenario; a RULESOFENGAGEMENT class ascertains if a ship can open fire on a given target; and an OBJECTIVE class contains the list of goals that both sides must achieve in order to win.

5.2.2 User Interface Classes

We need a class to interact with the player. This class should enable the player to visualize the ocean and the player's ship, the other players' ships, and all the known ocean features such as water depth. This class is called the SCREEN class. The interface must also be able to receive orders from a player for their ship and to display the result of the execution of those orders. A view of a simple user interface for a Windows platform is shown in Figure 2.

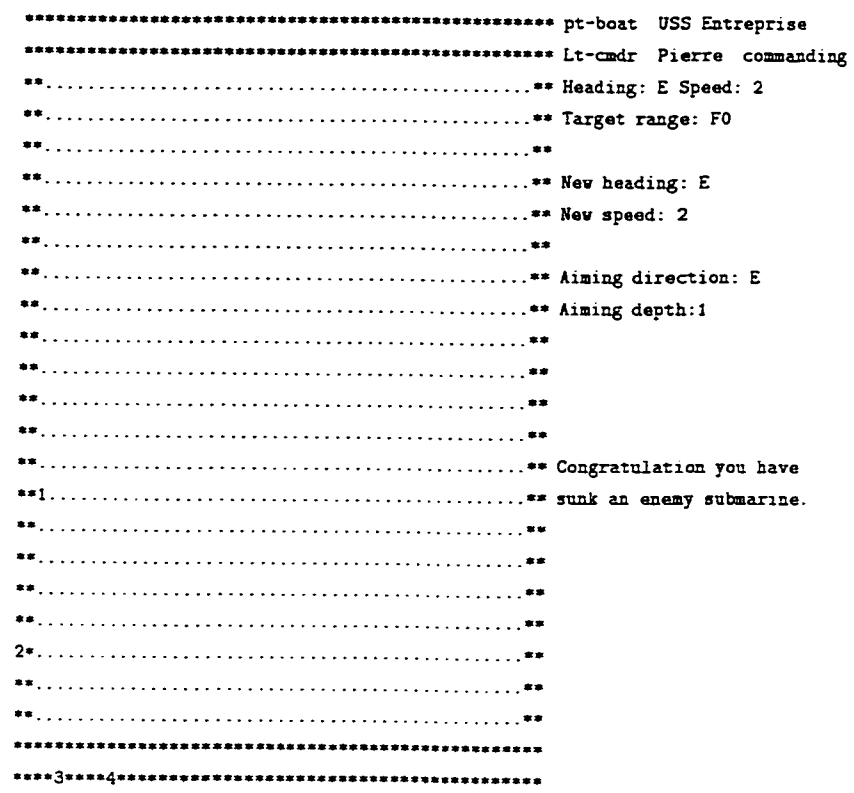

Figure 2: A Simple User Interface for Windows Platform

The screen has been divided into many areas, that will ultimately translate into many windows. The largest zone of the screen will be set apart to show the ocean. If the ocean cannot fit entirely into the screen scroll bar must be added to enable the viewing of the entire playing area. A second section is reserved for displaying information about the currently active ship. A third section of the screen is used to display aiming information when the player decides to (and can legally) open fire. The fourth section of the screen is used to write the messages from the referee. The background of this section should be green when writing a informative message and red when notifying the player of an infraction.

Windows 3.1 does not support stream IO, so *use of* the insertors — the output operators, ostream& operator <<(...) — of various classes, including the error handlers, should be restricted to concrete subclasses of SCREEN. *All the machine dependent code is located here in the* SCREEN *class, the rest of the framework should be portable to any ISO draft standard compliant C++ compiler.*

5.2.3 Error Handling Classes

The greatest number of implementation classes are those that are used to report errors. Heavy use had been made of C++ exceptions. All the classes thrown as exceptions are a subclass of an abstract class, TSUBERROR. TSUBERROR has two public subclasses, one for initialization errors, and one for logic errors. Both are abstract classes, and no class inherits from both of them. In each non-abstract error handling class, the *What()* member function will write an explicative message into string variable. The message can be written to the screen by using the error-handling class extractor, if the operating system permit it, otherwise, the content of the variable should be copied into an appropriate widget for screen output. Those messages are there to help a maintenance programmer to fix any problem that may arise; they are not intended to be understood by the user of the program.

We have used this approach because the SUB framework has been designed for reuse. If we were only interested in satisfying the minimal requirements of the problem then a much simpler design could have been used.

5.3 Description of the Principal Classes

The main classes are shown in Figure 3 as a UML class diagram. This diagram highlights the entities in the game domain, and ignores the classes for error handling and the user interface. An overview of the user interface classes, and their associations to those in the game domain is shown in Figure 4.

5.3.1 The COORDINATE Class

A *coordinate* is composed of longitude, latitude and depth. A depth of zero (0) corresponds to the surface of the ocean, while a positive depth is underwater. Depth is measured in arbitrary units.

5.3.2 The OCEAN Class

The OCEAN class is used to represent the ocean and the static (non-moving) entities in or on the ocean. The static entities include water and shallow water in the present

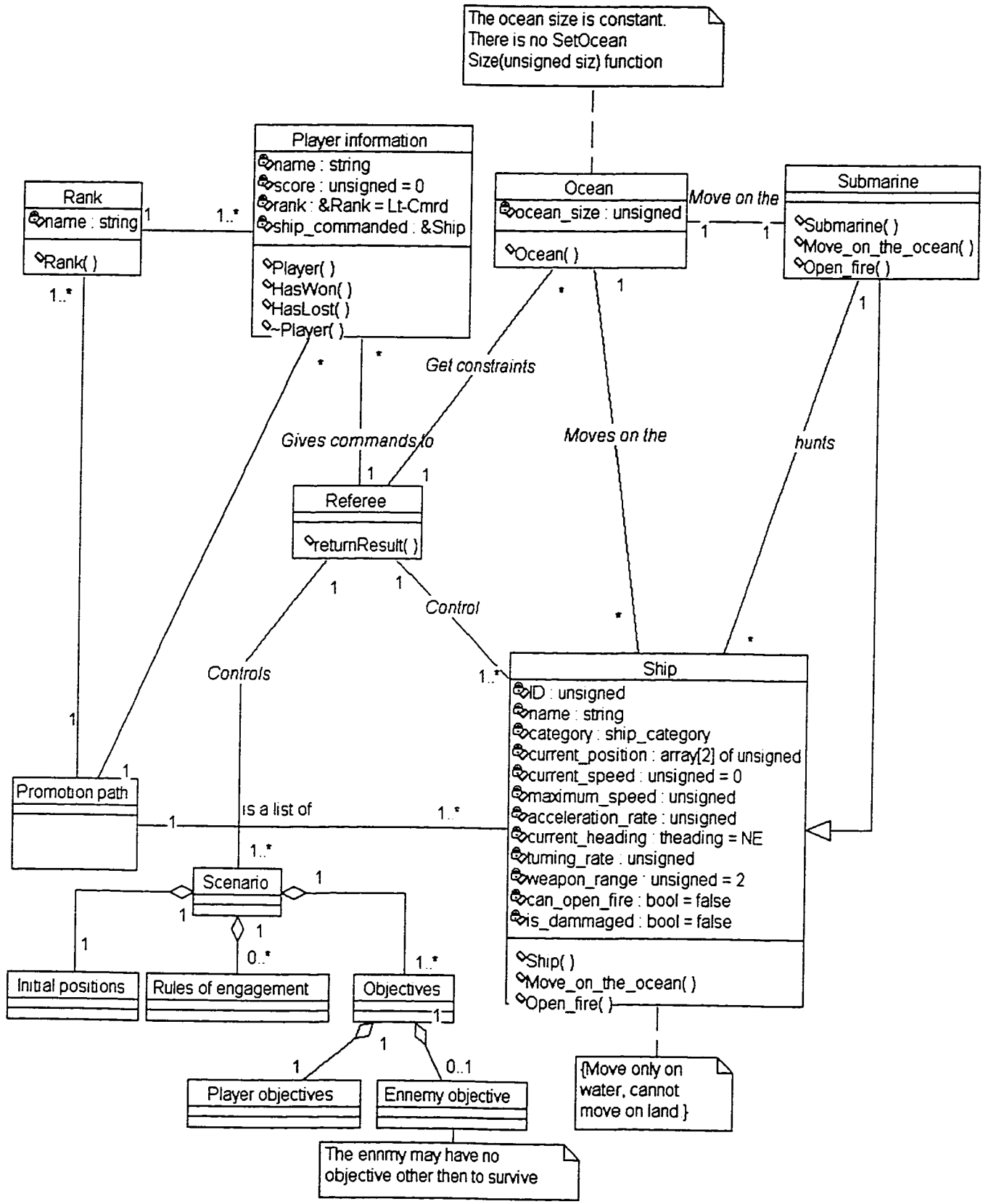

Figure 3: Class Diagram Showing Central Classes

basic version of the game, but could be extended to include islands, explosive mines, and other obstacles.

Moving objects on and in the ocean, such as surface ships and submarines are not the responsibility of this class.

The OCEAN class implements the *IsNavigable* function that determines if a given coordinate can be safely occupied by a ship.

5.3.2.1 The OCEANREPRESENTATION Class

For efficiency reasons, the representation of the ocean has been completely separated from the OCEAN class and placed in the OCEANREPRESENTATION class. This is based on the fact that more then 90% of the ocean is water, so using a non-sparse matrix to represent the positions would waste space. Also, it is often the case that a simple function can return the nature any ocean square, thus removing the need to store it. The representation can now be varied without modifying the ocean class.

5.3.3 The SHIP Class

In any game of this nature, the ships, along with the rules, are the most important features. The SHIP class represents a ship that can move on or in the ocean. In this class we store all the vital information about a ship: current position, damage status, ability or inability to open fire, maximum speed, maximum turning rate, maximum acceleration rate, and the commanding officer (an instance of the PLAYER class).

At all times, a ship must occupy a navigable ocean position, and must remain at depth zero.

The SHIP class participates in the Referee design pattern (see Section 5.5) that ensures that a ship cannot modify its own state without the knowledge and consent of the REFEREE class.

5.3.4 The SUBMARINE Class

The SUBMARINE class is a subclass of the SHIP class. A submarine is allowed to move underwater, as well as on the surface. A submarine moves in straight line until it reaches a non-navigable ocean position, or it is fired upon. It reacts to non-navigable

obstacles by randomly changing its heading, and when fired upon it will perform evasive manoeuvres and may counterattack.

Section 3.4.3 has a more detailed discussion of a submarine's behaviour.

5.3.5 The GAME Class

The GAME class is responsible for enforcing the proper sequence of play in the game.

A game is divided into rounds. In every round a submarine must be sunken. The player who sinks a submarine gets a promotion. When a player reaches the winning rank, he wins the game.

During a round, all the ships takes turns to play, starting with ship number one and ending with ship number four. During a turn, a player may move once and open fire once, however those actions may be performed in any order.

The game class is not responsible for checking that a players actions are legal; that is the task of the REFEREE class.

The GAME class controls the interface class. In coordination with the SHIP and REFEREE classes, it tells the interface what should be written on the screen and which controls should be disabled. The GAME class acts as a validation filter by preventing the players from entering commands that, after consultation of the SHIP and REFEREE classes, are known beforehand to be forbidden. For example, if the submarine is outside of firing range, the firing order will be filtered out.

Note that the REFEREE class is fully capable of handling all illegalities, the filtering is done for efficiency reasons only.

5.3.6 The RANK Class and PROMOTIONPATH Class

For the basic game, RANK is just a wrapper for the string class and holds the name of the rank such as "commander" or "captain". It has been made a class to ease future enhancement to the game such as restricting the right to command a given type of ships to certain ranks.

The PROMOTIONPATH class is an ordered collection of ranks, from the lowest to the highest. The highest rank is the one a player must achieve in order to win the game.

5.3.7 The SCENARIO Class

The SCENARIO class encapsulates the war scenario that is being simulated in the game. A SCENARIO is composed of three parts: the initial position of the ships, the rules of engagement and the objectives.

5.3.7.1 The INITIALPOSITION Class

For the basic game, the INITIALPOSITION class contains the starting position of each ship at the beginning of a round.

In the future, it will contain the state of the ship at the beginning, such as the amount of fuel, ammunition, etc.

5.3.7.2 The RULESOFENGAGEMENTS Class

The RULESOFENGAGEMENTS class determines the circumstances under which one ship may open fire on another ship. The member function *IsValidTarget* decides if the first given ship is permitted to fire on the second given ship at the current point of the game.

5.3.7.3 The OBJECTIVES Class

The OBJECTIVES class records all the goals of all players in the game, both human players and the computer player for the submarine. One goal is that a ship must be sunk before a win can be claimed.

At the beginning of a round, the REFEREE takes a copy of the objectives. When a ship is sunk or abandoned, then it is removed from the objectives. If all surface ships are eliminated, the computer wins; and if the submarine is destroyed, then the round is over.

5.3.8 The REFEREE Class

The REFEREE class is responsible for enforcing all the rules of the game. Although it is assisted by the GAME class for reasons of efficiency, the REFEREE is the final arbiter and is fully capable of enforcing all rules by itself.

The duties of the REFEREE class include

- checking for collision with other ships,
- checking for a ship running aground,
- announcing the result of firing,
- declaring that a ship is damaged,
- removing destroyed ships from the game,
- determining which ship will play the next turn,
- announcing the end of the round,
- promoting the players,
- announcing the end of the game, and
- supervision of all ships and players in the game.

To perform its task, the class works closely with the INITIALPOSITION, RULESOFENGAGEMENTS, and OBJECTIVES classes. The referee trusts the GAME class for the enforcement of some rules and does not double check to ensure that the game class correctly perform its duty.

More detail can be found in Section 5.5.

5.4 Dynamic Behaviour

The behaviour of the system is illustrated by documenting the main use cases, the major collaborations between classes, and the dynamics of the GAME class which implements the use cases. Each of these is described in the following subsections.

The most significant collaboration is the referee design pattern, which is described in Section 5.5.

5.4.1 Use Cases

Here are some of the most typical use cases for the game. We describe the basic scenario for each use case.

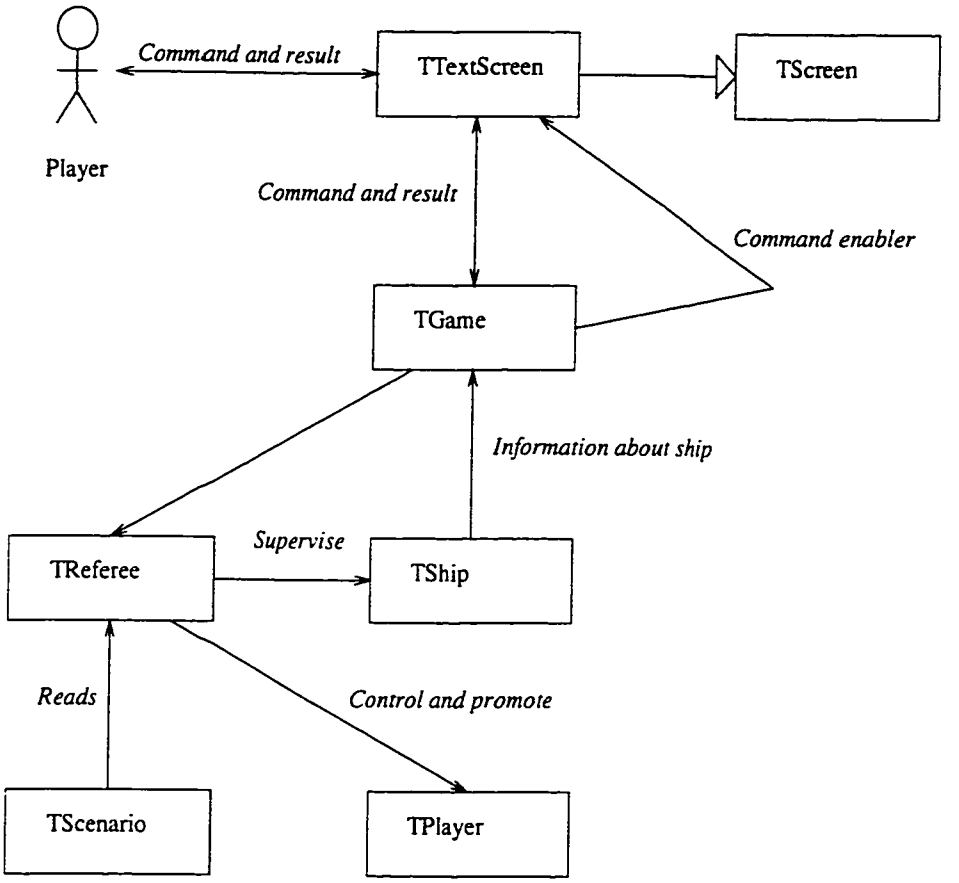

Figure 4: Class Diagram Showing Role of User Interface Classes

1. Normal use.

   Four players take turns, and play to the end of the game.

   (a) The game starts running.

   (b) A new round starts.

   (c) The players take turns making moves, until the target submarine is sunk.

   (d) The player who has sunken the submarine gets a promotion.

   (e) Repeat (b), (c), (d) until a player achieves the preset winning rank.

   (f) End of the game.

2. A player gives up and abandons the game.

   The remaining players may continue to play. Note that more than one player may abandon the game, but we assume at least one player does not abandon the game.

   (a) The game starts running.

   (b) A new round starts.

   (c) The remaining players take turns making moves, until the target submarine is sunk, or until a player abandons the game.

   (d) The player who has sunken the submarine gets a promotion, or the player who has abandoned the game is removed.

   (e) Repeat (b), (c), (d) until a player achieves the preset winning rank.

   (f) End of the game.

3. All players abandon the game.

   (a) The game starts running.

   (b) A new round starts.

   (c) The remaining players take turns making moves, until the target submarine is sunk, or until a player abandons the game.

   (d) The player who has sunken the submarine gets a promotion, or the player who has abandoned the game is removed.

(e) If all players have abandoned the game then the submarine wins, else repeat (b), (c), (d) until a player achieves the preset winning rank.

(f) End of the game.

5.4.2 Collaborations

The SCREEN class and the GAME class collaborate to keep the information displayed on the screen up to date. The GAME class sends a message to the SCREEN class every time that a piece of information needs to be displayed. It also sends a message when a user input is required: upon reception of the message the SCREEN class will query the player, and partially validate the input. The validation it performs is quite basic: the SCREEN class can tell if the user enters letters instead of digits. The SCREEN class will also disable certain interface functionality upon the request of the GAME class.

The GAME class and the SHIP class do not really collaborate: the GAME class merely reads some of the ship's attributes in order to pass them to the SCREEN class where they will be displayed on the screen.

The GAME class and the REFEREE class work closely together. Whenever the user's orders call for an action that does not fall within the limited competence of the GAME class, the GAME sends an appropriate message to the REFEREE which in turn replies with a message containing the effect of the last user action.

The REFEREE and the SHIP classes work so closely together that the entire Section 5.5 is devoted to their interaction, that we have called the *referee design pattern.*

5.4.3 Dynamics of the GAME Class

The dynamic behaviour for the game class is presented in Figures 5, 6 and 7. These state diagrams essentially expand the detail of the above use cases. Figure 5 is the overview state diagram for GAME, while Figure 6 expands the state for playing a round, and Figure 7 expands the state for playing a turn.

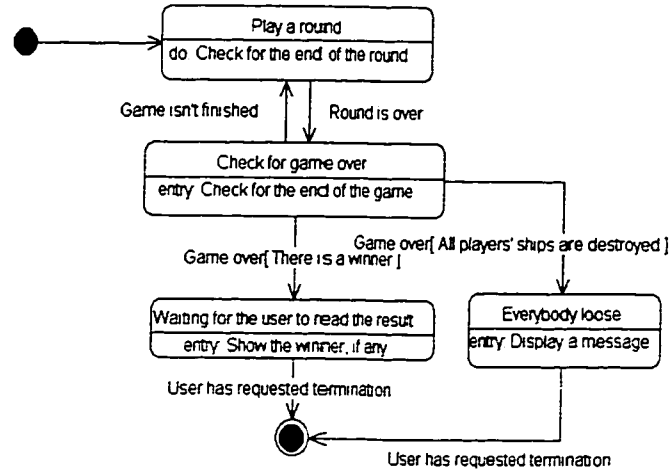

Figure 5: Overview State Diagram for GAME Class

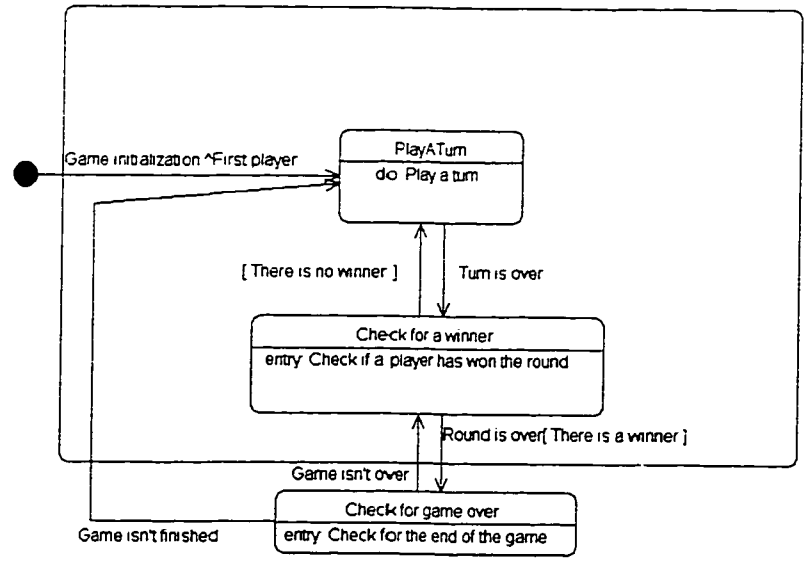

Figure 6: State Diagram for Play a Round

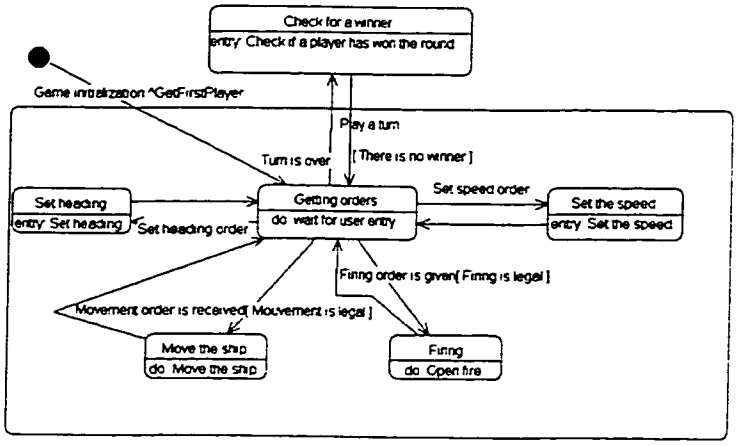

Figure 7: State Diagram for Play a Turn

5.5 The Referee Design Pattern

The most important design pattern used in this framework is the Referee Design Pattern. I have created this pattern as a small variation on the Mediator design pattern [12], and will describe it in the same manner.

5.5.1 *Name* : Referee — Object Behavioral

5.5.2 *Intent*

A mediator promotes loose coupling by keeping objects from referring to each other explicitly, and lets you vary their interaction independently. A referee goes further and ensures that the state of the interacting objects cannot be altered without the explicit consent of the referee.

