Bilkent University
Dept. of Computer Engineering

Object Oriented Software Engineering Project

RoM: Redeemers of the Monarchy

Design Report (1-3)

Project members: Efe Ulas Akay Seyitoglu, Erkam Berker Senol, Izel Gurbuz, Nashiha Ahmed
Course Instructor: Ugur Dogrusoz

Design Report (1-3)
November 12, 2016

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Object Oriented Software Engineering Project, CS319.

Contents

Design Report

1 Introduction

Our project is a "Tower Defense" type game. It will be a Java desktop application. Tower Defense is a strategy game. In Tower Defense games, players aim to defend their territories or possessions from enemies by destroying them before they reach the endpoint.

This design report is a kind of internal document, aimed to aid developers of the game. Specifically, this design report gives a description of the design of the game, which includes software architecture and subsystem services.

1.1 Purpose of the System

Redeemers of the Monarchy is a system designed to provide an easy-to-use and entertaining experience to the user. We intend to challenge the game player's strategic skills in this game of strategy. We aim to make the game fast and with as minimum bugs as possible. The game will have a simple, minimalistic user interface that is straightforward for the player to use.

1.2 Design Goals and Tradeoffs

1.2.1 Adaptability and Portability

The game will be programmed using Java. Java is a platform-independent language. Its platform-independence can allow it to work on all platforms that support Java, making it adaptable and portable to almost all platforms. Since our focus is to make the game a desktop application, it will not be available on the web or on mobile devices. However, we aim to code with the foresight to make the game available on the web and mobile devices.

1.2.2 Usability

The game will be easy-to-use through a simple and minimalistic user interface. We will use classic and straightforward user interface elements, keeping the design similar to existing games. The user interface elements will be consistent, as well. Input will be taken through classic keyboard shortcuts (arrow buttons, spacebar, enter key, esc, and "w, a, s, d OR i, k, j, l") and mouse-clicks. The tradeoff here is that the game will not be challenging or innovative in terms of UI and I/O; however, our main focus is to make the game challenging in its strategy- not challenging to use.

1.2.3 Reliability and Maintainability

To keep the system reliable with as minimum bugs, we will keep the system maintainable. We aim to do this by writing many tests. We may write test programs to test the code that we ourselves have not written to minimize programmer bias. We may do this by writing test programs for each other's code. We will also aim to keep the software and software logic simple, so that it is easy to maintain. We will keep our programs will documented so that we can review and develop each other's code. We aim to keep this design document simple and consistent as well so we are all "on the same page."

1.2.4 Extensibility

Because the game is written in Java and will be object-oriented, manipulations, development, and extensions to the program will be easy to implement. The object concerned will simply be tweaked. As explained in later sections, we will implement the game using the "Model, View, Controller" architecture. This will also benefit us in extensibility. For example, if we want to develop the model of the game, we may be able to do this without having to make tremendous changes to the view and controller.

1.2.5 High Performance and Efficiency

We aim to keep the framerate of our game around 30-40 frames per second. We also aim to keep a maximum of 2s response time. This is to ensure a smooth gameplay. Although we will try to avoid problems with high performance and efficiency completely, we aim to keep the user informed if a problem is encountered. For example, if the system is taking a long time to respond, we will display some sort of user interface that shows the user that the game is loading and has encountered a

problem. If the response time is too long, we will display a message to the user and restart the game.

1.2.6 Low cost

Our game is completely free since our game is mainly for this CS319 project. The tradeoffs of that is the game is like to not be maintained after the project is submitted. However, we aim to make the quality of the game as if it were to be purchased. We also aim to make the game such that it can be developed on in the future.

2 Software Architecture

2.1 Subsystem Decomposition

MVC ("Model, View, Controller") architecture will be used to design our system. The reason we chose MVC is because with MVC it is easier to simulate gaming experience into the software system compared to other architectures. As mentioned before, we also want to use MVC because it enables extensibility. MVC decouples data entity objects and data presentation. We can easily update the User Interface without greatly influencing the entity objects. Our system consists of three different subsystems namely the Model, View and Controller subsystems. These subsystems accomplish different tasks but they are dependent on each other. Relationships between them are depicted as in Figure 1.