5.5.3 *Motivation*

The motivation for the referee design pattern is to mediate and centralize connectivity and dependency amongst objects, and to authorize *every change in the state* of the collaborating objects. We will name the collaborating objects the *players*.

For example, in a simulation game depicting combat between surface ships, submarine and airplanes, the rules of the game end up widely distributed among all the classes. Any modification to the rules of the game is hard to implement because

each object knows only a small part of the rules: a plane knows how to fly, a ship how to navigate, a missile how to go from its launcher to its target. Moreover, there is no guarantee that anyone extending the game will not commit a serious error by breaching, involuntarily, some fundamental rules of the game, such as allowing ships to fly or to cross islands: the rules should be put in a unique place into a referee class. Furthermore, all the instances that are subject to the control of the referee must never alter their state without the explicit consent of the referee, otherwise it would be too easy to circumvent the rules. This latter part must be "compiler enforceable", we cannot count on the vigilance of a human programmer for that. Additionally, as in real life, the referee must be able to *impose penalties* on the instance of the player that he is monitoring. This implies that the referee must be able to alter the state of the player *without any restrictions* other than maintaining the integrity constraint of the instance. If appropriate, the referee could have the right to destroy an instance of a player, the real life equivalent being the expulsion of a player by the referee. In this context, *infractions* refer to some attempts made by a player to circumvent the rules of the game such as running aground, attempting to drive a ship through an island, colliding with another vehicle, attempting to drive a car at the speed of the light, attempting to move out of the playing area, attempting to fire without ammunition, etc. Because a referee has full access to the state of its player, *the most extreme care* must be taken when coding it in order to avoid *misuse* of this authority by the referee. Special care must be taken as to *under which circumstances* the referee should intervene in the game.

For example, it would be an *severe blunder* for the referee of a chess game to stop the game by claiming a draw by the 50 moves rules as this rule clearly states that the game is drawn "*when a player having the move demonstrates that at least fifty consecutive moves have been made*". So it is the player that must claim the draw, not the referee. This situation is not unique to chess, it might occur in any game for which the rules require an action by a player before the referee could legally intervene.

It should be emphasized that it is *almost impossible* to correct a referee error after the fact: after being *illegally* informed by the referee of a situation requiring that a claim to be made by a player, the player could easily and *legally* make this claim and the whole issue would go in appeal or the game would continue under protest. The converse is not better: the referee must actively intervene each time the rules require its intervention. A chess referee that would let two beginning players continue to play

despite the fact that one of the player has mated the other without realizing it would also be committing a *severe blunder*, the only difference being that it can be easily corrected at any time before the pairings of the next round.

5.5.4 *Applicability*

- Use the referee pattern when coding games in which the players must be constantly monitored for possible rule violation. In some games it might be possible that even the player played by the computer might attempt to commit infractions and in this case, the referee must handle the situation in the same manner as if it was a human player.

- The following two cases adapted from the mediator pattern are still relevant.

  1. A set of objects communicates in a well defined but complex way. The resulting interdependencies are unstructured and difficult to understand.
  2. The rules of the games (or business rules) that are distributed between several classes should be customizable without a lot of subclassing.

5.5.5 *Structure*

Figure 8 presents a UML object diagram for the design pattern.

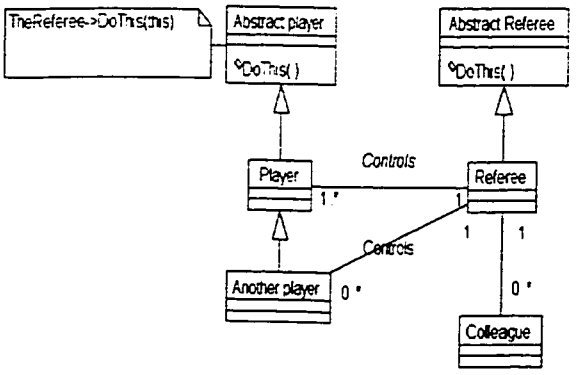

Figure 8: Object Diagram for Referee Design Pattern

5.5.6 *Participants*

- **Abstract Referee**
  - Define an interface that the players will use to communicate with the referee.
- **Concrete Referee**
  - Implement cooperative behavior between the players.
  - Know, control and penalize its players.
  - Ensure that its players are in a state of integrity and that they are following all rules.
- **Player classes**
  - Each player knows the referee.
  - Each player communicates with the referee when it would have performed any action that could result in a modification to its state or the state of another player.
  - Players are free to perform any action that does not modify their state without the knowledge nor the authorization of he referee.
- **Colleague classes**
  - Send and receive messages from the referee but are not under the control of the referee.
  - They are not important as far as the pattern is concerned and are mentioned only for sake of completeness.

5.5.7 *Collaborations*

The players send and receive requests from a Referee object. The Referee enforces the rules and implements the cooperative behavior by routing requests to the appropriate players or to other objects.

5.5.8 *Consequences*

Being a close relative of the Mediator pattern, the Referee pattern shares many consequences with it. The Referee Pattern has the following benefits and drawbacks:

1. It *limits subclassing.* A Referee localizes behavior that would otherwise be distributed among several objects. Minor changes to this behavior requires changing the Referee only: Players and Colleagues can be reused *as is.* Major changes might require new data members in the Players and/or the extension of the interface between the Player and the Referee. The Referee pattern ensures that the changes in the Player will be minimal and that the bulk of the work will be done in the Referee class.

2. *It decouples colleagues.* A Referee promotes loose coupling between colleagues. You can vary colleagues and the Referee class independently.

3. *It simplifies object protocols.* A Referee replaces many-to-many interactions with one-to-many interactions between the Referee, its Player and its Colleagues. One-to-many relationship are easier to understand, maintain and extend.

4. *It abstracts how objects cooperate.* Making refereeing an independent concept and encapsulating it in an object lets you separate how objects interact from their individual behavior. This can help clarify how objects interact in a system.

5. *It centralizes control.* The referee trades complexity in interaction for complexity in the Referee. Because a referee encapsulates protocols, it is usually more complex than any of its players and colleagues. This can make the referee itself a monolith that is hard to maintain. In order to lessen this situation, the referee should rely on aggregation and delegation to minimize its own size.

5.5.9 *Code example*

The implementation is a bit more tricky than for a mediator. We must find a way to *prevent* unauthorized access to the players' data members. This can be achieved in the following way:

1. The player class must *have no non-constant public member function* since such a function could be used to alter the state of the player behind the back of the referee.

2. All the member functions whose role is to *set* the value of the data members (the *SetVariableName()*) must be declared `protected`. This will limit access to the aforementioned functions to the class itself, its subclasses and its friends.

3. It is possible to implement this class without using a abstract player class if we want. All the behavior being transfered to the REFEREE class, there should be no virtual function (other then the destructor) in the abstract ship class. If there is no virtual class in a class, there is no need for an abstract superclass.

4. Make the abstract referee class a *friend* of the abstract player class. This will grant the referee the *exclusive right* to call the functions mentioned in the previous item.

5. Make sure that all the player classes, both the abstract and the concrete ones, *have no other friends* because those other friends *could corrupt the state of the player behind the back of the referee.*

6. The referee should always use the access functions when accessing its players, it should never directly access the internal details of its players. Other *Design Patterns* could be used in order to *hide* the internal structure of the player from the referee. Direct access to the internal structure of the player would result in an *unacceptably high level of coupling* between the referee and the player. This pattern has been created to reduce coupling, not to increase it.

7. Each time that a player of class T would have called a non-constant member function, say f(a,b,c), it must be replaced by the constant member f(a, b, c). The parameters a, b and c are passed in the same manner (by value or by reference) as in the original function. Inside function f(a,b,c) the whole body is replaced by Referee.f(MySelf,a,b,c). Again a,b and c are passed in the same manner as in the original function and MySelf, the instance of player that is calling f must always be passed by reference. In C++, a *const_cast* must be used to remove the constantness of the object in a constant member function.

```
void Player::f(int a, int&B, char c) const
{
     Referee->f(*const_cast<Player * >(this),a,b,c)
}
```

After doing this step, all the *behavior* is transferred to the referee; the player is just holding its data member.

8. A world of warning should be included directly into the source code of each of the player classes. To implement this pattern, we need to write a *constant member function* that will indeed *modify the state of an object.* This is not a contradiction, as any competent C++ programmer should know that const_cast is part of the language and that it has been included with the intent that it should be used when appropriate.

```
class AbstractShip
{
     friend Referee;

     protected:
          SetPosition(Position NewPosition)
          SetHeading(unsigned NewHeading) ...
     private:
          //all the data member
}

class WarShip : public AbstractShip
{
     public:
          WarShip(.....);
          virtual void MoveOnTheOcean(unsigned NewSpeed,
                                      unsigned NewHeading) const
     private:
          static Referee *TheReferee;
};
```

```
void WarShip::MoveOnTheOcean MoveOnTheOcean(unsigned NewSpeed,
                                            unsigned NewHeading) const
{
     TheReferee->MoveOnTheOcean(*const_cast<Ship *>(this),
                                NewSpeed, NewHeading);
}

TReferee::MoveOnTheOcean(AbstractShip *Aship, unsigned NewSpeed,
                         unsigned NewHeading)
{
     Aship->SetPosition(NewPosition); //protected member
}
```

9. Known uses

   This is an *original pattern*, for now it has been used only in this thesis.

Chapter 6

Conclusion

In this thesis we describe how to document an object-oriented application framework to encourage its reuse. In order to achieve this goal, we created a simple framework to document. The chapters of this thesis provide a framework overview, a cookbook of recipes. and a design overview incorporating a new design pattern. In the future, a set of graded applications should be built using the framework and experiments to verify the effectiveness of the documentation should be performed.

6.1 Guidelines for Framework Documentation

We summarize the past experience reported in the literature into guidelines on how to document a framework to assist application developers. The main point is to remember the audience: application developers, who may be somewhat inexperienced as developers, or in object-oriented technology, and may be somewhat ignorant of the application domain. The next point is to accept that it requires effort to create the documentation: it will not be a free by-product of the development of the framework.

All that application developer requires is an overview, examples, and recipes.

First, the application developers will need a context for the framework, so an *overview of the framework* should be prepared, both as a live presentation and as the first recipe in the cookbook.

Second, a *set of example applications* that have been specifically designed as documentation tools is required. The examples should be graded from simple through to advanced, and should incrementally introduce one hotspot at a time. A hotspot that

is very flexible may need several examples to illustrate its range of variability, from straightforward customization through to elaborate customization with all the bells and whistles.

One of the simpler example applications should be used in the overview presentation. The cookbook recipes will use sample source code from the example applications.

Third, a *cookbook* of recipes should be written, and organized along the lines of Johnson's pattern language. The recipes should use the example applications to make their discussion concrete. There will be cross-references between recipes, and between recipes and source code. There may also be cross-references to any other available documentation (such as a reference manual, contracts, or design patterns). A good cookbook can use just pen and paper, however, a hypertext browser will help navigate cross-references [23].

The guidelines emphasize

- prescriptive ("how-to") information — since this is what application developers need;
- concrete examples — to counter the abstractness of a framework design;
- graded, or spiral, organization of information — to minimize the amount of information needed, and to focus on the task at hand.

Access to more information, such as contracts, design patterns, or architecture, might also be available for consultation at rare times. Application developers should not need to regularly consult this information in order to do their job, but, on occasion, it might comfort the developer by dispelling the mystique of the inner workings of the framework, or by clarifying some detail through additional rigor.

6.2 Lessons Learnt from Case Study

The feedback from the documentation effort to the design, and its consequent improvement in viability, simplicity, and clarity were evident even as we wrote the documentation. So just creating end-user documentation had a positive effect on design quality.

In the final writing and review of the simplicity and clarity of the documentation (that is, the previous chapters), again, it is evident that convoluted documentation

often means a convoluted design (and not just a confused writer) where concepts and responsibilities are not clearly delineated.

6.3 Conclusion

It requires a large effort to understand any new software system, and application developers using a framework for the first time find the framework particularly difficult to understand. If documentation is to alleviate this situation it must be targeted to the needs of the application developer: how does the developer customize one or more hotspots of the framework in order to create an application.

Simple, easy-to-use, effective documentation may be viewed as the acid-test for the "goodness" of a framework design. It is hard to imagine that one could possibly create good documentation when there is a poor design for the framework. So good design is a necessary, but not sufficient, prerequisite for good documentation.

Reusable designs result from evolution and iteration, so both framework developers and maintainers deal with the evolution of a design. An important problem is therefore: How to describe (and specify) the evolution of a design and the differences between two versions of the design of a framework.

In the future, a set of graded applications should be built using the framework and experiments to verify the effectiveness of the documentation should be performed.

Bibliography

[1] Ackermann, P. *Developing Object-Oriented Multimedia Software.* dpunkt Publishing, Heidelberg, 1996.

[2] Apple Computer, **Macapp 2.0 General Reference Manual**,

[3] Arango, G., Schoen, E., and Pettengill, R. A process for consolidating and reusing design knowledge. In *Proceedings of 15th International Conference on Software Engineering.* IEEE Computer Press, Los Alamitos, CA, 1993, pp. 231–242.

[4] Beck, K. and Johnson, R. Patterns generate architectures. In *Object-Oriented Programming.* M. Tokoro and R. Pareschi (eds), LNCS *821*, Springer-Verlag, Berlin. 1994, pp. 139–149.

[5] Roger P. Beck, Satish R. Desai, Doris R. Ryan, Ronald W. Tower, Dennis Q. Vroom, Linda Mayer Wood, *Architecture for large-scale reuse*,

[6] A. Birrer and T. Eggenschwiler, *Frameworks in the financial engineering domain: An experience report*, **ECOOP'93 — Object-Oriented Programming**, O.M. Nierstrasz (ed.), Springer-Verlag, Berlin, 1993, pp.21–35.

[7] B. Bruegge, T. Gottschalk, B. Luo, *A framework for dynamic program analyzers*, **OOPSLA'93**, pp.65–82.

[8] Buhr, R.J.A. and Casselman, R.S. Architectures with pictures. In Proceedings of OOPSLA'92. ACM/SIGPLAN, New York, 1992, pp. 466–483.

[9] Roy H. Campbell, Nayeem Islam, David Raila, Peter Madany, *Designing and implementing* CHOICES*: An object-oriented system in C++*, Communications ACM **36**, 9 (September 1993) 117–126.

[10] Cotter, S. with Potel, M. *Inside Taligent Technology.* Addison-Wesley, Reading, Mass., 1995.

[11] L. Peter Deutsch, *Reusability in the Smalltalk-80 programming system*, (pp. 72–76, ITT Proceedings of the Workshop on Reusability in Programming 1983) in **Tutorial: Software Reusability**, Peter Freeman (ed.), IEEE Computer Society Press, 1987, pp.91–95.

[12] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, Reading, Mass., 1994.

[13] Helm, R., Holland, I.M., and Gangopadhyay, D. Contracts: specifying behavioral compositions in object-oriented systems. In Proceedings of OOPSLA'90. ACM/SIGPLAN, New York, 1990, pp. 169–180.

[14] Holland, I.M. Specifying reusable components with contracts. In *ECOOP'92.* LCNS *615*, Springer-Verlag, Berlin, 1992, pp. 287–308.

[15] Jacobson, I., Christorson, M., Jonsson, P. and Övergaard, G. *Object-Oriented Software Engineering: A Use Case Driven Approach.* Addison-Wesley, Reading, Mass., 1992.

[16] Johnson, R. Documenting frameworks using patterns. In Proceedings of OOPSLA'92. ACM/SIGPLAN, New York, 1992, pp. 63–76.

[17] R.E. Johnson and B. Foote, *Designing reusable classes*, Journal of Object-Oriented Programming **1** (1988) 22–35.

[18] R.E. Johnson, C. McConnell, J.M. Lake, *The RTL system: A framework for code optimization*, **Code Generation — Concepts, Tools, techniques**, R. Giegerich and S.L. Graham (eds), Springer-Verlag, London, 1991, pp.255–274.

[19] Lajoie, R. and Keller, R.K. Design and reuse in object-oriented frameworks: Patterns, contracts, and motifs in concert. In *Object-Oriented Technology for Database and Software Systems.* V.S. Alagar and R. Missaoui (eds). World Scientific Publishing, Singapore, 1995, pp. 295–312.

[20] Lewis, T. *Object-Oriented Application Frameworks.* Manning Publications, Greenwich, CT, 1995.

[21] Linn, M.C. and Clancy, M.J. The case for case studies in programming problems. *Comm. of ACM 35*, 3 (Mar. 1992) 121–132.

[22] M.A. Linton, J.M. Vlissides, P.R. Calder, *Composing user interfaces with Interviews*, IEEE Computer **22**, 2 (February 1989) 8–22.

[23] Meusel, M., Czarnecki, K. and Köpf, W. A model for structuring user documentation of object-oriented frameworks using patterns and hypertext. Proceedings of *ECOOP'97*. LCNS *1241*, Springer-Verlag, Berlin, 1997, pp. 496–510.

[24] Meyer, B. Applying design by contract. *IEEE Computer 25*, 10 (Oct. 1992) 40–51.

[25] Parker Brothers, *Code Name: Sector*, game.

[26] Pree, W. *Design Patterns for Object-Oriented Software Development.* Addison-Wesley, Reading, Mass., 1995.

[27] Rational Corporation, *Unified Modeling Language*, Home page: `http://www.rational.com/uml/index.html`

[28] Schappert, A., Sommerlad, P. and Pree, W. Automated framework development. Symposium on Software Reusability (SSR'95), *ACM Software Engineering Notes* (Aug. 1995) 123–127.

[29] Sparks, S., Benner, K., and Faris, C. Managing object-oriented framework reuse. *IEEE Computer 29*, 9 (Sep. 1996) 52–61. 1990. AT&T Technical Journal **71**, 6 (1992) 34–45.

[30] Stroustrup, B. *The C++ Programming Language*, 3rd edition, Addison-Wesley, Reading, Mass., 1997.

[31] Taligent, Inc., *Building object-oriented frameworks*, A Taligent White Paper, 1994.

[32] A. Weinand, E. Gamma, R. Marty, *Design and implementation of ET++, a seamless object-oriented application framework*, Structured Programming **10**, 2 (1989) 63–87.