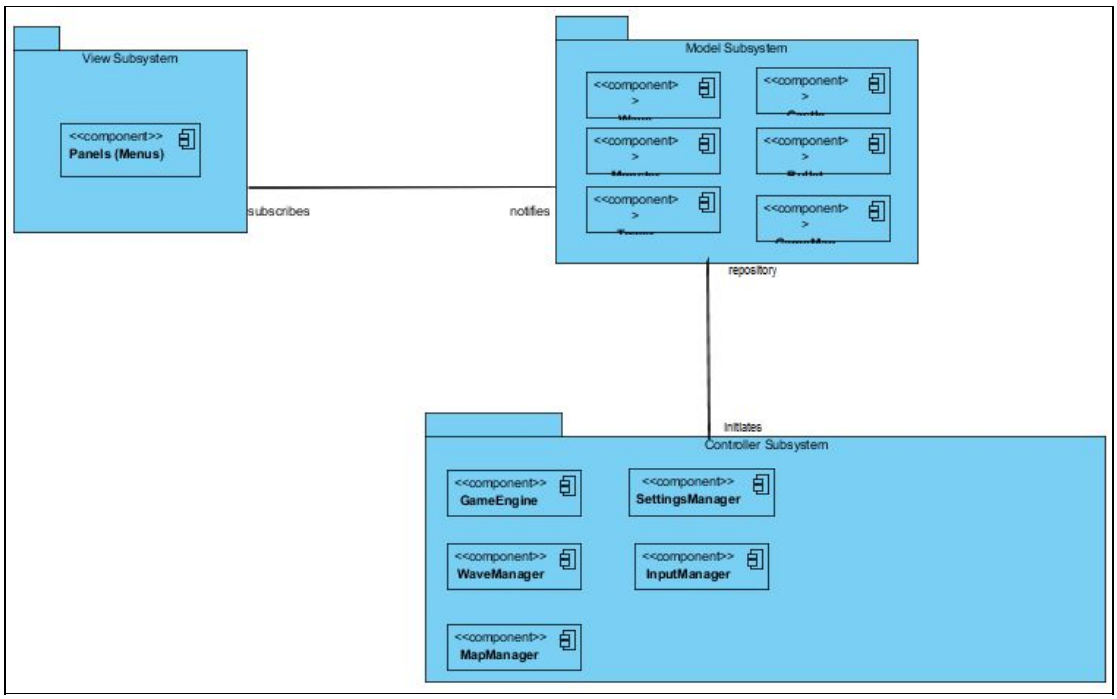

Figure 1-*Relationship between the subsystems*[1]

2.1.1 Model Subsystem

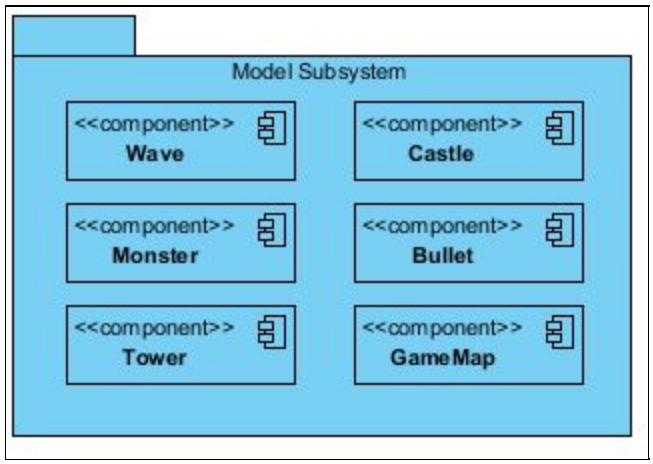

Figure 2 - *Model Subsystem*

In typical MVC patterns, the model is the core unit of the program. The data accesses or entity objects are in the Model subsystem. Model subsystem in our project will follow the traditional way of model subsystems to form the structure of the game.[3]

2.1.2 View Subsystem

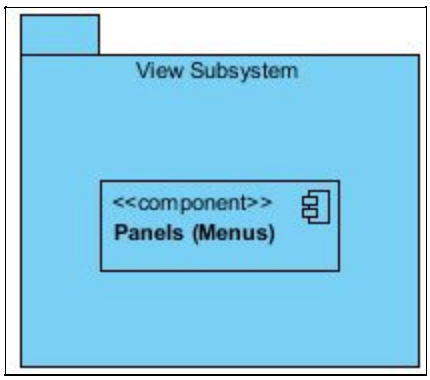

Figure 3 - *View Subsystem*

The View system represents data. It has the boundary objects, which will be the User Interface with which the user interacts. In the View subsystem, the menu interactions between the player and the game will be handled. Meaning, when the player pauses the game or enters the game at the first time the View subsystem will be notified. Furthermore at the end of the game, when the user is asked to provide information to be entered to the high-score list this sub-system will be notified. The View will have one frame and multiple panels. Thus, we will use the CardLayout in Java. Java Swing will be used for our User Interface.