Appendix A

Source Code

Source code of the Framework and of the sample application.

```
#include <conio.h>
#include "rank.h"
#include "ocean.h"
#include "BasicOceRep.h"
#include "InitialPosition.h"
#include "objective.h"
#include "Scenario.h"
#include "PromotionPath.h"
#include "Ship.h"
#include "Referee.h"
#include "coordinate.h"
#include "Game.h"

using namespace SUB_Framework;

int main()
{

   try
   {

      //The game object is composed of many parts that must be constructed

      //First the ocean representation and the ocean
      randomize();

      static TBasicGameOceanRepresentation Ocre(2);
      static TOcean TheOcean(50,24,Ocre);

      //Set the static pointer to the ocean that all ships share to point to
      //TheOcean

      TShip::SetTheOcean(&TheOcean);
      TReferee::SetTheOcean(&TheOcean);

      //Now we build what is required to construct a scenario

      TRulesOfEngagement RulesOfEngagement;

      //Now we create the Players and their ships

      //The player of the game, numerous pointers and references to those
      //players will be active during the game. It is imperative that they
      //do not be detroyed before the end of the program. so they are
      //declared static.

      static  TPlayer Player1("Pierre  "),
                      Player2("Jean    "),
                      Player3("Martine "),
                      Player4("Annie   ");

      TShip::RegisterANewShipType(TShip::pt_boat,9,180,9,2);
      TShip::RegisterANewShipType(TShip::destroyer,8,135,4,2);
      TShip::RegisterANewShipType(TShip::cruiser,6,90,3,2);
      TShip::RegisterANewShipType(TShip::battleship,4,45,2,2);

      //The ships of the game, numerous pointers and references to those
      //ships will be active during the game.

      static TShip Ship1 (TShip::pt_boat,&Player1,"USS Entreprise",0,9),
                   Ship2 (TShip::pt_boat,&Player2,"USS Alabama   ",0,4),
```

```
                    Ship3 (TShip::pt_boat,&Player3,"USS Lexinton  ",4,0),
                    Ship4 (TShip::pt_boat,&Player4,"USS Lenin     ",9,0);

      //The scenario requires a list of all the players' ships in the game

      vector<TShip*> PlayersShipsInTheGame, EnnemyShipsInTheGame;

      PlayersShipsInTheGame.push_back(&Ship1);
      PlayersShipsInTheGame.push_back(&Ship2);
      PlayersShipsInTheGame.push_back(&Ship3);
      PlayersShipsInTheGame.push_back(&Ship4);

      //Now we create the target submarine;

      static TSubmarine TargetSubmarine(&TSubmarine::GetDefaultCommandingOfficer(
),"U-182");
      TargetSubmarine.RandomizePosition();
      TargetSubmarine.Practice(TCoordinate(2,9,1));
      EnnemyShipsInTheGame.push_back(&TargetSubmarine);

      //And we now create the list of objectives: an objective need not to
      //be an ennemy, it be be neutral even if this is not politically correct

      TObjectives Objectives(&TargetSubmarine);
      Objectives.AddAComputerObjective(&Ship1);
      Objectives.AddAComputerObjective(&Ship2);
      Objectives.AddAComputerObjective(&Ship3);
      Objectives.AddAComputerObjective(&Ship4);

      //We now build the scenario

      TScenario TheScenario(RulesOfEngagement,Objectives,
                            PlayersShipsInTheGame, EnnemyShipsInTheGame);

      // With the scenario, we can build the referee
      TReferee TheReferee(TheScenario);

      //We give all ships a pointer to the referee
      TShip::SetReferee(&TheReferee);

      // Now we define a class to handle all the IO

      TTextScreen TheScreen(TheOcean);

      // We create the promotion path that the player must follow
      // in order to win.

      TPromotionPath::AddANewRank(TRank("Lt-cmdr"));
      TPromotionPath::AddANewRank(TRank("Cmdr"));
      TPromotionPath::AddANewRank(TRank("Captain"));
      TPromotionPath::AddANewRank(TRank("Comodore"));


      //We create the game object

      TGame TheGame(TheOcean, TheReferee, TheScreen);

      TheGame.Play();
      clrscr();
   }
```

```
    catch (TSubError &AnError)
    {
       clrscr();
       cerr<<AnError<<endl;
    }

    catch (std::exception &AnError)
    {
       clrscr();
       cerr<<AnError.what()<<endl;

    }

    catch (...)
    {
       clrscr();
       cout<<"unknown exception"<<endl;
    }


    cout<<"Game over "<<endl;
    char c;
    cin>>c;
    return 0;
}
```

```
#ifndef TextScreen_H
#define TextScreen_H

#include "TheScreen.h"

namespace SUB_Framework
{

//*****************************************************************************
//*****************************************************************************

class TTextScreen : public TScreen
{
   public :

      //Constants do not need to be put in the private part to prevent
      //their modification

      static const unsigned ScreenPositionToWrite;
      static const unsigned ScreenPositionToWriteInfractionMessages;
      static const unsigned ScreenPositionToWriteAimingInfo;

      explicit TTextScreen(const TOcean &AnOcean);
      virtual void PrepareANewRound();
      virtual void DrawTheOcean() const;
      virtual void DrawShip(const TShip &AShip,
                            const TCoordinate &PreviousPosition)const ;


      virtual void DisplayShipInfo (const TShip &AShip);
      virtual void GetNewHeadingAndSpeed(unsigned &NewHeading, unsigned &NewSpeed
);
      virtual unsigned GetAimingDirection() const;
      virtual void GetAimingInfo();
      virtual TAimingInformation GetAimingInformation() const;
      virtual void ShowFutureSpeed();
      virtual void ShowFutureHeading();
      virtual void ShowInfractionMessage(const string &InfractionMessage);
      virtual void ShowInformationMessage(const string &InformationMessage);
      virtual void ClearInfractionWindow();
      virtual void ClearInformationWindow();
      virtual void ClearAimingInfoWindow();
      virtual void ShowAimingInformation();
      virtual bool GetConfirmation(const string &Message);
      virtual void PressAnyKeyToContinue(const string &Message);

      //virtual void ShowInstructions(const bool HasAlreadyFired);
      virtual TReferee::TValidOrders GetInstructions();

      //virtual void ShowInfractionMessage(const string &InfractionMessage)=0;

      virtual void EraseAShip(const TCoordinate &PositionOfTheShip);
      virtual ~TTextScreen();

   protected :
      virtual void GetAimingDirectionFromTheUser();
      virtual void GetAimingDepthFromTheUser();

   private :
```

```
//There is only one text output screen so copying and assiging it must be prohibi
ted.
//There will be no definition for the next two functions.

      static unsigned short NumberOfActiveInstances=0;
      TTextScreen(const TTextScreen &AnotherScreen);
      TTextScreen& operator = (const TTextScreen &AnotherScreen);
};

//***************************************************************************

inline void TTextScreen::ShowFutureSpeed()
{
   gotoxy(ScreenPositionToWrite,7);
   // clreol(); tempo
   cout<<"                    ";
   gotoxy(ScreenPositionToWrite,7);
   cout<<"New speed: "<<FutureSpeed<<endl;
}

//***************************************************************************

inline void TTextScreen::ShowFutureHeading()
{
   gotoxy(ScreenPositionToWrite,6);
   clreol();
   cout<<"New heading: "<<TShip::ConvertHeading(FutureHeading)<<endl;;
}

//***************************************************************************

inline unsigned TTextScreen::GetAimingDirection() const
{
   if (AimingDirection <= 360)
      return AimingDirection;
   else
      throw TInvalidBearing(AimingDirection);
}

//***************************************************************************

inline TAimingInformation TTextScreen::GetAimingInformation() const
{
   return TAimingInformation(AimingDirection,AimingDepth);
}

//***************************************************************************

inline void TTextScreen::ClearInformationWindow()
{
   ClearInfractionWindow();
}

//***************************************************************************
//***************************************************************************

} //End Namespace


#endif
```

```
#include <strstream>
#include <conio>
#include <limits.h>
#include <ctype.h> // for toupper

#include "TextScreen.h"
#include "ArrowKeysCode.h"

namespace  SUB_Framework
{

//*****************************************************************************
//*****************************************************************************

void TThereCanBeOnlyOneOutputScreen::What() const
{
   strstream ErrMessage(Message,sizeof (Message),ios::out);
   ErrMessage<<"Logic error: there can be only one output screen."<<ends;

}

//*****************************************************************************

//Static member definition

//unsigned TTextScreen::NumberOfActiveInstance=0;
const unsigned  TTextScreen::ScreenPositionToWrite=53;
const unsigned  TTextScreen::ScreenPositionToWriteInfractionMessages=15;
const unsigned  TTextScreen::ScreenPositionToWriteAimingInfo=9;
//*****************************************************************************



TTextScreen:: TTextScreen(const  TOcean &AnOcean) : TScreen(AnOcean)
                               //An invalid value so
                                                    //that we can know when
                                                    //the user has entered
                                                    //a value.
{
   if(++NumberOfActiveInstances > 1)
      throw  TThereCanBeOnlyOneOutputScreen();

   //PrepareANewRound();

};

//*****************************************************************************

void  TTextScreen::PrepareANewRound()
{
   AimingDirection=361;
   DrawTheOcean();
   ClearAimingInfoWindow();
   ClearInfractionWindow();
}

//*****************************************************************************

void  TTextScreen::DrawTheOcean() const
{
   clrscr();
```

```
    //char c;
    TCoordinate Position;
    unsigned CurrentLineNumber=1;

    for (unsigned i=0; i<= TheOcean.GetDimensionY(); i++)
    {
       gotoxy(1,CurrentLineNumber++);

       for (unsigned j=0; j<= TheOcean.GetDimensionX(); j++)
       {
          try
          {
             Position=TCoordinate(j,i);
          }
          catch (TSubError &AnError)
          {
             cerr<<"Severe error: attempt to draw an unexisting part of the ocean\
n";
             //cerr<<AnError<<endl;
             throw; //propagate the exception
          }
          //A function call isn't used to save time as this will be done
          //one time for each ocean square.

          cout<<TheOcean.GetWhatIsAtCoordinate(Position);

       } //End for j
    } //End for i

    const vector <TShip*> &ListOfAlliedShips=TShip::GetReferee()->GetListOfAlliedS
hips();
    vector <TShip*>::const_iterator EndOfTheList=ListOfAlliedShips.end();

    for (vector <TShip*>::const_iterator it=ListOfAlliedShips.begin(); it < EndOfT
heList; it++)
    {
       DrawShip(**it,(*it)->GetCurrentPosition());
    }
}

//*****************************************************************************

void TTextScreen::DrawShip(const TShip &AShip,
                           const TCoordinate &PreviousPosition)const
{

    gotoxy(PreviousPosition.GetLongitude()+1,TCoordinate::GetMaxLatitude()-
           PreviousPosition.GetLatitude()+1);

    //char c;

    cout<<TheOcean.GetWhatIsAtCoordinate(PreviousPosition);


    TCoordinate ShipPosition=AShip.GetCurrentPosition();
    gotoxy(ShipPosition.GetLongitude()+1,TCoordinate::GetMaxLatitude()-ShipPositio
n.GetLatitude()+1);
    cout<<AShip.GetID();
```

```
}

//*****************************************************************************

void TTextScreen::DisplayShipInfo (const TShip &AShip)
{
  /*  for (int i=0; i<=4; i++)
   {
      gotoxy(ScreenPositionToWrite,i);
      clreol();
   }*/
   gotoxy(ScreenPositionToWrite,1);
   cout<<AShip; //For text mode output this is OK
   FutureSpeed=AShip.GetCurrentSpeed();
   FutureHeading=AShip.GetCurrentHeading();
   MaximumSpeed=AShip.GetMaxSpeed();

   //gotoxy(ScreenPositionToWrite,6);
   //cout<<"New heading: "<<AShip.ConvertHeading()<<endl;
   ShowFutureHeading();

   //gotoxy(ScreenPositionToWrite,7);
   //cout<<"New speed: "<<FutureSpeed<<endl;
   ShowFutureSpeed();

}

//*****************************************************************************

void TTextScreen::GetNewHeadingAndSpeed(unsigned &NewHeading, unsigned &NewSpeed)
{
   //There is nothing to do as the new course and speed are already set.
   //For non text mode operation, this is where the value from the GUI
   //controls should be transferred into FutureHeading and FutureSpeed.

   NewHeading=FutureHeading;
   NewSpeed=FutureSpeed;
}

//*****************************************************************************

TReferee::TValidOrders TTextScreen::GetInstructions()
{
   int c;
   //cin.ignore(INT_MAX,'\n'); //Get rid of any charaters already in the buffer
   //cin.clear();
   //cin.get(C);

   bool ValidOrder;

   do
   {
      ValidOrder=true;
      c= getch();

      switch (toupper(c))
      {
         case '\0': //Extended character
         {
            int CharacterCode;
            CharacterCode=getch();
```

```
            switch (CharacterCode)
            {
               case UpArrow:
               {
                  if (FutureSpeed < MaximumSpeed)
                  {
                     FutureSpeed++;
                     ShowFutureSpeed();
                  }
                     ValidOrder=false; //Data entry is not an order
                  break;
               }
               case LeftArrow:
               {
                  FutureHeading=(FutureHeading+45)%360;
                  ShowFutureHeading();
                  ValidOrder=false;
                  break;
               }

               case DownArrow:
               {
                  if (FutureSpeed > 0 )
                  {
                     FutureSpeed--;
                     ShowFutureSpeed();
                  }
                  ValidOrder=false;
                  break;
               }
               case RightArrow:
               {
                  if (FutureHeading == 0) //Putting negative value into an
                                          //unsigned variable is an undefined
                                          //behavior
                     FutureHeading=315;
                  else
                     FutureHeading-=45;

                  ShowFutureHeading();

                  //break;  No break because we must set ValidOrder to false
               }
               default :

                  ValidOrder=false;
            }
            break;
         }
         case 'M':
            return TReferee::set_course_and_speed;

         case 'F':
         {

            GetAimingInfo();
            if (AimingDirection <=360)
               return TReferee::open_fire;
            else
               ValidOrder=false;
```

```
            break;
         }
         case 'Q':
            return TReferee::end_of_the_turn;

         case 'A':
            return TReferee::abandon_ship;

         default :
            ValidOrder=false;
      }
   }
   while (!ValidOrder);

   //cout<<C<<endl;
   return TReferee::end_of_the_turn; //Tempo

}

//*****************************************************************************

void TTextScreen::GetAimingInfo()
{



   //char c;
   //cin>>c;
   //window(1,1,80,25);
   const TShip * const ActiveShip=TShip::GetReferee()->GetActiveShip();

   unsigned TargetRange=const_cast <TShip*>(ActiveShip)->GetTargetRange();
   if (ActiveShip->GetCanOpenFire())

   {
      /*if (TargetRange == 0 )
      {
         AimingDirection=0;//We are directly above the target.
                           //Aiming direction is irrelevant.
         AimingDepth=0;

      } *
      else  */
      {

         textbackground(BLACK);
         textcolor(WHITE);
         textattr(7);
         GetAimingDirectionFromTheUser();
      }
      ShowAimingInformation();
   }
   else
   {

      unsigned WeaponRange=ActiveShip->GetWeaponRange();
      if (TargetRange > WeaponRange)
         ShowInfractionMessage("Target is out of range");
      else
      {
         ShowInfractionMessage("You cannot open fire now");
```

```
            AimingDirection=1000; //Force an invalid command to avoid ending the loo
p
         }
      }
      //gettextinfo(&NewTextInfo);
}

//*****************************************************************************

void TTextScreen::GetAimingDirectionFromTheUser()
{
   int c;
   AimingDirection=0;
   AimingDepth=1;
   ShowAimingInformation();


   do
   {
      c=getch();
      if (c=='\0')
      {
         c=getch();
         switch (c)
         {
            case UpArrow:
            {
               if (AimingDepth != TCoordinate::GetMaxDepth())
                  AimingDepth++;
               else
                  AimingDepth=1;
               ShowAimingInformation();
               break;
            }

            case DownArrow:
            {
               if (AimingDepth !=1)
                  AimingDepth--;
               else
                  AimingDepth=1;
               ShowAimingInformation();
               break;
            }

         }
      }
   } while (toupper(c) != 'D');

   do
   {
      c=getch();
      if (c=='\0')
      {
         c=getch();
         switch (c)
         {
            case LeftArrow:
            {
               if (AimingDirection != 0 )
```

```
                    AimingDirection-=45;
                 else
                    AimingDirection=315;

                 ShowAimingInformation();

                 break;
              }
              case RightArrow:
              {
                 AimingDirection=(AimingDirection+45)%360;
                 ShowAimingInformation();
              }
           }//end switch

        }// end if
     } // end do while loop
     while (toupper(c) !='F');
  }

  //*****************************************************************************

  void TTextScreen::GetAimingDepthFromTheUser()

  {
     int c;

     do
     {
        c=getch();
        if (c=='\0')
        {
           c=getch();
           switch (c)
           {
              case UpArrow:
              {
                 if (AimingDepth != 3) //Tempo, should get the maximum depth

                    AimingDepth++;
                 else
                    AimingDepth=0;;

                 ShowAimingInformation();

                 break;
              }
              case DownArrow:
              {
                 AimingDepth=(AimingDepth !=0)?AimingDepth-1:3;
              }
           }//end switch

        }// end if
     }
     while (toupper(c) != 'F');
  }

  //*****************************************************************************

  void TTextScreen::ShowAimingInformation()
```

```
{
   window(ScreenPositionToWrite,ScreenPositionToWriteAimingInfo,79,ScreenPosition
ToWriteAimingInfo+4);
   textcolor(BLACK);
   textbackground(WHITE);
   //clrscr();

   gotoxy(ScreenPositionToWrite,ScreenPositionToWriteAimingInfo);

   //cout<<"Aiming direction: ";
   gotoxy(19,wherey());
   clreol();
   cout<<TShip::ConvertHeading(AimingDirection)<<endl;
   gotoxy(14,wherey());
   clreol();
   cout<<AimingDepth<<endl;
   window(1,1,80,25);
   textcolor(WHITE);
   textbackground(BLACK);
}

//*****************************************************************************

void TTextScreen::ShowInfractionMessage(const string& InfractionMessage)
{
   gotoxy(ScreenPositionToWrite,ScreenPositionToWriteInfractionMessages);
   window(ScreenPositionToWrite,ScreenPositionToWriteInfractionMessages,79,24);
   //struct text_info TextInfo,NewTextInfo;
   //gettextinfo(&TextInfo);
   textcolor(BLACK);
   textbackground(RED);
   clrscr();
   //cout<<InfractionMessage<<"This is a very long test";
   cprintf(InfractionMessage.c_str());
   //cout.flush();
   window(1,1,80,25);
   textbackground(BLACK);
   textcolor(WHITE);
   textattr(7);
   //gettextinfo(&NewTextInfo);

}

//*****************************************************************************

void TTextScreen::ShowInformationMessage(const string& InfractionMessage)
{
   gotoxy(ScreenPositionToWrite,ScreenPositionToWriteInfractionMessages);
   window(ScreenPositionToWrite,ScreenPositionToWriteInfractionMessages,79,24);
   //struct text_info TextInfo,NewTextInfo;
   //gettextinfo(&TextInfo);
   textcolor(BLACK);
   textbackground(BLUE);
   clrscr();
   //cout<<InfractionMessage<<"This is a very long test";
   cprintf(InfractionMessage.c_str());
   //cout.flush();
   window(1,1,80,25);
   textbackground(BLACK);
   textcolor(WHITE);
   textattr(7);
```

```
    //gettextinfo(&NewTextInfo);

}

//*****************************************************************************

void TTextScreen::ClearInfractionWindow()
{
    window(ScreenPositionToWrite,ScreenPositionToWriteInfractionMessages,79,24);
    //struct text_info TextInfo,NewTextInfo;
    //gettextinfo(&TextInfo);
    textcolor(BLACK);
    textbackground(GREEN);
    clrscr();
    window(1,1,80,25);
    //textbackground(BLACK);
    //textcolor(WHITE);
    textattr(7);

};

//*****************************************************************************

void TTextScreen::ClearAimingInfoWindow()
{
   window(ScreenPositionToWrite,ScreenPositionToWriteAimingInfo,79,ScreenPosition
ToWriteAimingInfo+4);
   textcolor(BLACK);
   textbackground(WHITE);
   clrscr();

   gotoxy(ScreenPositionToWrite,ScreenPositionToWriteAimingInfo);

   cprintf("Aiming direction: \n");
   window(1,1,80,25);
   gotoxy(ScreenPositionToWrite,ScreenPositionToWriteAimingInfo+1);
   cprintf("Aiming depth: ");


}

//*****************************************************************************

bool TTextScreen::GetConfirmation(const string &Message)
{
   gotoxy(ScreenPositionToWrite,ScreenPositionToWriteInfractionMessages);
   window(ScreenPositionToWrite,ScreenPositionToWriteInfractionMessages,79,24);
   //struct text_info TextInfo,NewTextInfo;
   //gettextinfo(&TextInfo);
   textcolor(BLACK);
   textbackground(MAGENTA);
   clrscr();
   cprintf(Message.c_str());
   gotoxy(1,wherey()+2);
      clreol();
      cprintf("Please confirm by typing Y ");
      int c;
      c=getch();
      c=toupper(c);
```

```
    ClearInformationWindow(); //tempo
    window(1,1,80,25);
    textbackground(BLACK);
    textcolor(WHITE);
    textattr(7);

    return c=='Y';
}

//***************************************************************************

void TTextScreen::PressAnyKeyToContinue(const string &Message)
{
   gotoxy(ScreenPositionToWrite,ScreenPositionToWriteInfractionMessages);
   window(ScreenPositionToWrite,ScreenPositionToWriteInfractionMessages,79,24);
   textcolor(BLACK);
   textbackground(YELLOW);
   clrscr();
   cprintf(Message.c_str());
   gotoxy(1,wherey()+2);
   clreol();
   cprintf("Press any key to continue.");
   int c;
   c=getch(); //Wait for a keypress

   ClearInformationWindow();
   window(1,1,80,25);
   textbackground(BLACK);
   textcolor(WHITE);
   textattr(7);

}

//***************************************************************************

void TTextScreen::EraseAShip(const TCoordinate &PositionOfTheShip)
{

   gotoxy(PositionOfTheShip.GetLongitude()+1,TCoordinate::GetMaxLatitude()-Positi
onOfTheShip.GetLatitude()+1);
   cout<<TheOcean.GetWhatIsAtCoordinate(PositionOfTheShip);

}

//***************************************************************************

TTextScreen::~TTextScreen()
{
   NumberOfActiveInstances--;
}

//***************************************************************************
//***************************************************************************

} // End namespace
```

```
#ifndef SUNKINFO_H
#define SUNKINFO_H

#include "Ship.h"

namespace  SUB_Framework
{

class  TSunkInfo
{
   public :
      TSunkInfo(const  TPlayer * APlayer=NULL, const  TShip * AShip=NULL);
      bool  operator  == (const  TSunkInfo &SI);
   private :
      const  TPlayer * ThePlayerWhoHasSunkTheShip;
      const  TShip * TheSunkShip;
};

bool  inline  TSunkInfo::operator  ==(const  TSunkInfo &SI)
{
   return    *ThePlayerWhoHasSunkTheShip == *SI.ThePlayerWhoHasSunkTheShip &&
             *TheSunkShip == *SI.TheSunkShip;
}

}//End namespace
#endif
```

```
#include "SunkInfo.h"

namespace  SUB_Framework
{

TSunkInfo::TSunkInfo(const TPlayer *APlayer, const TShip *AShip):
                     ThePlayerWhoHasSunkTheShip(APlayer),
                     TheSunkShip(AShip)
{
}


}
```

```
#ifndef SCENARIO_H

#include "InitialPosition.h"
#include "RulesOfEngagement.h"
#include "Objective.h"
#include "Ship.h"



namespace  SUB_Framework
{

//*****************************************************************************
//*****************************************************************************

   class  TThereMustBeAtLeastOneShip : public  TInitialisationError
   {
      public :

         TThereMustBeAtLeastOneShip(const TShip::TShipAllignment AnAllignment);
         virtual  void What() const ;
      private :
         TShip::TShipAllignment Allignment;
   };

//*****************************************************************************
//*****************************************************************************

   class  TReferee;
//*****************************************************************************
//*****************************************************************************

   class  TScenario
   {
      friend  TReferee;
      public :
         TScenario(const TRulesOfEngagement &Rules,
                   const TObjectives &AListOfObjectives,
                   vector<TShip*> &AlliedShips,
                   vector<TShip*> &EnnemyShips);

         const TInitialPositions& GetInitialPositions() const ;
         const TRulesOfEngagement& GetRulesOfEngagements() const ;
         const TObjectives& GetObjectives()const ;

      protected :

         vector<TShip*>& GetListOfAlliedShips() const ;
         vector<TShip*>& GetListOfEnemyShips () const ;

      private :
         TInitialPositions InitialPositions;   //Not really required because ever
ything is static in TInitial position //tempo
         const TRulesOfEngagement &RulesOfEngagement;
         const TObjectives &Objectives;
         vector<TShip*> &ListOfAlliedShips,
                        &ListOfEnemyShips;


   };
```

```
//*****************************************************************************

   const TInitialPositions& TScenario::GetInitialPositions() const
   {
      return InitialPositions;
   }

//*****************************************************************************

   const TRulesOfEngagement& TScenario::GetRulesOfEngagements() const
   {
      return RulesOfEngagement;
   }

//*****************************************************************************

   const TObjectives& TScenario::GetObjectives() const
   {
      return Objectives;
   }

//*****************************************************************************

   vector<TShip*>& TScenario::GetListOfAlliedShips() const
   {
      return ListOfAlliedShips;
   }

//*****************************************************************************

   vector<TShip*>& TScenario::GetListOfEnemyShips() const
   {
      return ListOfEnemyShips;
   }

//*****************************************************************************
//*****************************************************************************

} //End namespace
#define SCENARIO_H
#endif
```

```
#include <strstream>
#include "Scenario.h"


namespace  SUB_Framework
{

//*****************************************************************************
//*****************************************************************************

TThereMustBeAtLeastOneShip::TThereMustBeAtLeastOneShip(const  TShip::TShipAllignme
nt AnAllignment):
   Allignment(AnAllignment)
{
}

//*****************************************************************************
//*****************************************************************************

void TThereMustBeAtLeastOneShip::What() const
{
   strstream ErrMessage(Message,sizeof (Message),ios::out);
   ErrMessage<<"Initialisation error: at least one ";
   if (Allignment == TShip::allied)
      ErrMessage<<"allied ";
   else
      ErrMessage<<"ennemy ";

   ErrMessage<<"ship is required in order to play the game"<<ends;
}

//*****************************************************************************
//*****************************************************************************

TScenario::TScenario(/*const TInitialPositions &AnInitialPosition,*/
                     const  TRulesOfEngagement &Rules,
                     const  TObjectives &AListOfObjectives,
                     vector<TShip*> &AlliedShips,
                     vector<TShip*> &EnnemyShips):
                     RulesOfEngagement(Rules),
                     Objectives(AListOfObjectives),
                     ListOfAlliedShips(AlliedShips),
                     ListOfEnemyShips(EnnemyShips)
{
   if (ListOfAlliedShips.empty())
      throw  TThereMustBeAtLeastOneShip(TShip::allied);

   if (ListOfEnemyShips.empty())
      throw  TThereMustBeAtLeastOneShip(TShip::allied);

   vector<TShip*> TempoListOfAlliedShips=const_cast <vector<TShip*> &>(ListOfAllie
dShips);

//Sort the ships by shipID

   sort(TempoListOfAlliedShips.begin(),TempoListOfAlliedShips.end(),TShip::Compar
ePointersToShip);

//Remove any duplications

    for  (vector<TShip*>::iterator it=TempoListOfAlliedShips.end();
```

```
            it == TempoListOfAlliedShips.begin()+1; it--)

            {
               if (*it == *(it-1))
                  TempoListOfAlliedShips.erase(it);
            }

//Store the ships initial position in order to reuse them to start the next
//round

    for(vector<TShip*>::iterator it =ListOfAlliedShips.begin();
                                    it != ListOfAlliedShips.end(); it++)
      TInitialPositions::AddANewShip((*it)->GetCurrentPosition());

};

//*****************************************************************************
//*****************************************************************************

} //End namespace
```

```
#ifndef AIMINGINFO_H
#include "SubExceptions.h"
#include "Coordinate.h"

namespace SUB_Framework
{

//*****************************************************************************
//*****************************************************************************

class TInvalidBearing : public TInitialisationError
{
   public:
      explicit TInvalidBearing(const unsigned InvalBearing);

   private:
      virtual void What() const;
      const unsigned InvalidBearing;
};

//*****************************************************************************
//*****************************************************************************

class TInvalidDepth : public TInitialisationError
{
   public:
      explicit TInvalidDepth(const unsigned InvalDepth);

   private:
      virtual void What() const;
      const unsigned InvalidDepth;
};

//*****************************************************************************
//*****************************************************************************

class TAimingInformation
{
   public:
      TAimingInformation(const unsigned Bear, const unsigned Depth);
      unsigned TAimingInformation::GetTargetBearing() const;
      unsigned TAimingInformation::GetTargetDepth() const;



   private:
      unsigned TargetBearing;
      unsigned TargetDepth;
};

//*****************************************************************************

inline unsigned TAimingInformation::GetTargetBearing() const
{
   return TargetBearing;
}

//*****************************************************************************

inline unsigned TAimingInformation::GetTargetDepth() const
```

```
{
   return TargetDepth;
}

//*************************************************************************************
//*************************************************************************************

} //End namespace

#define AIMINGINFO_H
#endif
```

```
#include <strstream>

#include "AimingInfo.h"
#include "Coordinate.h"

namespace  SUB_Framework
{
//*****************************************************************************
//*****************************************************************************

TInvalidBearing::TInvalidBearing(const unsigned InvalBearing) :
                                    InvalidBearing(InvalBearing)
{
}

//*****************************************************************************

void TInvalidBearing::What() const
{
   strstream ErrMessage(Message,sizeof (Message),ios::out);
   ErrMessage<<"Initialisation error: cannot aim at bearing"<<
               InvalidBearing<<'.'<<ends;
}

//*****************************************************************************
//*****************************************************************************

TInvalidDepth::TInvalidDepth(const unsigned InvalDepth) :
                                    InvalidDepth(InvalDepth)
{
}

//*****************************************************************************

void TInvalidDepth::What() const
{
   strstream ErrMessage(Message,sizeof (Message),ios::out);
   ErrMessage<<"Initialisation error: aiming at depth "<<
               InvalidDepth<<" which is greater then "<<
               TCoordinate::GetMaxDepth()<<" the maximum depth of the ocean"<<end
s;
}

//*****************************************************************************
//*****************************************************************************

TAimingInformation::TAimingInformation(const unsigned Bear, const unsigned Depth)
:
                                        TargetBearing(Bear), TargetDepth(Depth)

{
   if (TargetBearing > 360)
      TargetBearing%=360;

   if (TargetDepth > TCoordinate::GetMaxDepth())
      throw TInvalidDepth(TargetDepth);
}

//*****************************************************************************
//*****************************************************************************
```

```
} //End namespace
```

```
#ifndef COLLISION_H

#include "Ocean.h"

namespace SUB_Framework
{

//This class store information about a collision. A collision accurs when a
//ships occupies either a non navigable ocean position (reef, island, mines...)
//or the position of another non sunk ship in the game. When immersed,
//submarines never collides with suface ship.

//Set member functions are deleberatly omitted, ounce created, the information
//about a collision should never be changed. The compiler generated operator =
//can be used to change all the information.

//INFORMATION STORED MIGHT BE VALID ONLY DURING THE TURN OF THE SHIP THAT
//HAS CAUSED THE COLLISION. THE REFEREE CLASS MUST TAKE APPROPRIATE ACTION BASE
//ON THE INFORMATION CONTAINED HERE.

//***************************************************************************
//***************************************************************************

class TShip;

//***************************************************************************
//***************************************************************************

class TInvalidCollision: public TLogicError
{
   private :
      virtual void What() const;
};

//***************************************************************************
//***************************************************************************

class TCollisionInfo
{
   public :

      TCollisionInfo(); //The default constructor sets the variable
                        //ContentOfTheOceanAtTheSiteOfTheCollision to invalid_oce
an_content

      TCollisionInfo(const TCoordinate &ACoordinate, const unsigned Speed,
                     const TOceanContent Content,  const TShip *AnotherShip=NULL)
;

      TCoordinate GetPositionOfTheCollision() const;
      unsigned GetSpeedOfTheShipWhenTheCollisionHasOccured() const;
      TOceanContent GetContentOfTheOceanAtTheSiteOfTheCollision() const;
      const TShip * GetOtherShipImpliedInTheCollision() const;
      bool IsValid(); //Does an instance contains information about a collision
                      //or is it the creation of the default constructor?
   private :

      TCoordinate PositionOfTheCollision;
      unsigned SpeedOfTheShipWhenTheCollisionHasOccured;
      TOceanContent ContentOfTheOceanAtTheSiteOfTheCollision;
      const TShip *OtherShipImpliedInTheCollision;
```

```
};

//*****************************************************************************

inline TCoordinate TCollisionInfo::GetPositionOfTheCollision() const
{
   return PositionOfTheCollision;
}

//*****************************************************************************

inline unsigned TCollisionInfo::GetSpeedOfTheShipWhenTheCollisionHasOccured() con
st
{
   return SpeedOfTheShipWhenTheCollisionHasOccured;
}

//*****************************************************************************

inline TOceanContent TCollisionInfo::GetContentOfTheOceanAtTheSiteOfTheCollision(
) const
{
   return ContentOfTheOceanAtTheSiteOfTheCollision;
}

//*****************************************************************************

inline bool TCollisionInfo::IsValid()
{
  return ContentOfTheOceanAtTheSiteOfTheCollision != invalid_ocean_content;
}

//*****************************************************************************

inline const TShip * TCollisionInfo::GetOtherShipImpliedInTheCollision() const
{
   return  OtherShipImpliedInTheCollision;
}

//*****************************************************************************
//*****************************************************************************

} //End Namespace

#define COLLISION_H
#endif
```

```
#include <strstream>
#include "Collision.h"

namespace  SUB_Framework
{

//********************************************************************************
//********************************************************************************

   void TInvalidCollision::What() const
   {
      strstream ErrMessage(Message,sizeof (Message),ios::out);
      ErrMessage<<"Logic error: an collision has occured at an invalid ocean posi
tion."<<ends;

   }

//********************************************************************************
//********************************************************************************

   TCollisionInfo::TCollisionInfo() : PositionOfTheCollision(TCoordinate(0,0)),
                                      SpeedOfTheShipWhenTheCollisionHasOccured(0)
,
                                      ContentOfTheOceanAtTheSiteOfTheCollision(in
valid_ocean_content),
                                      OtherShipImpliedInTheCollision(NULL)
   {
   }

//********************************************************************************

   TCollisionInfo::TCollisionInfo(const TCoordinate &ACoordinate,
                                  const unsigned Speed,
                                  const TOceanContent Content, const TShip * Othe
rShip):
                                  PositionOfTheCollision(ACoordinate),
                                  SpeedOfTheShipWhenTheCollisionHasOccured(Speed)
,
                                  ContentOfTheOceanAtTheSiteOfTheCollision(Conten
t),
                                  OtherShipImpliedInTheCollision(OtherShip)


   {
      if (ContentOfTheOceanAtTheSiteOfTheCollision == invalid_ocean_content)
         throw TInvalidCollision();

   }

   //********************************************************************************
**
   //********************************************************************************
**

} //End namespace
```

```
#ifndef NOOCEAN_H

#include "Ocean.h"

namespace  SUB_Framework
{

//***************************************************************************
//***************************************************************************

class TThereMustBeAnOcean : public TInitialisationError
{
   private :
      virtual  void What() const;
};

//***************************************************************************
//***************************************************************************

} //End Namespace

#define NOOCEAN_H
#endif
```

```
#include <strstream>
#include "NoOcean.h"

namespace  SUB_Framework
{

//******************************************************************************
//******************************************************************************

void TThereMustBeAnOcean::What() const
{
   strstream ErrMessage(Message,sizeof (Message),ios::out);
   ErrMessage<<"Initialisation error: there must be one ocean."<<ends;
}

//******************************************************************************
//******************************************************************************

} //End Namespace
```

```
#ifndef THESCREEN_H

#include <conio.h>

#include "Ocean.h"
#include "Ship.h"
#include "Referee.h" //For the enumerated type TValidOrder

namespace SUB_Framework
{

//*****************************************************************************
//*****************************************************************************

class TThereCanBeOnlyOneOutputScreen : public TLogicError
{
   private :
      virtual void What() const;
};

//*****************************************************************************
//*****************************************************************************

class TScreen
{
   public :
      explicit TScreen(const TOcean &AnOcean);
      virtual void PrepareANewRound()=0;
      virtual void DrawTheOcean() const=0;
      virtual void DrawShip(const TShip &AShip,
                            const TCoordinate &PreviousPosition)const =0;
      virtual void DisplayShipInfo (const TShip &AShip)=0;
      virtual void GetNewHeadingAndSpeed(unsigned &NewHeading, unsigned &NewSpeed
)=0; //Store in into FutureHeading and
                                                  //FutureSpeed
      virtual unsigned GetAimingDirection() const=0;
      virtual void GetAimingInfo() =0;
      virtual TAimingInformation GetAimingInformation() const=0;


      //We cannot assume that cout is supported on all platforms.
      //We all know that cout doesn't work in Windows 3.1

      virtual void ShowFutureSpeed()=0;
      virtual void ShowFutureHeading()=0;
      virtual void ShowInfractionMessage(const string &InfractionMessage)=0;
      virtual void ShowInformationMessage(const string &InformationMessage)=0;
      virtual void ClearInfractionWindow()=0;
      virtual void ClearInformationWindow()=0;
      virtual void ClearAimingInfoWindow()=0;
      virtual void ShowAimingInformation()=0;
      virtual bool GetConfirmation(const string &Message)=0;
      virtual void PressAnyKeyToContinue(const string &Message)=0;

      virtual TReferee::TValidOrders GetInstructions()=0;

      virtual void EraseAShip(const TCoordinate &PositionOfTheShip)=0;
      virtual ~TScreen();

   protected :
      virtual void GetAimingDirectionFromTheUser()=0;
```

```
        virtual void GetAimingDepthFromTheUser()=0;

        const TOcean &TheOcean;

        //Information on the currently displayed ship

        unsigned FutureSpeed, FutureHeading, MaximumSpeed, AimingDirection;
        unsigned AimingDepth;

};

//*****************************************************************************

//AimingDirection is set to an invalid value to indicate that the user
//hasn't aimed yet.

inline TScreen::TScreen(const TOcean &AnOcean): TheOcean(AnOcean),
                                                AimingDirection(361)
{
}

//*****************************************************************************

inline TScreen::~TScreen()
{
}

//*****************************************************************************
//*****************************************************************************


} //End namespace


#define THESCREEN_H
#endif
```

```
#include <strstream>
#include <conio>
#include <limits.h>
#include <ctype.h> // for toupper

#include "TheScreen.h"
#include "ArrowKeysCode.h"

namespace  SUB_Framework
{

//***************************************************************************

void TThereCanBeOnlyOneOutputScreen::What() const
{
   strstream ErrMessage(Message,sizeof (Message),ios::out);
   ErrMessage<<"Logic error: there can be only one output screen."<<ends;

}

//***************************************************************************
//***************************************************************************

} // End namespace
```

```
#ifndef GAME_H

#include <vector>

#include "Ship.h"
#include "Player.h"
#include "SubExceptions.h"
#include "TextScreen.h"
#include "Scenario.h"

namespace SUB_Framework
{

//*****************************************************************************
//*****************************************************************************

class TAtLeastOneShipIsRequiredInOrderToPlayTheGame: public TLogicError
{
   virtual void What() const;
};

//*****************************************************************************
//*****************************************************************************

class TThereCanBeOnlyOneGame :public TLogicError
{
   virtual void What() const;
};

//*****************************************************************************
//*****************************************************************************

class TGame
{
   public:
      TGame(TOcean &AnOcean, TReferee &AReferee, TScreen &AScreen);


      virtual void Play();

      virtual ~TGame();

   protected:

      TOcean & TheOcean;
      TReferee  &Referee;
      TScreen &TheScreen;
      virtual void PlayARound();
      virtual void PlayATurn(TShip * const ActiveShip);

   private:


      //Dissallow copying and assignment.
      //There must be no definition for the following two functions

      TGame(const TGame &AnotherGame);
      TGame& operator =(const TGame &AnnotherGame);


      static unsigned NumberOfActiveInstance;
```

```
        vector<TPlayer*> ListOfPlayers;
};

//*****************************************************************************
//*****************************************************************************

} //end namespace

#define GAME_H
#endif
```

```
#include <strstream>
#include "Game.h"
#include "Referee.h"


namespace  SUB_Framework
{

//*****************************************************************************
//*****************************************************************************

void TThereCanBeOnlyOneGame::What() const
{
   strstream ErrMessage(Message,sizeof (Message),ios::out);
   ErrMessage<<"Logic error: there can be only one game in progress."<<ends;

}

//*****************************************************************************
//*****************************************************************************

void TAtLeastOneShipIsRequiredInOrderToPlayTheGame::What() const
{
   strstream ErrMessage(Message,sizeof (Message),ios::out);
   ErrMessage<<"Logic error: at least one ship is required to play the game."<<en
ds;

}

//*****************************************************************************
//*****************************************************************************

//Static members definition

unsigned  TGame::NumberOfActiveInstance=0;


//*****************************************************************************
//*****************************************************************************

TGame::TGame(TOcean &AnOcean, TReferee &AReferee, TScreen &AScreen) :
              TheOcean(AnOcean), Referee(AReferee), TheScreen(AScreen)

{

   if (++NumberOfActiveInstance > 1)
      throw TThereCanBeOnlyOneGame();


}

//*****************************************************************************


void TGame::Play()
{
   while (!Referee.GameOver())
   {
      Referee.StartANewRound();
      PlayARound();
   }
```

```
    TheScreen.ClearInformationWindow();
    strstream WinningMessage;
    WinningMessage<<"Congratulation "<<Referee.WhoIsTheWinner()<<" you have won th
e game"<<ends;
    char * WinMessage=WinningMessage.str();
    TheScreen.PressAnyKeyToContinue(WinMessage);
    delete [] WinMessage; //We must delete it
}

//*****************************************************************************

void TGame::PlayARound()
{
    TShip * ActiveShip;

    TheScreen.PrepareANewRound();
    while (!Referee.RoundOver())
    {

        ActiveShip=Referee.GetActiveShip();

        PlayATurn(ActiveShip);
        Referee.GetNextShip();
    }
}

//*****************************************************************************

void TGame::PlayATurn(TShip * const ActiveShip)
{

    Referee.MoveTheEnemies();
    ActiveShip->ComputeTargetRange(); //The targer has just moved
    TheScreen.DisplayShipInfo(*ActiveShip);
    TheScreen.ClearInfractionWindow();

    if (!ActiveShip->GetIsDammaged())
    {
        bool HasAlreadyFired=false,HasCommittedAnInfraction=false;
        bool HasAlreadyMoved=false;

        TReferee::TValidOrders Order; //Must be declared outside the loop because
                                      //we need its value after the end of the loop

        Order=TReferee::no_orders; //To enter the loop, value is irrelevant,

        //TheScreen.DrawShip(*Referee.ListOfEnemyShips[0],TCoordinate(0,0));
        while (Order != TReferee::end_of_the_turn)
        {
            Order = TheScreen.GetInstructions();

            if (HasCommittedAnInfraction &&
                (Order != TReferee::end_of_the_turn &&
                 Order != TReferee::abandon_ship))
            {
                Order=TReferee::no_orders; //No other actions are permitted after
                                           //an infraction
            }

            switch (Order)
```

```
            {
               case TReferee::end_of_the_turn:
                  break;

               case TReferee::set_course_and_speed:
               {
                  if (!HasAlreadyMoved)
                  {
                     unsigned NewHeading, NewSpeed;
                     TheScreen.GetNewHeadingAndSpeed(NewHeading,NewSpeed);
                     TCoordinate PreviousPosition=ActiveShip->GetCurrentPosition();
                     if (NewSpeed > 0)
                     {
                        ActiveShip->MoveOnTheOcean(NewHeading,NewSpeed);
                        HasCommittedAnInfraction=Referee.WasThereAnInfraction();
                        if (PreviousPosition != ActiveShip->GetCurrentPosition())
                        {
                           TheScreen.DrawShip(*ActiveShip,PreviousPosition);
                        }
                        if (HasCommittedAnInfraction)
                        {
                           TheScreen.ShowInfractionMessage(Referee.GetDescriptionOfT
heInfraction());
                        }

                        TheScreen.DisplayShipInfo(*ActiveShip);
                        HasAlreadyMoved=true;

                     }//speed zero displacement isn't a movement
                  }
                  else
                     TheScreen.ShowInfractionMessage("You have already moved     dur
ing this turn");
                  break;
               }


               case TReferee::open_fire:
               {

                  if (!HasAlreadyFired)
                  {
                     TShip::TFiringResult FiringResult=
                           ActiveShip->OpenFire(TheScreen.GetAimingInformation());
                     switch (FiringResult)
                     {
                        case TShip::sunk:
                        {

                           TheScreen.ShowInformationMessage("Congratulation you have
   sunked an ennemy submarine.");
                           Order=TReferee::end_of_the_turn;
                           int c;
                           c=getch(); //Wait for a keystroke
                           break;

                        }
                        case TShip::sos:
                        {
                           TheScreen.ShowInfractionMessage("SOS");
                           break;
```

```
                        }
                        case TShip::off_1:
                        {
                           TheScreen.ShowInformationMessage("Off 1");
                           break;
                        }

                        case TShip::off_2:
                        {
                           TheScreen.ShowInformationMessage("Off 2");
                           break;
                        }
                     }
                     HasAlreadyFired=true;
                  }
                  else
                     TheScreen.ShowInfractionMessage("You have already fired in this
 turn.");
                  break;
               } //end case

               case TReferee::abandon_ship:
               {
                  if(TheScreen.GetConfirmation("Do you really want to      abandon y
our ship?"))
                  {
                     TheScreen.EraseAShip(ActiveShip->GetCurrentPosition());
                     Referee.AbandonShip(ActiveShip);
                     Order=TReferee::end_of_the_turn; //There is no longer a ship to
 move
                  }
                  break;
               }
               case TReferee::perform_repairs: //This is automatic in the basic game
               {
                  break;
               }

               case TReferee::no_orders:
               {
                  break;
               }
               default:
                  throw TInvalidOrders();
            } //End case

         }//end while
      }

      else
      {
         TheScreen.PressAnyKeyToContinue("Your ship has been dammagedyou must loose
this turn   repairing it"); //Write a message to the player
         ActiveShip->DoRepairs(); //Loose the turn to repair the ship
      }

}


//***************************************************************************
```

```
TGame::~TGame()
{
   NumberOfActiveInstance--;
};

//*****************************************************************************
//*****************************************************************************


} //End namespace
```

```
// This class strore the information about the game player

#ifndef PLAYER_H

#include <string>

#include "rank.h"

namespace SUB_Framework
{

//class TPlayer must have a default constructor because a vector of TPlayer
//must be maintained by the referee.

class TReferee;

class TPlayer
{
   friend ostream& operator <<(ostream &os, const TPlayer APlayer);
   friend TReferee;

   public :
      TPlayer(const string &PName="Error", const unsigned ARankIndex=0,
              const unsigned StartingScore=0);

      bool operator ==(const TPlayer &APlayer) const ;
      bool operator  <(const TPlayer &APlayer) const ;

   protected :

      bool Promote();

   private :
      string Name;
      unsigned Score;
      unsigned RankIndex, NavyID;
      static unsigned NextNavyID;

      //vector<TShip*> ShipCommanded;
};


inline bool TPlayer::operator ==(const TPlayer &APlayer) const
{
   return NavyID == APlayer.NavyID;
}

//Usefull only to put on ordered data structures
//such as a binary search tree

inline bool TPlayer::operator <(const TPlayer &APlayer) const
{
   return NavyID < APlayer.NavyID;
}

} // End namespace

#define PLAYER_H
#endif
```

```
#include "Player.h"
#include "PromotionPath.h"
namespace  SUB_Framework
{

//*************************************************************************

ostream& operator <<(ostream &os, const  TPlayer APlayer)
{
   os<<TPromotionPath::ConvertRankIndexToRank(APlayer.RankIndex)<<"  "<<APlayer.N
ame; //tempo
   return  os;
}

//*************************************************************************
//*************************************************************************

//Static member definition

unsigned  TPlayer::NextNavyID=0;

//*************************************************************************
//*************************************************************************

TPlayer::TPlayer(const  string &PName, const unsigned  ARankIndex,
                 const unsigned  StartingScore):
                 Name(PName),RankIndex(ARankIndex),Score(StartingScore)
{
   NavyID=NextNavyID++;
}

//*************************************************************************

bool  TPlayer::Promote()
{
   TRank FutureRank=TPromotionPath::GetNextRank(TPromotionPath::ConvertRankIndexT
oRank(RankIndex));
   if (FutureRank!=TPromotionPath::ConvertRankIndexToRank(RankIndex))
   {
      RankIndex++;
      return  true;
   }
   else
      return  false;
   //else the player is already at the top rank, he cannot be promoted.

}
//*************************************************************************
//*************************************************************************

} //End namespace
```

```
#ifndef REFEREE_H

#include <vector>
//#include <list>
//#include "Ship.h"
#include "Player.h"
#include "SubExceptions.h"
//#include "TheScreen.h"

/*#include "scenario.h"
#include "Submarine.h" */
#include "Collision.h"
#include "AimingInfo.h"
#include "SunkInfo.h"

#include "AbstractReferee.h"

namespace SUB_Framework
{

//*****************************************************************************
//*****************************************************************************

class TThereCanBeOnlyOneReferee: public TLogicError
{
   private :
      virtual void What() const;
};

//*****************************************************************************
//*****************************************************************************

class TTheWinnerMustNotBeNULL : public TLogicError
{
   private :
      virtual void What() const;
};

//*****************************************************************************
//*****************************************************************************

//It is tempting to pass this class the invalid value but copying values outside
//of the unumeration bounds in an UNSPECIFIED BEHAVIOR.

class TInvalidInfractionType : public TLogicError
{
   private :
      virtual void What() const;
};

//*****************************************************************************
//*****************************************************************************

class TShip; //Referee need to declare reference to this type

//*****************************************************************************
//*****************************************************************************

class TReferee : public TAbstractReferee
{
   //friend TGame; //To monitor the enemy position, must be remove before final
```

```
                    //submission
        public :

            //In the basic game, all repairs are performed automatically so there will
            //be no use for the last item. Also, basic game ships are never sunk so
            //there will be no need to abandon them unless a player get tired of
            //playing.

            typedef enum {set_course_and_speed, open_fire, abandon_ship,
                          perform_repairs, end_of_the_turn, no_orders} TValidOrders;

            typedef enum {none, collision, run_aground, out_of_ammunition,
                          out_of_fuel, target_out_of_firing_range,
                          dammaged_ship_cannot_open_fire,
                          movement_in_violation_of_ship_specifications,
                          cant_open_fire, your_are_hit_by_ennemy_fire} TInfractionType;


            TReferee(const TScenario& AScenario);


            virtual void Initialize(vector<TShip*> &ListOfShips);
            virtual bool GameOver() const;
            virtual bool RoundOver()const;
            virtual TPlayer WhoIsTheWinner() const;
            virtual TPlayer WhoIsTheCurrentRoundWinner() const;

            //virtual void PlayATurn(TShip &AShip) const ;
            virtual void MoveTheEnemies();
            virtual void RepairAShip(TShip &ADammagedShip) const;

            virtual void MoveOnTheOcean(TShip &AShip, const unsigned NewHeading,
                                        const unsigned NewSpeed);

            virtual void MoveEnemyOnTheOcean(TShip &AShip, const unsigned NewHeading,
                                             const unsigned NewSpeed);

            virtual TShip::TFiringResult OpenFire(TShip& AShip, const TAimingInformatio
        n &AimingInfo);

            virtual void ComputeTargetRange(TShip &AShip) const;

            virtual void TakeControl(TSubmarine *AShip);

            const TScenario& GetScenario() const;
            TInfractionType GetMostRecentInfraction() const;

            virtual void StartANewRound();

            TShip* GetActiveShip() const;
            TShip* GetNextShip();

            const vector<TShip*>& GetListOfAlliedShips() const;

            const TCollisionInfo& GetCollisionInfo() const;

            virtual void Penalize(TShip &TheShipToPenalize);
            //We pass the ship to penalize because it might be
            //necessary to penalize a ship that is not the
            //currently active ship when the game will be
            //expanded.
```

```
      virtual ~TReferee();

      static const TOcean* GetTheOcean();
      static void SetTheOcean(const TOcean* AnOcean);
      bool WasThereAnInfraction();
      const string& GetDescriptionOfTheInfraction() const;
      void AbandonShip(const TShip * const AShip);

      void PerformEvasiveManeuvers(TSubmarine &EvadingSubmarine);

   protected:
      virtual void IsThereACollisionWithAnotherObject(const TCoordinate &Position
,
                                                      const TShip &AShip,
                                                      const TShip *& VictimOfTheCollisi
on,
                                                      bool &ThereIsACollision);

      virtual void GetNextCoordinate(const unsigned CurrentSpeed,
                                     const int DeltaX,
                                     const int DeltaY,
                                     TCoordinate &ACoordinate,
                                     bool &IsLegalMove);

      virtual void ComputeDeltaXAndDeltaY(const unsigned Heading,
                                          int &DeltaX, int &DeltaY);
   private:


      TReferee(const TReferee &AnotherReferee); //Forbid copying
      TReferee& operator =(const TReferee &LHS); // Forbid assignment

      const TScenario &TheScenario;

      const TPlayer *WinnerOfTheGame, *CurrentRoundWinner, *ActivePlayer;

      vector<TShip*> ListOfEnemyShips;

      TCollisionInfo CollisionInfo;

      static unsigned NumberOfActiveInstance;

      static const TOcean *TheOcean; //The referee need an acces to the
                                     //content of the ocean but it should
                                     //normally not modify it. On the rare
                                     //occasions that this would be usefull,
                                     //such as the adding of a ship wreakcage
                                     //on some square, a const_cast must be used
                                     //to draw the attention to the fact that
                                     //this is abnormal.

      unsigned DeltaDepth;
      vector<TPlayer*> ListOfPlayers;

      vector<TShip*> &ListOfAlliedShips;
      vector<TShip*>::iterator ActiveShip;
      TInfractionType MostRecentInfraction;

      string DescriptionOfTheInfraction;
      vector<TSunkInfo> ShipsSunkDuringTheCurrentTurn;
```

```
      bool GameIsOver;
};

//*****************************************************************************

inline const TScenario& TReferee::GetScenario() const
{
   return TheScenario;
}

//*****************************************************************************

inline TShip* TReferee::GetActiveShip() const
{
   return *ActiveShip;
}

//*****************************************************************************

void TReferee::RepairAShip(TShip &ADammagedShip) const
{
   ADammagedShip.SetIsDammaged(false);
}

//*****************************************************************************

class TInvalidOrders : public TLogicError
{
   public:
      virtual void What() const;
};

//*****************************************************************************

inline const vector<TShip*>& TReferee::GetListOfAlliedShips() const
{
      return ListOfAlliedShips;
}

//*****************************************************************************

inline const TOcean* TReferee::GetTheOcean()
{
   return TheOcean;
}

//*****************************************************************************

inline void TReferee::SetTheOcean(const TOcean *AnOcean)
{
   TheOcean=AnOcean;
}

//*****************************************************************************

inline const TCollisionInfo& TReferee::GetCollisionInfo() const
{
   return CollisionInfo;
}

//*****************************************************************************
```

```
inline const string& TReferee::GetDescriptionOfTheInfraction() const
{
   return DescriptionOfTheInfraction;
}

//*****************************************************************************

inline TReferee::TInfractionType TReferee::GetMostRecentInfraction() const
{
   return MostRecentInfraction;
}

//*****************************************************************************
//*****************************************************************************

} //End namespace

#define REFEREE_H
#endif
```

```
#include <strstream>
#include <conio.h> //tempo must be removed for final version
#include "Referee.h"
#include "Ship.h"
#include "NoOcean.h"

namespace  SUB_Framework
{

//***************************************************************************
//***************************************************************************

//Definition of static member

const TOcean *TReferee::TheOcean=NULL;

//***************************************************************************
//***************************************************************************

void TThereCanBeOnlyOneReferee::What() const
{
   strstream ErrMessage(Message,sizeof (Message),ios::out);
   ErrMessage<<"Logic error: there can be only one referee."<<ends;

}

//***************************************************************************
//***************************************************************************

void TInvalidOrders::What() const
{
   strstream ErrMessage(Message,sizeof (Message),ios::out);
   ErrMessage<<"Logic error: there can be only one referee."<<ends;

}

//***************************************************************************
//***************************************************************************

void TInvalidInfractionType::What() const
{
   strstream ErrMessage(Message,sizeof (Message),ios::out);
   ErrMessage<<"Logic error: the type of infraction is "
             <<"outside of the allowed range."<<ends;
}

//***************************************************************************
//***************************************************************************

void TTheWinnerMustNotBeNULL::What() const
{
   strstream ErrMessage(Message,sizeof (Message),ios::out);
   ErrMessage<<"Logic error: the winner is equal to NULL"<<ends;

}
//***************************************************************************
//***************************************************************************

unsigned  TReferee::NumberOfActiveInstance=0;

TReferee::TReferee(const  TScenario& AScenario):TheScenario(AScenario),
```

```
                        ListOfAlliedShips(AScenario.GetListOfAlliedShips()),
                        ListOfEnemyShips(AScenario.GetListOfEnemyShips()),
                        WinnerOfTheGame(NULL), CurrentRoundWinner(NULL),
                        ActivePlayer(NULL), ActiveShip(NULL),
                        MostRecentInfraction(none),GameIsOver(false)

{
   //There is no need to used other trick, in this case I totally with
   //the computer definition of the number of instance.

   if (++NumberOfActiveInstance > 1)
      throw TThereCanBeOnlyOneReferee();

}

//***************************************************************************

void TReferee::ComputeTargetRange(TShip &AShip) const
{

   TCoordinate Tempo = AShip.GetCurrentPosition();
   unsigned TargetRange=Tempo.ComputeDistance(ListOfEnemyShips[0]->GetCurrentPosi
tion());
   AShip.SetTargetRange(TargetRange); //For the basic game
   AShip.SetCanOpenFire(TargetRange <= AShip.GetWeaponRange() && !AShip.GetIsDamm
aged());

}

//***************************************************************************
//***************************************************************************

void TReferee::Initialize(vector<TShip*> &ListOfShips)
{
   vector<TShip*>::iterator First=ListOfShips.begin(), Last=ListOfShips.end(), it
;
   unsigned ShipID;

   for (it=First; it != Last; it++)
   {
      ShipID=(*it)->GetID();
      (*it)->SetCurrentPosition(TheScenario.GetInitialPositions()[ShipID-1]);
   }
}

//***************************************************************************
//***************************************************************************

void TReferee::TakeControl(TSubmarine *AShip)
{
   ListOfEnemyShips.push_back(AShip); //Add duplication control if needed
}

//***************************************************************************
//***************************************************************************

bool TReferee::GameOver() const
{
   return GameIsOver; //tempo
}
```

```
//*****************************************************************************
//*****************************************************************************

bool TReferee::RoundOver() const
{
   return ListOfEnemyShips.empty();
}

//*****************************************************************************

TPlayer TReferee::WhoIsTheWinner() const
{
   if (WinnerOfTheGame != NULL)
      return *WinnerOfTheGame;
   else
      throw TTheWinnerMustNotBeNULL();
}

//*****************************************************************************

TPlayer TReferee::WhoIsTheCurrentRoundWinner() const
{
   return *CurrentRoundWinner;
}

//*****************************************************************************

/*void TReferee::PlayATurn(TShip &AShip) const
{
   cout<<AShip.GetAllignment(); //Tempo
};*/

//*****************************************************************************

TShip* TReferee::GetNextShip()
{
   ActiveShip++;
   if (ActiveShip == ListOfAlliedShips.end())
      ActiveShip=ListOfAlliedShips.begin();

   //When we go to the next ship, it is the begining of a new turn, we
   //must clear the list of sunk ships during the current turn

   if (!ShipsSunkDuringTheCurrentTurn.empty())
      ShipsSunkDuringTheCurrentTurn.erase(ShipsSunkDuringTheCurrentTurn.begin(),
                                          ShipsSunkDuringTheCurrentTurn.end());

   return *ActiveShip;
}

//*****************************************************************************

void TReferee::ComputeDeltaXAndDeltaY(const unsigned Heading, int &DeltaX, int &D
eltaY)
{

   DeltaX=DeltaY=0; //Ensure initialisation of output parameters

   switch (Heading)
   {
      case 0:
```

```
        {
            DeltaX=1;
            break;
        }
        case 45:
        {
            DeltaX=DeltaY=1;
            break;
        }
        case 90:
        {
            DeltaY=1;
            break;
        }
        case 135:
        {
            DeltaY=1;
            DeltaX=-1;

            break;
        }
        case 180:
        {
            DeltaX=-1;
            break;
        }
        case 225:
        {
            DeltaX=DeltaY=-1;
            break;
        }
        case 270:
        {
            DeltaY=-1;
            break;
        }
        case 315:
        {
            DeltaX=1;
            DeltaY=-1;
            break;
        }

    } //End switch
}

//*****************************************************************************

// This function compute the coordinate that will be reached by performing a
// displacement of one unit starting at ACoordinate and moving in the direction
// of Heading for one unit. If the move is legal, ACoordinate will hold the
// value of the destination coordinate, IsLegalMove will be true and
// NextCoordinateContent will hold the content of the ocean at the new coordinate
.
//
// If the planned movement is ILLEGAL, ACoordinate will remain unchanged,
// IsLegalMove will be false. If the intended destination coordinate is
// legal, NextCoordinateContent will be the content of the intended destination
// coordinate, if not, it will be invalidOceanContent

void TReferee::GetNextCoordinate(const unsigned CurrentSpeed,
```

```
                                        const int DeltaX,
                                        const int DeltaY,
                                            TCoordinate &ACoordinate,
                                            bool &IsLegalMove)
{
   TCoordinate PreviousCoordinate=ACoordinate; //Save it because in case of
                                               //error we must restore the
                                               //previous coordinate.
   IsLegalMove=false;


   unsigned Latitude=ACoordinate.GetLatitude(),
            Longitude=ACoordinate.GetLongitude();

   //It is an undefined behavior to attempt to store a negative value into an
   //unsigned int. There are two special cases for which this will happens, they
   //are dealt with here.

   if (Latitude == 0 && DeltaY == -1)
   {
      MostRecentInfraction=run_aground;
      return; //Moving out of the map
   }

   if (Longitude == 0 && DeltaX == -1)
   {
      MostRecentInfraction=run_aground;
      return; //Moving out of the map
   }

   unsigned NewLongitude=Longitude+DeltaX,
            NewLatitude=Latitude+DeltaY;

   if (TCoordinate::WouldBeLegalCoordinate(NewLongitude,NewLatitude))
   {
      if (TheOcean != NULL)
      {
         TCoordinate NewCoordinate(NewLongitude,NewLatitude,ACoordinate.GetDepth(
));
         if (IsLegalMove=TheOcean->IsNavigable(NewCoordinate))
            ACoordinate=NewCoordinate;
         else
         {

            CollisionInfo=TCollisionInfo(NewCoordinate, CurrentSpeed,
                                         TheOcean->GetWhatIsAtCoordinate(NewCoord
inate),
                                         NULL);
         }

      }
      else
         throw TThereMustBeAnOcean();

   }
   else //The coordinates are not on the ocean
      MostRecentInfraction=run_aground;

}
```

```
//*************************************************************************

void TReferee::MoveOnTheOcean(TShip &AShip, const unsigned NewHeading,
                              const unsigned NewSpeed)
{
   unsigned CurrentSpeed=AShip.GetCurrentSpeed(),
            CurrentHeading=AShip.GetCurrentHeading(),
            CurrentTurningRate=AShip.GetCurrentTurningRate();

   TCoordinate CurrentPosition=AShip.GetCurrentPosition();

   MostRecentInfraction = none;
   DescriptionOfTheInfraction="";

   if (!AShip.GetIsDammaged())
   {
      if (NewSpeed <= AShip.GetMaxSpeed() &&
          abs(static_cast <int>(CurrentHeading)-NewHeading)%181 <=CurrentTurningRa
te &&
          abs(static_cast <int>(CurrentSpeed)-NewSpeed) <= AShip.GetCurrentAcceler
ationRate())
      {
         //cout<<static_cast<int>(CurrentHeading)-NewHeading<<endl;
         AShip.SetCurrentHeading(NewHeading); //Even if the ship can't move we
                                              //considered that it has changed t
o its new heading
         AShip.SetCurrentSpeed(NewSpeed);

         unsigned IntersectionsToMove=NewSpeed; //Number of intersections remaini
ng to be crossed
         bool MovementIsLegal=true;
         //TCoordinate CurrPosition=CurrentPosition;
         //TOceanContent ContentOfTheNextCoordinate; //The content of the next oc
ean
                                                   //intersection on the planned
path

         //TCollisionInfo *InformationAboutACollision=NULL;

          int DeltaX, DeltaY;
          ComputeDeltaXAndDeltaY(NewHeading,DeltaX, DeltaY);
          if (DeltaX ==0 && DeltaY ==0 )
            IntersectionsToMove = 0; //Do nothing the heading is illegal

         while (IntersectionsToMove > 0 && MovementIsLegal)
         {

            //TCoordinate NextPositionToMove

            GetNextCoordinate(NewSpeed, DeltaX, DeltaY, CurrentPosition,MovementI
sLegal);

            if(MovementIsLegal) //Try to move one unit in the requested position
            {
               //GetNextCoordinate has checked that the ocean is navigable
               //it remains to be checked that there is not a collision with
               //another seagoing object such as other ship.

               bool ThereIsACollision=true; //To avoid a warning
               const TShip *VictimOfTheCollision;
               IsThereACollisionWithAnotherObject(CurrentPosition,AShip,
```

```
                                        VictimOfTheCollision, ThereIsACollision);
                    if (!ThereIsACollision)
                    {
                        IntersectionsToMove--;
                        AShip.SetCurrentPosition(CurrentPosition);


                    }
                    else
                    {
                        CollisionInfo=TCollisionInfo(CurrentPosition,
                                                     CurrentSpeed,
                                                     TheOcean->GetWhatIsAtCoordinate(Cu
rrentPosition),
                                                     VictimOfTheCollision);
                        MovementIsLegal=false;
                        MostRecentInfraction=collision;
                        Penalize(AShip);

                    }


                }
                else  // The intended movement is illegal
                {
                    Penalize(AShip);
                    if (MostRecentInfraction != run_aground)
                    {
                        //Collision with someting other then a ship


                    }  //cout<<"Error the position at coordinate (to be completed) is
not navigable"; //Tempo



                }


            } //end while

            if (IntersectionsToMove == 0)
                AShip.SetCurrentSpeed(NewSpeed);
            else
                        //If the ship has it an obstruction, such as an island or
                        //another ship, its speed is set to 0.
                AShip.SetCurrentSpeed(0);
        }
        else  //End if speed and change of heading are legal
        {
            MostRecentInfraction=movement_in_violation_of_ship_specifications;
        }

    }
    else //The ship is damaged
        AShip.DoRepairs();

    AShip.ComputeTargetRange();
    AShip.SetCanOpenFire(AShip.GetTargetRange() <= AShip.GetWeaponRange());
};
```

```
//*****************************************************************************

void TReferee::MoveEnemyOnTheOcean(TShip &AShip, const unsigned NewHeading,
                                   const unsigned NewSpeed)
{
   MoveOnTheOcean(AShip,NewHeading,NewSpeed);

   if (TheOcean->GetWhatIsAtCoordinate(AShip.GetCurrentPosition()) != water)

   //if (MostRecentInfraction != none ) //The submarine has hitted something
   {
      //Reverse course, put the submarine back on the sea


      unsigned ReverseCourseHeading=(AShip.GetCurrentHeading()+180)%360;

      //Compute a random change of heading, the following algorithm is
      //good only for the basic game.

      MoveOnTheOcean(AShip,ReverseCourseHeading,1);
      int RandomChangeOfHeading=random(3);
      int DeltaHeading=0;

      if (RandomChangeOfHeading==0)
         ReverseCourseHeading=(ReverseCourseHeading+45)%360;
      else if (RandomChangeOfHeading==1)
      {
         if (ReverseCourseHeading !=0 )
            ReverseCourseHeading -= 45;
         else
            ReverseCourseHeading=315;
      }


      AShip.SetCurrentHeading(ReverseCourseHeading+DeltaHeading%360);


      MoveOnTheOcean(AShip,ReverseCourseHeading,NewSpeed);
      MostRecentInfraction=none;
   }
}

//*****************************************************************************

TShip::TFiringResult TReferee::OpenFire(TShip& AShip, const TAimingInformation &A
imingInfo)
{

   if (AShip.GetCanOpenFire())
   {
      const unsigned TargetRange = AShip.GetTargetRange();
      //const unsigned TargetDepth = AimingInfo.GetTargetDepth();
      TAimingCoordinate PositionUnderFire;

      PositionUnderFire=AShip.GetCurrentPosition();
      PositionUnderFire.SetDepth(AimingInfo.GetTargetDepth());

      switch (TargetRange)
      {
         case 2:
```

```
            {
               int DeltaX, DeltaY;
               ComputeDeltaXAndDeltaY(AimingInfo.GetTargetBearing(),DeltaX,DeltaY);
               if (TCoordinate::WouldBeLegalCoordinate(
                    PositionUnderFire.GetLongitude()+DeltaX,
                    PositionUnderFire.GetLatitude()+DeltaY))
               {
                  PositionUnderFire.SetLongitude(PositionUnderFire.GetLongitude()+De
ltaX);
                  PositionUnderFire.SetLatitude(PositionUnderFire.GetLatitude()+Delt
aY);

               }
               else
                  return TShip::sos;

            }
            case 1:
            {
               int DeltaX, DeltaY;
               ComputeDeltaXAndDeltaY(AimingInfo.GetTargetBearing(),DeltaX,DeltaY);
               if (TCoordinate::WouldBeLegalCoordinate(
                    PositionUnderFire.GetLongitude()+DeltaX,
                    PositionUnderFire.GetLatitude()+DeltaY))
               {
                  PositionUnderFire.SetLongitude(PositionUnderFire.GetLongitude()+De
ltaX);
                  PositionUnderFire.SetLatitude(PositionUnderFire.GetLatitude()+Delt
aY);

               }
               else
                  return TShip::sos;

               break;
            }

            case 0:
            {

               break;
            }


            //Check for a hit


            //default:
               //throw TShipClassInvariantHasBeenViolated();

         }//End switch

         //remove casualties

         DeltaDepth=0;
         vector<TShip*>::iterator it,End=ListOfEnemyShips.end();
         for (it=ListOfEnemyShips.begin(); it <End; it++)
         {
            TCoordinate CurrentPosition=(*it)->GetCurrentPosition();
            if (CurrentPosition == PositionUnderFire)
            {
```

```
                ShipsSunkDuringTheCurrentTurn.push_back(TSunkInfo(ActivePlayer,*it));
                ListOfEnemyShips.erase(it);
                CurrentRoundWinner=const_cast <TPlayer*>((*ActiveShip)->GetCommandingO
 fficer());
                const_cast <TPlayer*>(CurrentRoundWinner)->Promote();
                GameIsOver = !AShip.Improve();


                if (GameIsOver)
                   WinnerOfTheGame=CurrentRoundWinner;



                return TShip::sunk;

             }
             //Good position ,wrong depth

             else if (CurrentPosition.GetLatitude() == PositionUnderFire.GetLatitude(
 ) &&
                CurrentPosition.GetLongitude() == PositionUnderFire.GetLongitude())
             {
                DeltaDepth=abs(static_cast <int>(CurrentPosition.GetDepth())-PositionU
 nderFire.GetDepth());

                (dynamic_cast <TSubmarine*>(*it))->PerformEvasiveManeuvers();

                return static_cast <TShip::TFiringResult>(DeltaDepth);
             }
             else //The taget has been missed and will retaliate
             {

                MostRecentInfraction=your_are_hit_by_ennemy_fire;
                Penalize(**ActiveShip);
                return TShip::sos;
             }
          }
       }
       else //For some reason(s), the ship cannot open fire at this moment.
       {
          if (AShip.GetTargetRange() > AShip.GetWeaponRange())
             MostRecentInfraction = target_out_of_firing_range;
          else if (AShip.GetIsDammaged())
             MostRecentInfraction = dammaged_ship_cannot_open_fire;
          else
             MostRecentInfraction=cant_open_fire;
       }
       return TShip::cannot_open_fire;
}

//*****************************************************************************

void TReferee::StartANewRound()
{
   vector<TShip*>::iterator Begin=ListOfAlliedShips.begin();


//************************
   //temporary disable to allow testing
   //for_each(ListOfEnemyShips.begin(), ListOfEnemyShips.end(), TShip::RandomizeS
```

```
hipPosition);
//*****************************

   //We now place all players' ship to theirs starting position.
   //We cannot use for_each because the required function signature is
   //not correct

   unsigned i=0;

   ListOfEnemyShips=TheScenario.GetListOfEnemyShips();
   dynamic_cast <TSubmarine*>(ListOfEnemyShips[0])->Practice(TCoordinate(2,9,1));
//Tempo

   //Remove preceeding line after testing

   const TInitialPositions & InitialPositions=TheScenario.GetInitialPositions();

   for (vector<TShip*>::iterator it=Begin; it != ListOfAlliedShips.end(); it++)
   {
      (*it)->SetCurrentPosition(InitialPositions[i++]);
      (*it)->Initialize();
      (*it)->ComputeTargetRange();
   }

   ActiveShip = Begin;
   ActivePlayer = (*ActiveShip)->GetCommandingOfficer();
   CurrentRoundWinner = NULL;
   DescriptionOfTheInfraction="";
   MostRecentInfraction=none;



};

//*****************************************************************************

void TReferee::IsThereACollisionWithAnotherObject(const TCoordinate &Position,
                                                  const TShip &AShip,
                                                  const TShip *& VictimOfTheCollisi
on, //The "innocent" ship that will get hit
                                                  bool &ThereIsACollision)
{
   vector<TShip*>::const_iterator it, End=ListOfAlliedShips.end();

   for (it = ListOfAlliedShips.begin(); it != End; it++)
   {
      if (**it != AShip) //Check for self-collision
      {
         if ((*it)->GetCurrentPosition() == Position)
         {
            ThereIsACollision=true;
            VictimOfTheCollision=*it;
            return;
         }
      }
   }

   VictimOfTheCollision=NULL;
   ThereIsACollision=false;
}
```

```
//*****************************************************************************

TReferee::~TReferee()
{
   NumberOfActiveInstance--;
};

//*****************************************************************************

bool TReferee::WasThereAnInfraction()
{
   if(MostRecentInfraction == none)
      return false;
   else
   {
      strstream Message;
      char *Tempo; //to get the content of the strstream

      switch (MostRecentInfraction)
      {


         case collision:
         {
            const TShip * const OtherShip= CollisionInfo.GetOtherShipImpliedInThe
Collision();
            if (OtherShip != NULL)
            {
               Message<<"WARNING collision with     ship "<<OtherShip->GetID()<<
               " located at "<<CollisionInfo.GetPositionOfTheCollision();
            }
            else //Collision with static ocean features such as islands and reefs
            {
               Message<<"WARNING attempt to cross a non naviguable ocean"
                      <<" position at "<<CollisionInfo.GetPositionOfTheCollision(
)
                      <<" the is a "<<CollisionInfo.GetContentOfTheOceanAtTheSite
OfTheCollision()<<
                      "there.";
            }
            break;

         }
         case run_aground:
         {
            Message<<"WARNING your ship has run  aground.";
            break;
         }
         case target_out_of_firing_range:
         {
            Message<<"WARNING: there is no target within firing range.";
            break;
         }

         case dammaged_ship_cannot_open_fire:
         {
            Message<<"WARNING: your ship is dammaged, it cannot open fire.";
            break;
         }
         case cant_open_fire:
         {
```

```
                Message<<"You cannot open fire.";
                break;
            }

            case your_are_hit_by_ennemy_fire:
            {
                Message<<"DANGER: you have been hit by an ennemy torpedoe, "<<
                "your ship is dammaged";
                break;
            }

            case movement_in_violation_of_ship_specifications:
            {
                Message<<"Caution: your ship has limitations";
                Message<<"Maximum turning rate="<<
                (*ActiveShip)->GetCurrentTurningRate();
                Message<<"    Maximum acceleration rate="<<
                          (*ActiveShip)->GetCurrentAccelerationRate();
            }
            default :
                throw TInvalidInfractionType();
        } //end case

        //str() returns a char*, not a string
        //Since it is our reponsability to delete it, we must store it
        //ErrorMessage=Message.str() would be legal be would have a memory leak

        Message<<ends; //Add the required \0
        Tempo=Message.str();
        DescriptionOfTheInfraction=string(Tempo);
        delete [] Tempo;
        MostRecentInfraction=none; //Reset infraction flag
        return true;
    }
}

//***************************************************************************

void TReferee::AbandonShip(const TShip * const AShip)
{
    vector<TShip*>::iterator End=ListOfAlliedShips.end(),
                             Begin=ListOfAlliedShips.begin();

    vector<TShip*>::iterator it =find(Begin,End,AShip);

    if (it != End)
    {
        //After the deletion, ActivePlayer will no longer be pointing to
        //the active player, we must compensate. There might even be no longer
        //any active players!

        ActiveShip=End;

        if (ListOfAlliedShips.size() > 1) //There are some ships left
        {
            if (it != Begin)
            {
                ActiveShip= it-1;
                ListOfAlliedShips.erase(it);
            }
```

```
            else //Current ship is the first one in the array
                 //The
            {
               it=ListOfAlliedShips.end();
               ActiveShip=--it;
            }
         }
         else //All allied ships are sunk or abandonned, the round is over
            ActiveShip=NULL;
      }
         //else throw TThisShipIsNotUnderTheControlOfTheReferee(AShip);

}

//***************************************************************************

void TReferee::Penalize(TShip &TheShipToPenalize)
{
   TheShipToPenalize.SetIsDammaged(true);
}

void TReferee::MoveTheEnemies()
{
   vector<TShip*>::iterator it;

   for (it=ListOfEnemyShips.begin(); it < ListOfEnemyShips.end(); it++)
   {
      TShip * ActiveShip= *it;
      ActiveShip->MoveOnTheOcean(ActiveShip->GetCurrentHeading(),1); //tempo
   }

}

void TReferee::PerformEvasiveManeuvers(TSubmarine &EvadingSubmarine)
{
   int i=random(2);
   switch (i)
   {
      case 0:
      {
         EvadingSubmarine.
                  SetCurrentHeading(EvadingSubmarine.GetCurrentHeading()+45);
         break;
      }

      case 1:
      {
         unsigned CurrentHeading=EvadingSubmarine.GetCurrentHeading();
         if (CurrentHeading != 0)
            CurrentHeading=(CurrentHeading+45)%360;
         else
            CurrentHeading=315;

         EvadingSubmarine.SetCurrentHeading(CurrentHeading);
         break;

      }
      default : ;
         //Do nothing
   }
}
```

```
} //End namespace
```

```
#ifndef PROMOTIONPATH_H

#include <vector>
#include "Rank.h"
#include "SubExceptions.h"


namespace SUB_Framework
{

//*****************************************************************************
//*****************************************************************************

class TRankDoesNotExist : public TInitialisationError
{
   public:
      explicit TRankDoesNotExist (const TRank &ABadRank );
      virtual void What() const;
   private:
      TRank BadRank;
};

//*****************************************************************************
//*****************************************************************************

class TRankIndexIsOutOfRange : public TInitialisationError
{
   public:
      explicit TRankIndexIsOutOfRange (const unsigned &ABadRankIndex );
      virtual void What() const;
   private:
      unsigned BadRankIndex ;
};

//*****************************************************************************
//*****************************************************************************

class TPromotionPathIsEmpty : public TInitialisationError
{
   public:
      virtual void What() const;
};

//*****************************************************************************
//*****************************************************************************

class TPromotionPath
{
   public:
      static AddANewRank (const TRank& ARank);
      static TRank GetNextRank (const TRank &ARank);
      static TRank GetNextRank (const unsigned ARankIndex );
      static const TRank& GetHighestRank ();
      static TRank ConvertRankIndexToRank (const unsigned ARankIndex );

   private:
      TPromotionPath(); //Dissallow creation of instances
      static vector<TRank> PromotionPath ;
};

//*****************************************************************************
```

```
inline TPromotionPath ::AddANewRank (const TRank & ARank)
{
   PromotionPath.push_back(ARank);
}
//*****************************************************************************
//*****************************************************************************

} //End namespace

#define PROMOTIONPATH_H
#endif
```

```
#include <algorithm>
#include <strstream>
#include "PromotionPath.h"


namespace SUB_Framework
{

//*****************************************************************************
//*****************************************************************************

using namespace std;

//*****************************************************************************
//*****************************************************************************

TRankDoesNotExist::TRankDoesNotExist(const TRank &ABadRank):BadRank(ABadRank)
{
}

//*****************************************************************************

void TRankDoesNotExist::What() const
{
   strstream ErrStream(Message,sizeof(Message),ios::out);
   ErrStream<<"Initialisation error: "<<
   "the rank \""<<BadRank<<"\" does not exist"
            <<ends;

}

//*****************************************************************************
//*****************************************************************************

TRankIndexIsOutOfRange::TRankIndexIsOutOfRange(const unsigned &ABadRankIndex):
                                                BadRankIndex(ABadRankIndex)
{
}

//*****************************************************************************

void TRankIndexIsOutOfRange::What() const
{
   strstream ErrStream(Message,sizeof(Message),ios::out);
   ErrStream<<"Initialisation error: "<<
   "the rank index O"<<BadRankIndex+4<<" does not exist in this game"<<ends;

}

//*****************************************************************************
//*****************************************************************************

void TPromotionPathIsEmpty::What() const
{
   strstream ErrStream(Message,sizeof(Message),ios::out);
   ErrStream<<"Initialisation error: the rank hierachy isn't defined."<<ends;
}



//*****************************************************************************
```

```
//***************************************************************************

vector<TRank> TPromotionPath::PromotionPath; //Static variable definition

//***************************************************************************
//***************************************************************************

TRank TPromotionPath::GetNextRank(const TRank &ARank)
{
   vector<TRank>::iterator it=PromotionPath.begin();
   it=find(it,PromotionPath.end(),ARank);

   if (it != PromotionPath.end())
   {
      if (++it != PromotionPath.end())
         return *it;
      else
         return ARank; //return the given rank if no superior one exist
   }
   else
      throw TRankDoesNotExist(ARank);
}

//***************************************************************************

TRank TPromotionPath::GetNextRank(const unsigned ARankIndex)
{
   if (ARankIndex < PromotionPath.size())
   {
      if (ARankIndex != PromotionPath.size()-1)
         return PromotionPath[ARankIndex+1];
      else
         return PromotionPath[ARankIndex];
   }
   else
      throw TRankIndexIsOutOfRange(ARankIndex);
}

//***************************************************************************

TRank TPromotionPath::ConvertRankIndexToRank(const unsigned ARankIndex)
{
   if (ARankIndex < PromotionPath.size())
      return PromotionPath[ARankIndex];
   else
      throw TRankIndexIsOutOfRange(ARankIndex);
}

//***************************************************************************

const TRank& TPromotionPath::GetHighestRank()
{
   if (!PromotionPath.empty())
      return *(PromotionPath.end()-1);
   else
      throw TPromotionPathIsEmpty();
}

//***************************************************************************
//***************************************************************************
```

```
}//End namespace
```

```
#ifndef BASICGAMESHIPID_H

namespace  SUB_Framework
{

const unsigned  TargetSubmarineID  =0;

} //End namespace

#define BASICGAMESHIPID_H
#endif
```

```
#ifndef SUBMARINE_H

#include "Ship.h"

namespace SUB_Framework
{

//*****************************************************************************
//*****************************************************************************

class TSubmarine: public TShip
{

   public:

      TSubmarine(TPlayer *ACommandingOfficer=&DefaultCommandingOfficer,
            const string &ShipName="",
            const unsigned Longitude=0, const unsigned Latitude=0,
            const TShipAllignment AnAllignment=ennemy,
            const unsigned Depth=1, const unsigned MaxSpeed=1,
            const unsigned MaxTurn=180 );

      virtual void MoveOnTheOcean(unsigned NewHeading, unsigned NewSpeed) const;
      virtual void RandomizePosition();
      void Practice(const TCoordinate &TargetPosition); //Place a practice target
 at a preset position
      void PerformEvasiveManeuvers() const;

      static TPlayer GetDefaultCommandingOfficer();

   protected:

   private:
      static TPlayer DefaultCommandingOfficer;
}; //End class TSubmarine

//*****************************************************************************

TPlayer TSubmarine::GetDefaultCommandingOfficer()
{
   return DefaultCommandingOfficer;
}

//*****************************************************************************

void TSubmarine::Practice(const TCoordinate& TargetPosition)
{
   SetCurrentPosition(TargetPosition);
   SetCurrentHeading(0);
}

//*****************************************************************************
//*****************************************************************************

} //End namespace

#define SUBMARINE_H
#endif
```

```
#include "Submarine.h"
#include "OceRep.h"
#include "BasicGameShipID.h"
#include "Referee.h"

namespace SUB_Framework
{
//*****************************************************************************
//*****************************************************************************

//Static date member definition

TPlayer TSubmarine::DefaultCommandingOfficer("Default ennemy");

//*****************************************************************************
//*****************************************************************************

TSubmarine::TSubmarine(TPlayer *ACommandingOfficer,
            const string &ShipName,
            const unsigned Longitude, const unsigned Latitude,
            const TShipAllignment AnAllignment,
            const unsigned Depth, const unsigned MaxSpeed,
            const unsigned MaxTurn):
            TShip(ACommandingOfficer,TShip::submarine, ShipName,
                  Longitude,Latitude,AnAllignment,Depth,MaxSpeed,MaxTurn)
{
};

void TSubmarine::MoveOnTheOcean(unsigned NewHeading, unsigned NewSpeed) const
{
   //TShip::MoveOnTheOcean(NewHeading,NewSpeed);
   const_cast <TReferee*>(Referee)->MoveEnemyOnTheOcean(*const_cast <TSubmarine*>(t
his),NewHeading,NewSpeed);
   //Should be moved to referee

   /*if (CurrentPosition.GetWhatIsAtCoordinate() != water) //Something is blockin
g
                                                         //the sub path.
   {

      //Reverse course, put the submarine back on the sea

      unsigned ReverseCourseHeading=(CurrentHeading+180)%361;
      TShip::MoveOnTheOcean(ReverseCourseHeading,CurrentSpeed);

      //Compute a random change of heading, the following algorithm is
      //good only for the basic game.

      int RandomChangeOfHeading=random(3);
      int DeltaHeading=0;

      if (RandomChangeOfHeading==0)
         DeltaHeading=45;
      else if (RandomChangeOfHeading==1)
         DeltaHeading=-45;

         CurrentHeading=(ReverseCourseHeading+DeltaHeading)%361;

   } */
```

```
};

void TSubmarine::RandomizePosition()
{
   //The following default randomize function relies on the fact that more then
   //90% of the ocean is water. If this is not the case for a scenario, you
   //should create a new subclass of TSubmarine and redifine this virtual
   //function.

   //WARNING: this function could run almost FOREVER if there is more land then
   //water in the game ocean.

   //The basic game submarines are never on the surface (depth 0).


   do
      CurrentPosition.Randomize(true); //Always return a non-zero depth
   while (TheOcean->GetWhatIsAtCoordinate(CurrentPosition) != water);


}

void TSubmarine::PerformEvasiveManeuvers() const
{
   const_cast <TReferee*>(Referee)->PerformEvasiveManeuvers(*const_cast <TSubmarine
*>(this));
}
//***************************************************************************
//***************************************************************************

} //End Namespace
```

```
#ifndef OBJECTIVE_H

#include <vector>
#include "SubExceptions.h"

namespace SUB_Framework
{

using namespace std;

// This class store a collection of objectives for both the player and the
// computer. An objective is the unique identifier of a ship that need to be
// sink in order to win.

// When all target ships in the collection
// have been sunked by one side, it has won this round.

// If the objective vector is empty, this means that this side cannot win.
// It is an error to leave the player's objective list empty. This is why
// the only constructor for this class requires a pointer to a ship that
// is the first player objective.

// ***************************************************************************
//***************************************************************************

class TShip;

//***************************************************************************
//***************************************************************************

class TThereCanBeOnlyOneListOfObjectivesActiveAtAnyTime: public TLogicError
{
   public:
      virtual void What() const;


};

class TThereMustBeAtLeastOnePlayerObjective : public TInitialisationError
{
   virtual void What() const;
};

//***************************************************************************
//***************************************************************************

class TObjectives
{
   public:
      explicit TObjectives(TShip *AnObjective);
      void AddAPlayerObjective  (TShip *AShip); //Any ship can be a target, not j
ust sub
      void AddAComputerObjective(TShip *AShip);
      bool NoMorePlayersObjectives() const;
      bool NoMoreComputerObjectives() const;
      bool CheckIntegrity() const;

      virtual ~TObjectives();

   private:
      static unsigned NumberOfActiveInstance=0;
```

```
        vector<TShip*> PlayersObjectives;
        vector<TShip*> ComputerObjectives;

        void AddAnObjective(vector<TShip*> &AVector, TShip* AShip);

};

//***************************************************************************

inline bool TObjectives::NoMorePlayersObjectives() const
{
   return PlayersObjectives.empty();
}

//***************************************************************************

inline bool TObjectives::NoMoreComputerObjectives() const
{
   return ComputerObjectives.empty();
}

//***************************************************************************

inline TObjectives::~TObjectives()
{
   NumberOfActiveInstance--;
}

//***************************************************************************
//***************************************************************************

} //End Namespace

#define OBJECTIVE_H
#endif
```

```
#include <strstream>
#include "Objective.h"
#include "BasicGameShipID.h"
#include "Ship.h"
#include "AddToTheVector.h"

namespace  SUB_Framework
{

//*********************************************************************************
//*********************************************************************************

void TThereCanBeOnlyOneListOfObjectivesActiveAtAnyTime::What() const
{
   strstream ErrorMessage(Message,sizeof (Message),ios::out);
   ErrorMessage<<" Logic error: there can only be one list of objectives actives
at any time."<<ends;


}

//*********************************************************************************

void TThereMustBeAtLeastOnePlayerObjective::What() const
{
   strstream ErrorMessage(Message,sizeof (Message),ios::out);
   ErrorMessage<<" Initialisation error: the players must have at least one objec
tive."<<ends;
}

//*********************************************************************************
//*********************************************************************************

TObjectives::TObjectives(TShip *AnObjective)
{
   if (NumberOfActiveInstance == 0)
   {
      NumberOfActiveInstance++;
      if (AnObjective != NULL)
         PlayersObjectives.push_back(AnObjective);
      else throw TThereMustBeAtLeastOnePlayerObjective();
   }
   else
      throw TThereCanBeOnlyOneListOfObjectivesActiveAtAnyTime();

}

//*********************************************************************************

void TObjectives::AddAnObjective(vector<TShip*> &AVector, TShip* AShip)
{
   AddToTheVectorIfNotAlreadyThere<TShip>(AVector,AShip);
}

//*********************************************************************************

void TObjectives::AddAPlayerObjective  (TShip *AShip)
{
   AddAnObjective(PlayersObjectives,AShip);
}
```

```
//*****************************************************************************

void TObjectives::AddAComputerObjective(TShip *AShip)
{
   AddAnObjective(ComputerObjectives,AShip);
}

//*****************************************************************************

bool TObjectives::CheckIntegrity() const //There must be some player's objectives
{
   return !PlayersObjectives.empty();
}

//*****************************************************************************
//*****************************************************************************

} //End namespace
```

```
#include <vector>


//template <class T> AddToTheVector(vector<T*> &AVector, T* AnElement);


//Preconditions

//    This function works solely on vector made of POINTERS to type T.
//    Type T must have a publicly accessible operator ==


//Postcondition

//    If the VALUE POINTED BY AnElement is NOT already present in the vector,
//    AnElement is added at the end of the vector, if it is, nothing is done.

//Worst case complexity O(AVector.size())

template <class T> void AddToTheVectorIfNotAlreadyThere (vector<T*> &AVector, T* A
nElement)
{
   vector<T*>::const_iterator it=AVector.begin();
   const vector<T*>::const_iterator EndOfTheVector =AVector.end();

   bool NotFound =true;

   while (it!=EndOfTheVector && NotFound)
   {
      if (**it == *AnElement) //compare the elements, not the pointers
         NotFound=false;
      else
         it++;
   }

   if (NotFound)
      AVector.push_back(AnElement);
};
```

```
#ifndef RULESOFENGAGEMENT_H

#include "Ship.h"

namespace SUB_Framework
{


// This class defines a function that is used to tell if a certain target
// can be attaked by an attacking ship.

// The function is virtual, there is no member data as valid targets
// changes for each different game that can be derived from the framework.

// The default behavior is that only ennemy targets can be fired uppon.
// Each player may have different rules of engagement and the ennemy usually
// uses its own rules of engagement

//*****************************************************************************

class TRulesOfEngagement
{
   public:
      virtual bool IsValidTarget(const TShip &Attacker, const TShip &Target)const
;
};



} //End namespace

//*****************************************************************************

#define RULESOFENGAGEMENT_H
#endif
```

```
#include "RulesOfEngagement.h"

namespace SUB_Framework
{
//*****************************************************************************
//*****************************************************************************

bool TRulesOfEngagement::IsValidTarget(const TShip &Attacker,
                                       const TShip &Target)const
{
   TShip::TShipAllignment AttackerAllignment=Attacker.GetAllignment();
   TShip::TShipAllignment TargetAllignment=Target.GetAllignment();

   if (AttackerAllignment != TargetAllignment)
   {
      if (AttackerAllignment==TShip::allied && TargetAllignment==TShip::ennemy ||
          AttackerAllignment==TShip::ennemy && TargetAllignment==TShip::allied)

         return true; //Ships are not on the same side, it is OK to attack
   }

   return false; //Do not open fire on friendly or neutral ships.

}

}

//*****************************************************************************
//*****************************************************************************
```

```
#ifndef SHIPEXCEPTIONS_H

#include "SubExceptions.h"

namespace  SUB_Framework
{

class  TShipDoesNotExist  : public  TLogicError
{
   public :

      TShipDoesNotExist(const  unsigned  AShipNumber );
      virtual  void  What() const ;
      virtual  ~TShipDoesNotExist  ();

   private :
      unsigned  ShipNumber ;
};

} //End namespace
#define SHIPEXCEPTIONS_H
#endif
```

```
#include <strstream>

#include "ShipExceptions.h"

namespace  SUB_Framework
{

//********************************************************************************
//********************************************************************************

TShipDoesNotExist::TShipDoesNotExist(const unsigned  AShipNumber ) :
                    ShipNumber(AShipNumber)
{
}

//********************************************************************************

void  TShipDoesNotExist  ::What () const
{
   strstream ErrStream(Message,sizeof (Message ),ios::out);
   ErrStream<<"Logic Error: the ship "<<ShipNumber<<" does not exists\0";

}

//********************************************************************************

TShipDoesNotExist::~TShipDoesNotExist()
{
}

//********************************************************************************
//********************************************************************************

} //End namespace
```

```
#ifndef INITIALPOSITION_H

#include <vector>
#include "SubExceptions.h"
#include "OceRep.h"
#include "Coordinate.h"
#include "ShipExceptions.h"

//This class is used to store the ships position at he beginning of the
//game. Because there must be exactly one list of initial position, all
//members will be static. For each new starting position, we must ensure
//that there is not already another ship at the same coordinate and that
//all ships starts in water or shallow water ocean intersection.

namespace SUB_Framework
{

//*****************************************************************************
//*****************************************************************************

class TTwoShipsAtTheSamePosition: public TInitialisationError
{
   public:

      TTwoShipsAtTheSamePosition(const TCoordinate &ABadCoordinate);
      virtual void What() const;
      virtual ~TTwoShipsAtTheSamePosition();

   private:
      TCoordinate BadCoordinates;
};

//*****************************************************************************
//*****************************************************************************

class TShipNotInTheWater: public TInitialisationError
{
   public:

      TShipNotInTheWater(const TCoordinate &ABadCoordinate,
                         const TOceanContent AnOceanContent);
      virtual void What() const;
      virtual ~TShipNotInTheWater();

   private:
      TCoordinate BadCoordinates;
      TOceanContent OceanContent;
};

//*****************************************************************************
//*****************************************************************************

using namespace std;


// The position are stored here because there will be many rounds
// from the same starting position before a winner can be found.

//The position in the vector must match an existing ship ID number
// This class can throw many exception
```

```
//    TTwoShipsAtTheSamePosition if two ships starts from the same position

//    TShipNotInTheWater if a ship doesn't start on a safe (non mined) water or
//                       or shallow water position.

// TShipDoesNotExist if there are more starting coordinates then ships

//IMPORTANT NOTE: Ships must be defined before they can have an initial position.


class TInitialPositions
{
   public:

      //The return type of the size member function of the STL

      typedef vector<TCoordinate>::size_type size_type;

      static void AddANewShip(const TCoordinate& StartingCoordinates);
      static size_type GetNumberOfShipsInPlay();

      TCoordinate& operator[](const int i) const;

   private:
      TinitialPositions(); //No definition so instance cannot be created
      static vector<TCoordinate> AlliedShipsStartingPosition;
};

//*****************************************************************************

inline vector<TCoordinate>::size_type
       TInitialPositions::GetNumberOfShipsInPlay()
{
   return AlliedShipsStartingPosition.size();
}

//*****************************************************************************

inline TCoordinate& TInitialPositions::operator[](const int i) const
{

   if (i >=0 && i < AlliedShipsStartingPosition.size())
      return AlliedShipsStartingPosition[i];
   else
      throw TShipDoesNotExist(i);
}

//*****************************************************************************
//*****************************************************************************

} //End Namespace
#define INITIALPOSITION_H
#endif
```

```
#include <strstream>
#include "InitialPosition.h"
#include "Ship.h"
#include "ShipExceptions.h"


namespace SUB_Framework
{

//#########################################################################
//#########################################################################

TTwoShipsAtTheSamePosition::
   TTwoShipsAtTheSamePosition(const TCoordinate &ABadCoordinate):
   BadCoordinates(ABadCoordinate)
{
}

//#########################################################################

void TTwoShipsAtTheSamePosition::What() const
{
   strstream ErrMessage(Message,sizeof(Message),ios::out);
   ErrMessage<<"Initialisation error: two ships occupies the same coordinates : "
<<
               BadCoordinates<<".\0";


}

//#########################################################################

TTwoShipsAtTheSamePosition::~TTwoShipsAtTheSamePosition()
{
}

//#########################################################################
//#########################################################################

TShipNotInTheWater::
   TShipNotInTheWater(const TCoordinate &ABadCoordinate,
                      const TOceanContent AnOceanContent):
   BadCoordinates(ABadCoordinate),OceanContent(AnOceanContent)
{
}

//#########################################################################

void TShipNotInTheWater::What() const
{
   strstream ErrMessage(Message,sizeof(Message),ios::out);
   ErrMessage<<"Initialisation error: a ship at coordinates : "<<
               BadCoordinates<<" occupies a non water ("<<OceanContent<<
               ") position.\0";


}

//#########################################################################

TShipNotInTheWater::~TShipNotInTheWater()
```

```
{
}

//*****************************************************************************
//*****************************************************************************

vector<TCoordinate> TInitialPositions::AlliedShipsStartingPosition;
void TInitialPositions::AddANewShip(const TCoordinate &StartingCoordinates)
{

   vector<TCoordinate>::iterator i=AlliedShipsStartingPosition.begin();

   i=find(i,AlliedShipsStartingPosition.end(), StartingCoordinates);

   if (i==AlliedShipsStartingPosition.end()) //End of the vector reached
                                             //without finding the coordinates
   {
      unsigned NextShipNumber=AlliedShipsStartingPosition.size();
      if (NextShipNumber < TShip::GetShipNumber())
         if (TShip::GetTheOcean()->IsNavigable(StartingCoordinates))
            AlliedShipsStartingPosition.push_back(StartingCoordinates);
         else
            throw TShipNotInTheWater(StartingCoordinates,
                     TShip::GetTheOcean()->
                     GetWhatIsAtCoordinate(StartingCoordinates));
      else
         throw TShipDoesNotExist(NextShipNumber);
   }
   else
      throw TTwoShipsAtTheSamePosition(StartingCoordinates);

};


} //End namespace
```

```
#include "Ship.h"

namespace  SUB_Framework
{

//*************************************************************************
//*************************************************************************

TShip::TShipInfo::TShipInfo(const TShipType AShipType,
                const unsigned MaxSpeed,
                const unsigned MaxTurn,
                const unsigned MaxAccel,
                const unsigned WRange) :


                MaximumTurningRate(MaxTurn),
                MaximumSpeed(MaxSpeed),
                MaximumAccelerationRate(MaxAccel),
                ShipType(AShipType),WeaponRange(WRange)

{
   if (WeaponRange > 2)
      WeaponRange=2;
}

//*************************************************************************
//*************************************************************************

} //End namespace
```

```
#ifndef SHIP_H

#include <string>
#include <vector>
#include <set>

#include "Ocean.h"
#include "Player.h"
#include "BasicOceRep.h"
#include "AimingInfo.h"


namespace  SUB_Framework
{

//*****************************************************************************
//*****************************************************************************

   using namespace  std;

//*****************************************************************************
//*****************************************************************************
   class  TReferee;

   class  TShipInfoIsLess; //Ordering for insertion into the set

//*****************************************************************************
//*****************************************************************************

   class  TThereMustBeAReferee : public  TInitialisationError
   {
      private :
         virtual  void  What() const ;
   };

//*****************************************************************************
//*****************************************************************************

   class  TInvalidShipType: public  TLogicError
   {
      private :
         virtual  void  What() const ;
   };

//*****************************************************************************
//*****************************************************************************

   class  TAShipMustHaveACommandingOfficer: public  TInitialisationError
   {
      private :
         virtual  void  What() const ;
   };

//*****************************************************************************
//*****************************************************************************

   class  TShipTypeIsntRegistered: public  TInitialisationError
   {
      private :
         virtual  void  What() const ;
```

```
   };

//***************************************************************************
//***************************************************************************

   class TShip
   {
      friend TReferee;
      friend ostream& operator <<(ostream& os, const TShip& AShip);

      public :
         typedef enum {pt_boat, destroyer, cruiser, battleship,submarine} TShipTy
pe;
         typedef enum {allied, ennemy, neutral} TShipAllignment;
         typedef enum {sunk, off_1, off_2, sos, cannot_open_fire} TFiringResult;

         static unsigned GetDefaultMaximumSpeed();
         static unsigned GetDefaultWeaponRange();

//***************************************************************************
//***************************************************************************

         class TShipInfo
         {
            friend TShip;
            public :
               TShipInfo::TShipInfo(const TShipType AShipType,
               const unsigned MaxSpeed=TShip::GetDefaultMaximumSpeed(),
               const unsigned MaxTurn=180,
               const unsigned MaxAccel=TShip::GetDefaultMaximumSpeed(),
               const unsigned WRange=TShip::GetDefaultWeaponRange() );

               TShipType GetShipType() const;

            private :

               unsigned MaximumSpeed,MaximumTurningRate,MaximumAccelerationRate;
               TShipType ShipType;
               unsigned WeaponRange;
         };

//***************************************************************************
//***************************************************************************

         TShip(const TShipType AShipType,
            TPlayer *CommandingOfficer,const string &ShipName="",
            const unsigned Longitude=0, const unsigned Latitude=0,
            const TShipAllignment AnAllignment=allied,
            const unsigned Depth=0);
      protected :
         TShip(TPlayer *CommandingOfficer,
            const TShipType AShipType=pt_boat, const string &ShipName="",
            const unsigned Longitude=0, const unsigned Latitude=0,
            const TShipAllignment AnAllignment=allied,
            const unsigned Depth=0, const unsigned MaxSpeed=DefaultMaximumSpeed,
            const unsigned MaxTurn=180,
            const unsigned MaxAccel=DefaultMaximumSpeed,
            const unsigned WRange=2 );
      public :
         static unsigned GetShipNumber();
         static const TOcean * GetTheOcean();
```

```
        static void SetTheOcean(TOcean *AnOcean);

    // The sort function requires a static or non-member function that can be used
    // to compare two pointers to ship. Quite obviously, we do not want to compare
    // the adresses since the result of the comparison would be unspecified.

    //*************************************************************************
    //*************************************************************************

        static bool ComparePointersToShip(const TShip * const Ship1, const TShip
* const Ship2)
        {
           return *Ship1 == *Ship2; //Compare the ships, not the pointers
        };

    //*************************************************************************
    //*************************************************************************

        TShipAllignment GetAllignment() const;

        TCoordinate GetCurrentPosition() const;
        unsigned GetID()const;

        unsigned GetCurrentSpeed() const;
        unsigned GetCurrentHeading() const;
        unsigned GetCurrentTurningRate() const;
        unsigned GetMaxSpeed() const;
        unsigned GetMaxTurningRate() const;
        const unsigned GetWeaponRange() const;
        TShipType GetShipType() const;

        bool GetCanOpenFire() const;
        unsigned GetCurrentAccelerationRate() const;
        const TPlayer* GetCommandingOfficer() const;
        bool GetIsDammaged() const;

        virtual void MoveOnTheOcean(unsigned NewHeading, unsigned NewSpeed) cons
t;
        virtual TFiringResult OpenFire(const TAimingInformation &AimingInfo);
        virtual void DoRepairs();
        static string ConvertHeading(const unsigned AnHeading);

        virtual void ComputeTargetRange();
        virtual unsigned GetTargetRange();

        bool operator == (const TShip& AShip) const;

        //For inclusion in a set structure

        bool operator < (const TShip& AShip) const;

        static void SetReferee(const TReferee * AReferee);
        static const TReferee* GetReferee();
        static bool RegisterANewShipType(const TShipType AShipType,
                                         const unsigned MaxSpeed=DefaultMaximumSpeed,
                                         const unsigned MaxTurn=180,
                                         const unsigned MaxAccel=DefaultMaximumSpeed,
                                         const unsigned WRange=2 );

        virtual ~TShip();
```

```
    TShip& operator =(const TShipInfo &RHS);

  private :
    TPlayer * const CommandingOfficer;
                                  //A reference to the player that will
                                  //move this ship.

                                  //Ennemy ships needs a commanding
                                  //officer even if it is the computer that
                                  //will move it.

    static unsigned ShipNumber; //The ID of the next ship that will
                                  //be constructed. Those IDs are
                                  //created sequentially starting with 0.


    unsigned ID;
    unsigned MaximumSpeed, MaximumTurningRate, MaximumAccelerationRate;
    string Name; // A name such as HMCS Huron, USS Entreprise

    //This function is define to be compatible with the for_each function of
    //the standard C++ library. It will be used to ramdomize the position of
    //a collection of ships. This function cannot be a non-static member.

    static void RandomizeShipPosition(TShip *Aship);


  protected :

    void Initialize();

    unsigned CurrentSpeed, CurrentMaximumSpeed,
             CurrentMaximumTurningRate,CurrentMaximumAccelerationRate,
             AimingDirection;
    //Damaged ships current max speed and turning rate may be less.

    TCoordinate CurrentPosition;

    bool CanOpenFire;
    bool IsDammaged;
    TShipType ShipType;
    const unsigned WeaponRange;
    TShipAllignment Allignment;
    unsigned TargetRange, CurrentHeading;
    void SetTargetRange(const unsigned NewTargetRange);
    void SetCanOpenFire (const bool CanFire);

    virtual void RandomizePosition();
    virtual void AbandonShip();
    //static TPlayer SampleEnnemyCommander;

//There is only one referee for all ships and subclass of ships

    static const TReferee * Referee;
    static set<TShipInfo,TShipInfoIsLess> ListOfRegisteredShipTypes;

    void SetCurrentPosition(const TCoordinate &NewCoordinates);
    void SetIsDammaged(bool NewValue);
    void SetCurrentHeading(const unsigned NewHeading);
    void SetCurrentSpeed(const unsigned NewSpeed);
```

```
        bool Improve(); //Get the next bigger ship, if any

        static TShipType TheBiggestShip;
        static TOcean *TheOcean; // The ocean the ships are moving on.
                                 // All sips shares the same ocean.

        //vector<int> ListOfWeapons, ListOfSensors;
        static unsigned DefaultMaximumSpeed;
        static unsigned DefaultWeaponRange;
    };

//*****************************************************************************

    inline unsigned TShip::GetShipNumber()
    {
       return ShipNumber;
    }

//*****************************************************************************

    inline TShip::TShipAllignment TShip::GetAllignment() const
    {
       return Allignment;
    }

//*****************************************************************************

    inline TCoordinate TShip::GetCurrentPosition() const
    {
       return CurrentPosition;
    }

//*****************************************************************************
    inline unsigned TShip::GetID()const
    {
       return ID;
    }

//*****************************************************************************

    inline unsigned TShip::GetCurrentSpeed() const
    {
       return CurrentSpeed;
    }

//*****************************************************************************

    inline unsigned TShip::GetCurrentHeading() const
    {
       return CurrentHeading;
    }

//*****************************************************************************

    inline TShip::TShipType TShip::GetShipType() const
    {
       return ShipType;
    }

//*****************************************************************************
```

```
   inline unsigned TShip::GetCurrentTurningRate() const
   {
      return CurrentMaximumTurningRate;
   }

//*****************************************************************************

   inline unsigned TShip::GetMaxSpeed() const
   {
      return CurrentMaximumSpeed;
   }

//*****************************************************************************

   inline unsigned TShip::GetMaxTurningRate() const
   {
      return CurrentMaximumTurningRate;
   }

//*****************************************************************************

   inline const unsigned TShip::GetWeaponRange() const
   {
      return WeaponRange;
   }

//*****************************************************************************

   inline bool TShip::GetCanOpenFire() const
   {
      return CanOpenFire;
   }

//*****************************************************************************

   inline unsigned TShip::GetCurrentAccelerationRate() const
   {
      return CurrentMaximumAccelerationRate;
   }

//*****************************************************************************

   inline const TPlayer* TShip::GetCommandingOfficer() const
   {
      return CommandingOfficer;
   }

//*****************************************************************************

   inline bool TShip::GetIsDammaged() const
   {
      return IsDammaged;
   }

//*****************************************************************************

   inline bool TShip::operator == (const TShip& AShip) const
   {
      return ID==AShip.ID;
   }
```

```
//*****************************************************************************

   inline bool TShip::operator < (const TShip& AShip) const
   {
      return static_cast <int>(ShipType) < static_cast <int>(AShip.ShipType);
   }

//*****************************************************************************

   inline void TShip::SetTheOcean(TOcean *AnOcean)
   {
      TheOcean = AnOcean;
   }

//*****************************************************************************

   inline const TOcean * TShip::GetTheOcean()
   {
      return TheOcean;
   }

//*****************************************************************************

   inline void TShip::SetIsDammaged(bool NewValue)
   {
      IsDammaged=NewValue;
   }

//*****************************************************************************

   void TShip::SetTargetRange(const unsigned NewTargetRange)
   {
      TargetRange=NewTargetRange;
   }

//*****************************************************************************

   inline void TShip::SetCurrentHeading(const unsigned NewHeading)
   {
      CurrentHeading=NewHeading;
   }

//*****************************************************************************

   inline void TShip::SetCurrentSpeed(const unsigned NewSpeed)
   {
      if (NewSpeed <= CurrentMaximumSpeed)
         CurrentSpeed=NewSpeed;
   }

//*****************************************************************************

   inline const TReferee* TShip::GetReferee()
   {
      return Referee;
   }

//*****************************************************************************

   inline void TShip::SetCanOpenFire (const bool CanFire)
   {
```

```
      CanOpenFire=CanFire;
   }

//***************************************************************************

   inline TShip::TShipType TShip::TShipInfo::GetShipType() const
   {
      return ShipType;
   }

//***************************************************************************
//***************************************************************************

   class TShipInfoIsLess
   {
      public :
         inline bool operator () (const TShip::TShipInfo& S1, const TShip::TShipIn
fo& S2) const
         {
            return S1.GetShipType() < S2.GetShipType();
         }
   };

//***************************************************************************
//***************************************************************************

   inline unsigned TShip::GetDefaultMaximumSpeed()
   {
      return DefaultMaximumSpeed;
   }

//***************************************************************************

   inline unsigned TShip::GetDefaultWeaponRange()
   {
      return DefaultWeaponRange;
   }



} //End namespace

#define SHIP_H
#endif
```

```
#include <strstream>
#include "Ship.h"
#include "referee.h"
#include "Game.h" //To have a reference to the referee
#include <conio>
#include "TextScreen.h" //For the position to write
#include "NoOcean.h"


namespace  SUB_Framework
{

//*************************************************************************
//*************************************************************************

using namespace  std;

//*************************************************************************
//*************************************************************************

void TThereMustBeAReferee::What() const
{
   strstream ErrMessage(Message,sizeof (Message),ios::out);
   ErrMessage<<"Initialisation error: there must be a referee."<<ends;

}

//*************************************************************************
//*************************************************************************

void TInvalidShipType::What() const
{
   strstream ErrMessage(Message,sizeof (Message),ios::out);
   ErrMessage<<"Logic error: an invalid (or uninitialised) type of ship exist."<<
ends;
}

//*************************************************************************
//*************************************************************************

void TAShipMustHaveACommandingOfficer::What() const
{
   strstream ErrMessage(Message,sizeof (Message),ios::out);
   ErrMessage<<"Initialisation error: each ship must have a commanding offficer."
<<ends;
}

//*************************************************************************
//*************************************************************************

void TShipTypeIsntRegistered::What() const
{
   strstream ErrMessage(Message,sizeof (Message),ios::out);
   ErrMessage<<"Initialisation error: the ship type that you have used "<<
               "isn't registered";
}

//*************************************************************************
//*************************************************************************
```

```
//Definition of static class member

set<TShip::TShipInfo, TShipInfoIsLess> TShip::ListOfRegisteredShipTypes;
unsigned TShip::ShipNumber=1; //Number 0 is reserved for ships
                              //build by the default constructor

unsigned TShip::DefaultMaximumSpeed=9; //The maximum acceleration used
unsigned TShip::DefaultWeaponRange=2;
                                       //if none is specified
TShip::TShipType TShip::TheBiggestShip=TShip::battleship;

TOcean* TShip::TheOcean;
const TReferee *TShip::Referee=NULL; // TShip::Referee=TGame::GetReferee();

//***************************************************************************
//***************************************************************************

ostream& operator <<(ostream &os, const TShip::TShipType ShipType)
{
   switch (ShipType)
   {
      case TShip::pt_boat:
      {
         os<<"pt-boat";
         break;
      }
      case TShip::destroyer:
      {
         os<<"destroyer";
         break;
      }
      case TShip::cruiser:
      {
         os<<"cruiser";
         break;
      }
      case TShip::battleship:
      {
         os<<"batleship";
         break;
      }
      default:
         throw TInvalidShipType();

   }
   return os;
}

//***************************************************************************
//***************************************************************************

ostream& operator <<(ostream &os, const TShip& AShip)
{
   gotoxy(TTextScreen::ScreenPositionToWrite,1);
   os<<AShip.ShipType<<"  "<<AShip.Name<<'\n';
   gotoxy(TTextScreen::ScreenPositionToWrite,2);
   os<<*AShip.CommandingOfficer<<"commanding"<<endl;

   gotoxy(TTextScreen::ScreenPositionToWrite,3);
   clreol();
   os<<"Heading: "<<AShip.ConvertHeading(AShip.GetCurrentHeading())<<" Speed: "<<
```

```
AShip.CurrentSpeed<<endl;
   gotoxy(TTextScreen::ScreenPositionToWrite,4);
   clreol();
   os<<"Target range: ";
   clreol();
   if (AShip.CanOpenFire)
      os<<'F'; //Add the letter F to remind the player that he is within firing
               //range of the target.
   else
      os<<' '; //To overwrite any F that might be on the screen
   os<<AShip.TargetRange;

   return os;
}

//*****************************************************************************

TShip::TShip(const TShipType AShipType,
             TPlayer *ACommandingOfficer,const string &ShipName,
             const unsigned Longitude, const unsigned Latitude,
             const TShipAllignment AnAllignment,
             const unsigned Depth) :

             CommandingOfficer(ACommandingOfficer)     ,
             ID(ShipNumber++),
             CurrentPosition(TCoordinate(Longitude,Latitude,Depth)),
             Name(ShipName), CurrentSpeed(0),AimingDirection(0),
             CanOpenFire(false),IsDammaged(false), ShipType(AShipType),
             Allignment(AnAllignment), TargetRange(-1), CurrentHeading(45),
             WeaponRange(2)

{

   set<TShip::TShipInfo, TShipInfoIsLess>::iterator it=
       ListOfRegisteredShipTypes.find(TShipInfo(ShipType));

   if (it != ListOfRegisteredShipTypes.end())
   {
      *this = *it;
   }
   else
      throw TShipTypeIsntRegistered();
}

//*****************************************************************************

TShip::TShip(TPlayer *ACommandingOfficer,
             const TShipType AShipType, const string &ShipName,
             const unsigned Longitude, const unsigned Latitude,
             const TShipAllignment AnAllignment,
             const unsigned Depth, const unsigned MaxSpeed,
             const unsigned MaxTurn,
             const unsigned MaxAccel,
             const unsigned WRange):

             CommandingOfficer(ACommandingOfficer)     ,
             ID(ShipNumber++), MaximumSpeed(MaxSpeed),
             MaximumTurningRate(MaxTurn),
             MaximumAccelerationRate(MaxAccel),
             CurrentPosition(TCoordinate(Longitude,Latitude,Depth)),
             Name(ShipName), CurrentSpeed(0),
```

```
                CurrentMaximumSpeed(MaxSpeed),
                CurrentMaximumTurningRate(MaxTurn),
                CurrentMaximumAccelerationRate(MaximumAccelerationRate),
                AimingDirection(0),CanOpenFire(false),
                IsDammaged(false), ShipType(AShipType), WeaponRange(WRange),
                Allignment(AnAllignment), TargetRange(-1), CurrentHeading(45)

                // An invalid value is put into Target range because the
                // target may not even has been constructed.
                // It is the referee task to set this value
{
   if (CommandingOfficer == NULL)
   {
      throw TAShipMustHaveACommandingOfficer();
   }
   if (WeaponRange > 2) //We must modify a constant
      *const_cast <unsigned *>(&WeaponRange)=2; //The basic game weapon rangeis at
most 2

   //CurrentMaximumAccelerationRate=MaximumAccelerationRate=9; //Tempo must chang
e the constructor
};

//***************************************************************************

void TShip::MoveOnTheOcean(unsigned NewHeading, unsigned NewSpeed) const
{

   if (Referee != NULL)
      const_cast <TReferee*>(Referee)->MoveOnTheOcean(*const_cast <TShip*>(this),Ne
wHeading, NewSpeed);
   else
      throw TThereMustBeAReferee();

}

//***************************************************************************

TShip::TFiringResult TShip::OpenFire(const TAimingInformation &AimingInfo)
{
   if (Referee != NULL)
      return const_cast <TReferee*>(Referee)->OpenFire(*this, AimingInfo);
   else
      throw TThereMustBeAReferee();
}

//***************************************************************************

TShip::~TShip()
{
}

//***************************************************************************

void TShip::SetCurrentPosition(const TCoordinate &NewCoordinates)
{
   CurrentPosition=NewCoordinates;
}

//***************************************************************************
```

```
void TShip::DoRepairs()
{
    const_cast <TReferee*>(Referee)->RepairAShip(*this);
}

//***************************************************************************

unsigned TShip::GetTargetRange()
{
   return TargetRange;
}

//***************************************************************************

void TShip::ComputeTargetRange()
{
   Referee->ComputeTargetRange(*this);
}

//***************************************************************************

string TShip::ConvertHeading(const unsigned AnHeading)
{
   switch (AnHeading)
   {
      case   0:  return "E";
      case  45:  return "NE";
      case  90:  return "N";
      case 135: return "NW";
      case 180: return "W";
      case 225: return "SW";
      case 270: return "S";
      case 315: return "SE";
      default :
         return "This heading cannot be converted"; //Exception is better?
   }
}

//***************************************************************************

void TShip::RandomizePosition()
{
   //The following default randomize function relies on the fact that more then
   //90% of the ocean is water. If this is not the case for a scenario, you
   //should create a new subclass of TShip and redifine this virtual
   //function.

   //WARNING: this function could run almost FOREVER if there is more land then
   //water in the game ocean.

   //All surface ships are always on the surface (depth 0).


   if (TheOcean == NULL)
      throw TThereMustBeAnOcean();
   do
      CurrentPosition.Randomize(false); //Do not randomize the depth
   while (TheOcean->GetWhatIsAtCoordinate(CurrentPosition) != water);


}
```

```
//***************************************************************************

void TShip::SetReferee(const TReferee * AReferee)
{
   Referee=AReferee;
}

//***************************************************************************

void TShip::RandomizeShipPosition(TShip *AShip)
{
   AShip->RandomizePosition();
}

//***************************************************************************

bool TShip::RegisterANewShipType(const TShipType AShipType,
                                 const unsigned MaxSpeed,
                                 const unsigned MaxTurn,
                                 const unsigned MaxAccel,
                                 const unsigned WRange)

{
    ListOfRegisteredShipTypes.insert(TShipInfo(AShipType,MaxSpeed,MaxTurn,
                                 MaxAccel,WRange));
    return true; //tempo
};

//***************************************************************************

TShip& TShip::operator =(const TShipInfo &RHS)
{
   //No need to check for equality, two different classes are always different
   CurrentMaximumSpeed=MaximumSpeed=RHS.MaximumSpeed;
   CurrentMaximumTurningRate=MaximumTurningRate=RHS.MaximumTurningRate;
   CurrentMaximumAccelerationRate=MaximumAccelerationRate=RHS.MaximumAcceleration
Rate;
   *const_cast <unsigned *>(&WeaponRange)=RHS.WeaponRange;
   return *this;
}

//***************************************************************************

void TShip::AbandonShip()//The ship isn't physically deleted, it is removed
                         //from the list of active ships
{
   const_cast <TReferee*>(Referee)->AbandonShip(const_cast <TShip*>(this));
}

//***************************************************************************

void TShip::Initialize()
{
   CurrentSpeed=0;
   CurrentMaximumSpeed=MaximumSpeed;
   CurrentMaximumTurningRate=MaximumTurningRate;
   CurrentMaximumAccelerationRate=MaximumAccelerationRate;
   AimingDirection=0;
   CanOpenFire=false;
   IsDammaged=false;
```

```
    TargetRange=-1;
     CurrentHeading=45;
}

//***************************************************************************

bool TShip::Improve()
{
   if (ShipType != TheBiggestShip)
   {
      ShipType++;

      set<TShip::TShipInfo, TShipInfoIsLess>::iterator it=

      ListOfRegisteredShipTypes.find(TShipInfo(ShipType));

      if (it != ListOfRegisteredShipTypes.end())
      {
         *this = *it; //Change fundamentals characteristics
      }
      else
         throw TShipTypeIsntRegistered();

      return true;
   }
   else
      return false;
}

//***************************************************************************
//***************************************************************************

}// End namespace
```

```
#ifndef OCEAN_H

#include "ocerep.h"

namespace SUB_Framework
{

//***************************************************************************
//***************************************************************************

// Note to the maintainer

// WARNING: the constructor of this class may throw an exception of the type
// TInvalidOceanContent. If you modify this class in such a way that it
// contains pointers then you are resposible for keeping the constructor
// safe.

class TOcean
{
   public:
      TOcean(const unsigned DimX, const unsigned DimY,
             TOceanRepresentation &Rep);

      TOceanContent GetWhatIsAtCoordinate(const TCoordinate &ACoordinate) const;
      bool IsNavigable(const TCoordinate &ACoordinate) const;

      unsigned GetDimensionX() const;
      unsigned GetDimensionY() const;

   private:
      const unsigned DimensionX, DimensionY;
   virtual void CheckIntegrity(const TOceanContent First=water,
                               const TOceanContent Last=hurricane);

      TOceanRepresentation &Representation;

};

//***************************************************************************
//***************************************************************************


bool TOcean::IsNavigable(const TCoordinate &ACoordinate) const
{
   return Representation.IsNavigable(ACoordinate);
}

//***************************************************************************

inline unsigned TOcean::GetDimensionX() const
{
   return DimensionX;
}

//***************************************************************************

inline unsigned TOcean::GetDimensionY() const
{
   return DimensionY;
}
```

```
//***************************************************************************

inline TOceanContent TOcean::GetWhatIsAtCoordinate(const TCoordinate &ACoordinate
) const
{
   return Representation.GetWhatIsAtCoordinate(ACoordinate);
}
//***************************************************************************

} //End namespace

#define OCEAN_H
#endif
```

```
//#include <strstream>

#include "Ocean.h"

namespace SUB_Framework
{

//********************************************************************************
//********************************************************************************

TOcean::TOcean(const unsigned DimX, const unsigned DimY,
               TOceanRepresentation &Rep):
                     DimensionX(DimX), DimensionY(DimY),
                     Representation(Rep)
{
   TCoordinate::SetMaxLongitude(DimensionX);
      TCoordinate::SetMaxLatitude(DimensionY);
   CheckIntegrity();
};

//********************************************************************************

void TOcean::CheckIntegrity(const TOceanContent First,
                                  const TOceanContent Last)
{
   TCoordinate Coordinates;
   TOceanContent Content;

   unsigned j;
   for (unsigned i=0; i< DimensionX; i++)
   {
      for (j=0; j< DimensionY; j++)
      {
         /*try
         { */
            Coordinates=TCoordinate(i,j); //Check validity of all the coordinates
/*       }
         catch (TSubError &AnError)
         {
            cout<<AnError<<endl;

            char c;
            cin>>c;
         }*/

         //Check the validity of the ocean content

         Content= Representation.GetWhatIsAtCoordinate(Coordinates);

         if (First > Content || Content > Last)
            throw (TInvalidOceanContent(Coordinates));
      }
   }
}

//********************************************************************************

} //End namespace
```

```
#ifndef INVALIDCOORDINATES_H

#include <iostream>
#include "SubExceptions.h"
namespace SUB_Framework
{

//********************************************************************************
//********************************************************************************

class TInvalidCoordinates :public TLogicError
{

   public:
      TInvalidCoordinates(const unsigned Longi, const unsigned MaxLongi,
                          const unsigned Lati, const unsigned MaxLat,
                          const unsigned Dep, const unsigned MaxDep); //Depth and
 MaxDepth


   private:
      virtual void What() const;
      unsigned const Latitude, MaxLatitude, Longitude, MaxLongitude,
                     Depth, MaxDepth;
};

//********************************************************************************
//********************************************************************************

} //End namespace

#define INVALIDCOORDINATES_H
#endif
```

```
#include <strstream>
#include "InvalidCoordinates.h"

namespace SUB_Framework
{

//*****************************************************************************

void TInvalidCoordinates::What() const

{
   Message[0]='\0';
   strstream ErrorStream(Message, sizeof(Message),ios::out);
   ErrorStream<<"Invalid coordinates: ("<<Latitude<<','<<Longitude<<")\n";
   ErrorStream<<"The limits are ("<<MaxLatitude<<','<<MaxLongitude<<")"<<endl<<en
ds;
   //cerr<<ErrorStream.str();
}

//*****************************************************************************

TInvalidCoordinates::TInvalidCoordinates(const unsigned Longi,
                           const unsigned MaxLongi,
                           const unsigned Lati, const unsigned MaxLat,
                           const unsigned Dep, const unsigned MaxDep):
                           Latitude(Lati), Longitude(Longi),
                           MaxLatitude(MaxLat), MaxLongitude(MaxLongi),
                           Depth(Dep),MaxDepth(MaxDep)
{
}

//*****************************************************************************

} //End namespace
```

```
#ifndef SUBEXCEPTIONS_H
#include <iostream>


// This include files contains all the abstract class that should be
// used to throw exception.

// The mother of all exceptions

// The What() function should fill the message buffer with an appropiate
// error message.

// The constructor for SubError MUST NOT THROW ANY EXCEPTION. It must
// not allocate memory from the free store as this may lead to an exception
// being thrown.

namespace SUB_Framework
{

//*****************************************************************************
//*****************************************************************************

typedef enum {logicError,initialisationError} TErrorType;

//*****************************************************************************
//*****************************************************************************

class TSubError
{
   friend ostream& operator<<(ostream& os, const TSubError& S);

   public:
      TSubError(const TErrorType Et, const char * const AFileName,
                unsigned ALineNumber);
      virtual ~TSubError()=0; //Abstract base class must have a virtual destructo
r

   private:
      const TErrorType ErrorType;

      const char * const FileName;
      const unsigned LineNumber;

      virtual void What() const=0;

   protected:
      static char Message[2048]; // In order to avoid call to operator new
                                 // a fixed size has to be used.

};

//*****************************************************************************
//*****************************************************************************

inline TSubError::TSubError(const TErrorType Et, const char * const AFileName,
                unsigned ALineNumber) : ErrorType(Et), FileName(AFileName),
                                        LineNumber(ALineNumber)
{

};
```

```
//*****************************************************************************

inline TSubError::~TSubError()
{
};


//*****************************************************************************
//*****************************************************************************


// The mother of all functions precondition violation errors

class TLogicError: public TSubError
{
   public:
      TLogicError(const char * const AFileName,
                  unsigned ALineNumber);
};


inline TLogicError::TLogicError(const char * const AFileName="",
                  unsigned ALineNumber=0) : TSubError(logicError,AFileName,ALineNum
ber)
{
}


//*****************************************************************************
//*****************************************************************************

// The mother of all initialisation errors

class TInitialisationError: public TSubError
{
   public:
      TInitialisationError(const char * const AFileName="",
                           unsigned ALineNumber=0) :
                           TSubError(initialisationError,AFileName,ALineNumber)
      {
      };

};

//*****************************************************************************
//*****************************************************************************

} // End Namespace SubFrasmework
#define SUBEXCEPTIONS_H
#endif
```

```
#include "SubExceptions.h"

namespace SUB_Framework
{

//*****************************************************************************
//*****************************************************************************

// definition of static members

char TSubError::Message[2048];

//*****************************************************************************
//*****************************************************************************

ostream& operator<<(ostream& os, const TSubError& S)
{   S.What();
    os<<S.Message<<endl;
    return os;
}

}// End namespace
```

```
#ifndef COORDINATE_H

// This class represent the coordinates of an elementary element of the
// abstract ocean.


#include "InvalidCoordinates.h"

namespace SUB_Framework
{


//*********************************************************************************
//*********************************************************************************

class TCoordinate
{
   public :
      friend ostream& operator <<(ostream&os, const TCoordinate& Coordinates);

      bool operator ==(const TCoordinate &Coordinates) const;

      explicit TCoordinate(unsigned Longi=0, unsigned Lati=0,
                           unsigned Depth=0);
      unsigned GetLongitude() const;
      unsigned GetLatitude() const;
      unsigned GetDepth() const;
      unsigned ComputeDistance(const TCoordinate &AnotherCoordinate);

      void Randomize(const bool RandomizeDepth);
      static bool WouldBeLegalCoordinate(const unsigned Longi, const unsigned Lat
i,
                                         const unsigned depth=0);
      static unsigned GetMaxLatitude();
      static unsigned GetMaxLongitude();
      static unsigned GetMaxDepth();
      static void SetMaxLatitude(const unsigned MaxLati);
      static void SetMaxLongitude(const unsigned MaxLongi);
      static void SetMaxDepth(const unsigned MaxDepth);

   protected :
      void SetLatitude(const unsigned NewLatitude);
      void SetLongitude(const unsigned NewLongitude);
      void SetDepth(const unsigned NewDepth);
   //private:
      unsigned Longitude, Latitude, Depth;

// Those variables are initialized to INVALID value to ensure that
// the user will call SetMaxLongitude and SetMaxLatitude.
// If such a call isn't made, only the coordinate (0,0) will be valid!
// All others will throw an exception.

      static unsigned MaxLatitude;
      static unsigned MaxLongitude;
      static unsigned MaxDepth;


};

//*********************************************************************************
//*********************************************************************************
```

```
//An aiming coordinate is a coordinate for which a full set of mutators
//have been defined

class TAimingCoordinate : public TCoordinate
{
   public :
      explicit TAimingCoordinate(unsigned Longi=0, unsigned Lati=0,
                                 unsigned Depth=0);
     TAimingCoordinate& operator =(const TCoordinate &ACoordinate)    //tempo
      {
         //*(dynamic_cast<TCoordinate>(this))=ACoordinate;
         Longitude =ACoordinate.GetLongitude();
         Latitude=ACoordinate.GetLatitude();
         Depth=ACoordinate.GetDepth();
         return *this;
      }
      //operator TCoordinate() {return *dynamic_cast<TCoordinate * const>(this);}
;
      TAimingCoordinate(const TCoordinate &ACoordinate):TCoordinate(ACoordinate){
};

      void SetDepth(const unsigned NewDepth);
      void SetLongitude(const unsigned NewLongitude);
      void SetLatitude(const unsigned NewLatitude);
};

//*****************************************************************************
//*****************************************************************************

inline TCoordinate:: TCoordinate(unsigned Longi, unsigned Lati, unsigned Dept):
   Longitude(Longi), Latitude(Lati), Depth(Dept)
{
   if ( Longi > MaxLongitude || Lati > MaxLatitude || Depth > MaxDepth)
      throw (TInvalidCoordinates(Longi,MaxLongitude,Lati,MaxLatitude,
             Depth,MaxDepth));
}

//*****************************************************************************

inline unsigned TCoordinate::GetLongitude() const
{
   return Longitude;
}

//*****************************************************************************

inline unsigned TCoordinate::GetLatitude() const
{
   return Latitude;
}

//*****************************************************************************

unsigned TCoordinate::GetDepth() const
{
   return Depth;
}

//*****************************************************************************
```

```
inline unsigned TCoordinate::GetMaxLatitude()
{
   return MaxLatitude;
}

//*****************************************************************************

inline unsigned TCoordinate::GetMaxLongitude()
{
   return MaxLongitude;
}

//*****************************************************************************

inline unsigned TCoordinate::GetMaxDepth()
{
   return MaxDepth;
}

//*****************************************************************************

inline bool TCoordinate::WouldBeLegalCoordinate(const unsigned Longi,
                                                const unsigned Lati,
                                                const unsigned Depth)
{
   return Longi <= MaxLongitude && Lati <= MaxLatitude && Depth <= MaxDepth;
}

//*****************************************************************************

inline bool TCoordinate::operator ==(const TCoordinate& Coordinates) const
{
   return Latitude==Coordinates.Latitude  &&
          Longitude==Coordinates.Longitude &&
          Depth==Coordinates.Depth;
}

//*****************************************************************************
//Protected member functions
//*****************************************************************************

void TCoordinate::SetLatitude(const unsigned NewLatitude)
{
   if (NewLatitude <= MaxLatitude)
      Latitude=NewLatitude;
   else
      throw TInvalidCoordinates(Longitude,MaxLongitude,NewLatitude,MaxLatitude,
                                Depth,MaxDepth);
}
      void SetLongitude(const unsigned NewLongitude);
      void SetDepth(const unsigned NewDepth);

//*****************************************************************************

inline TAimingCoordinate::TAimingCoordinate(unsigned Longi, unsigned Lati,
                                            unsigned Depth):TCoordinate(Longi,Lati,Depth
)
{
}

//*****************************************************************************
```

```
inline void TAimingCoordinate::SetLongitude(const unsigned NewLongitude)
{
   TCoordinate::SetLongitude(NewLongitude);
}

inline void TAimingCoordinate::SetLatitude(const unsigned NewLatitude)
{
   TCoordinate::SetLatitude(NewLatitude);
}
inline void TAimingCoordinate::SetDepth(const unsigned NewDepth)
{

   TCoordinate::SetDepth(NewDepth);
}

} //End namespace


#define COORDINATE_H
#endif
```

```
#include <stdlib.h> //For random
#include "Coordinate.h"

namespace SUB_Framework
{

// static variables are initialized to 0 automatically.
// explicit initialisation has been used to enhence readability
// but isn't really required.

//*****************************************************************************
//*****************************************************************************

unsigned TCoordinate::MaxLatitude=0;
unsigned TCoordinate::MaxLongitude=0;
unsigned TCoordinate::MaxDepth=3;

//*****************************************************************************
//*****************************************************************************

void TCoordinate::SetMaxLatitude(const unsigned MaxLati)
{
   if (MaxLatitude==0) //Only one setting of MaxLatitude is allowed

      MaxLatitude=MaxLati;
}

//*****************************************************************************

void TCoordinate::SetMaxLongitude(const unsigned MaxLongi)
{
   if (MaxLongitude==0) //Only one setting of MaxLatitude is allowed
      MaxLongitude=MaxLongi;
}

//*****************************************************************************

void TCoordinate::SetMaxDepth(const unsigned MaxDept)
{
   if (MaxDepth==0) //Only one setting of MaxDepth is allowed
      MaxDepth=MaxDept;
}

//*****************************************************************************

void TCoordinate::Randomize(const bool RandomizeDepth)
{
   Longitude=random(MaxLongitude);
   Latitude=random(MaxLatitude);
   if(RandomizeDepth)
      Depth=random(MaxDepth)+1;
   else
      Depth=0;
}

//*****************************************************************************

unsigned TCoordinate::ComputeDistance(const TCoordinate &AnotherCoordinate)
{
```

```
    if (*this == AnotherCoordinate)
       return 0;

    //Longitude and Latitude are unsigned. The range between two coordinate
    //migth be greter then INT_MAX so we must handle the details ourself.

    unsigned DeltaLongitude = Longitude >=AnotherCoordinate.Longitude?
                              Longitude-AnotherCoordinate.Longitude:
                              AnotherCoordinate.Longitude-Longitude;
    unsigned DeltaLatitude = Latitude >=AnotherCoordinate.Latitude?
                              Latitude-AnotherCoordinate.Latitude:
                              AnotherCoordinate.Latitude-Latitude;

    return DeltaLongitude > DeltaLatitude ? DeltaLongitude: DeltaLatitude;
}

//***************************************************************************

ostream& operator <<(ostream&os, const TCoordinate& Coordinates)
{
   os<<'('<<Coordinates.Longitude<<','<<Coordinates.Latitude<<')';
   return os;
}

//***************************************************************************

void TCoordinate::SetDepth(const unsigned NewDepth)
{

if (Depth <=MaxDepth)
      Depth=NewDepth;
   else
      throw TInvalidCoordinates(Longitude,MaxLongitude,
                                Latitude,MaxLatitude,
                                NewDepth,MaxDepth);
}

//***************************************************************************
void TCoordinate::SetLongitude(const unsigned NewLongitude)
{

if (Longitude <= MaxLongitude)
      Longitude=NewLongitude;
   else
      throw TInvalidCoordinates(NewLongitude,MaxLongitude,
                                Latitude,MaxLatitude,
                                Depth,MaxDepth);
}

} // End namespace
```

```
#ifndef OCEREP_H
#include<iostream>
#include "coordinate.h"
#include "SubExceptions.h"

namespace SUB_Framework
{
// This enumerated type defined what are the allowable STATIC ocean
// features. Moving object on or under the sea are NOT covered.
// water is used for an ordinary, featureless ocean position.
// The effect of those features in the game must be specified in the
// REFEREE class, not here.

typedef enum
   {water,shallow_water,island,reef,mine,storm,hurricane,invalid_ocean_content} T
OceanContent;

// It is appropriate to separate the Ocean representation from the Ocean
// definition: there is no universal representation that can satisfies
// each case.

// An Ocean representation must return the TOceanContent of any ocean
// coordinates. The valid coordinates ranges from 0 to MaxLat and
// 0 to MaxLong.

// GetWhatIsAtCoordinates can throw the following two exceptions:
// SubError::LogicErrors::OutOfTheOceanCoordinates and
// SubError::InitialisationErrors::InvalidOceanContent

ostream& operator <<(ostream& os, const TOceanContent &Oc);

class TInvalidOceanContent : public TInitialisationError
{
   public :
      TInvalidOceanContent(TCoordinate ABadCoordinate);
   private :
      virtual void What() const ;
      TCoordinate BadCoordinates;
};

class TOceanRepresentation
{
   public :
      virtual TOceanContent GetWhatIsAtCoordinate
               (const TCoordinate &Coordinates)const =0;

      //This function returns true if a ship can safely occupy the position
      //at coordinate ACoordinate, false otherwise. The referee must take
      //suitable action each time that a ship attempt to enter a non navigable
      //coordinate. This function DOES NOT DETECT COLLISION with other ships.

      virtual bool IsNavigable(const TCoordinate &ACoordinate)=0;

      virtual ~TOceanRepresentation();
};


inline TOceanRepresentation::~TOceanRepresentation()
{
}
```

```
} //End namespace
#define OCEREP_H
#endif
```

```
#include <strstream>
#include "ocerep.h"

namespace  SUB_Framework
{

//*****************************************************************************

TInvalidOceanContent::TInvalidOceanContent(TCoordinate ABadCoordinate):
                        BadCoordinates(ABadCoordinate)
{
};

//*****************************************************************************

void TInvalidOceanContent::What() const
{
   strstream ErrStream(Message,sizeof (Message),ios::out);
   ErrStream<<"Initialisation error: invalid ocean content at coordinate ";
   ErrStream<<BadCoordinates<<'.'<<ends;
}

//*****************************************************************************

ostream& operator  <<(ostream& os, const TOceanContent &Oc)
{
   char c;

   switch  (Oc)
   {

      case water:
         c='.';
         break;
      case shallow_water:
         c='*';
         break;
      case mine:
         c='M';
         break;
      case island:
         c='I';
         break;
      case reef:
         c='R';
         break;
      case storm:
         c='S';
         break;
      case hurricane:
         c='H';
         break;
      default :
         c='E'; //E means error, since this is displayed on the screen, any
                //error will be quickly spotted, there is no need to throw
                //an exception

   } // end switch
   os<<c;
   return os;
}
```

```
} //End namespace
```

```
// The representation of the Ocean for the Basic Game

// The basic game ocean is a 50x50 flat square grid
// surronded on all side by a width 2 zone of shallow water

#ifndef BASICOCEREP_H

#include "ocerep.h"

namespace SUB_Framework
{

class TBasicGameOceanRepresentation: public TOceanRepresentation
{
   public :
      TBasicGameOceanRepresentation(const unsigned Border=2);

      virtual TOceanContent GetWhatIsAtCoordinate
              (const TCoordinate& Coordinates) const ;

      virtual bool IsNavigable(const TCoordinate &ACoordinate);
      virtual ~TBasicGameOceanRepresentation();

   private :
      const SizeOfTheShallowBorder;
};

inline TBasicGameOceanRepresentation::
   TBasicGameOceanRepresentation(const unsigned Border):
      SizeOfTheShallowBorder(Border)
      {
      }

inline TBasicGameOceanRepresentation::~TBasicGameOceanRepresentation()
{
}

inline bool TBasicGameOceanRepresentation::IsNavigable(const TCoordinate &ACoordi
nate)
{
   TOceanContent Oc= GetWhatIsAtCoordinate(ACoordinate);
   return Oc == water || Oc == shallow_water;
}
} //end namespace

#define BASICOCEREP_H
#endif
```

```
#include "BasicOceRep.h"

namespace SUB_Framework
{

TOceanContent TBasicGameOceanRepresentation::GetWhatIsAtCoordinate
                        (const TCoordinate& Coordinates) const
{
   if (Coordinates.GetLatitude() <= SizeOfTheShallowBorder -1
       || Coordinates.GetLatitude() >= Coordinates.GetMaxLatitude() - SizeOfTheSh
allowBorder +1
       || Coordinates.GetLongitude() <= SizeOfTheShallowBorder-1
       || Coordinates.GetLongitude() >= Coordinates.GetMaxLongitude() - SizeOfThe
ShallowBorder+1)
      return shallow_water;

      return water; //Default case
}

} //End namespace
```

```
#ifndef RANK_H


#include <string>

using namespace std;

namespace SUB_Framework
{

class TRank
{
   friend ostream& operator <<(ostream& os, const TRank& Rank);

   private :
      string Name;
   public :
      explicit TRank( const string &AName=""); //A default constructor is require
d to create vector of this type


      const string& GetName() const;
      bool operator ==(const TRank &AnotherRank) const;
      bool operator !=(const TRank &AnotherRank) const;

};
inline const string& TRank::GetName() const
{
   return Name;
}

inline bool TRank::operator ==(const TRank &AnotherRank) const
{
   return Name==AnotherRank.Name;
}

inline bool TRank::operator !=(const TRank &AnotherRank) const
{
   return !operator ==(AnotherRank);
}

} //End Namespace

#define RANK_H
#endif
```

```
#include "rank.h"


namespace  SUB_Framework
{

explicit  TRank::TRank( const  string &AName): Name(AName)
{
}

ostream& operator <<(ostream& os, const  TRank& Rank)
{
   os<<Rank.Name;
   return  os;
}

} //End Namespace
```