2.1.3 Controller Subsystem

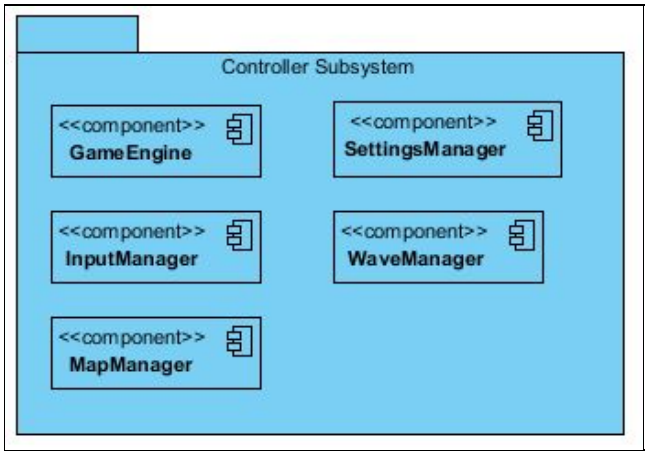

Figure 4 - *Controller Subsystem*

The Controller subsystem pieces together the Model and the View. It handles the user interactions with the boundary objects and notifies the entity objects as needed. As it is the case for programs that follow the MVC architecture, our system will also use the controller subsystem to dictate the control flow.[2] The actions that the player takes during the game will be managed by this component of our program.

2.2 Hardware/Software Mapping

The project will be implemented using Java and we will use JDK 8. The required hardwares for inputs are keyboard and mouse. Keyboard will be used for shortcuts and typing the name at high score table. Specifically, the arrow buttons (up, down, left, and right) and "w, a, s, d" or "i, k, j, l" can be used to navigate through selections. Enter button and spacebar selects an option. The mouse can be used to navigate through the application. A double left-click selects the user interface object it clicks over. The high score and latest settings will be saved in a .txt file in the memory. For sounds and music .MP3 files will be used and for images .jpeg files will be used, so the operating system should support .txt, .jpg/jpeg and .MP3 files. The computer does not need internet connection to run the game.

2.3 Persistent Data Management

We will not hold a database. The game's data, such as the pictures, settings, and high score list will be saved on files. These files will be on the user's hard drive and loaded when needed by the game. Sound files will be stored as .MP3 files and images are stored in .jpg/.jpeg formats. The data will be managed by the "Manager" classes.

2.4 Access Control and Security

The player will not need to sign up or in to play our game. Therefore, there will not be a password or identity check. The reason of this kind of implementation is that the player will only have access to the desktop application. Thus, access control and security is not a major concern. Many of our methods and classes will be made private to disable users from manipulating our code. Access to the files is only managed by our program, specifically the "Game Engine Class" and some "Manager" classes.

2.5 Boundary Conditions

When the user opens the .exe file (assuming the user had already installed the game to the computer), the system presents the main menu. Options given by the main menu include: Play, Help, Settings, High Scores, Tutorial, and Exit. In order for the game to start, the user needs to select play. When play is selected, the game engine class will respond by starting the game, which calls the necessary methods from other classes to start the game (MapManager loads the map). The game is paused if the player clicks the pause button during the game. Player is presented with options (Resume, Help, Settings, High Scores, and Exit.) Resume game returns back to the game. Exit terminates the game. The game displays high score at the end of every game, whether won or lost (a player can get a score even though he/she had not won). The high score table is only updated if the player has a high score above the tenth score in the high score table. The game will return to the main menu after the high score table is shown. If the game shuts down before the high-score of the player is entered, for whatever the reason may be, the high score is not saved.

JRE must be installed to run the game. If the platform the application is on does not support Java, an error message will pop up to the user. If the system takes more than 2s to load when the application is running, the user will be notified. There will be an internal loading bar. If the system takes longer than 1 minute, the application will be restarted. If the user runs into an error (any kind of run-time error during the game), an error message will pop up, and the user will be requested to restart the app. If the game is interrupted by power loss or forced termination, the game will start from the beginning when the user restarts the application. If the user tries to open the application while the game is already running, an error message will pop up and will request the user to terminate the current game before opening the application again. If any kind of internal material that the game needs are lacking, the user will also be notified.

3 Subsystem Services

Model-view-controller design pattern will be used in the implementation. The system will be decomposed into three subsystems. In this section each subsystem will be described.

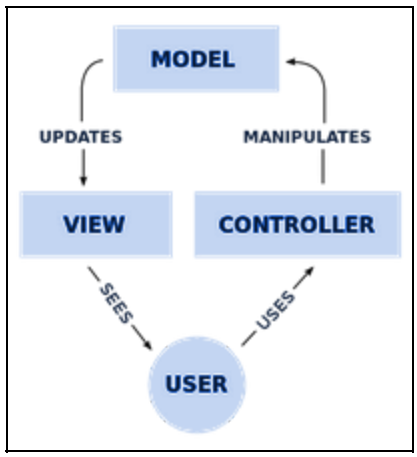

Figure 6 *- MVC Diagram [3]*

3.1 Model Services

The data of the objects are encapsulated in model subsystem, it responds to requests from the view subsystem about its current state and model can be updated with controller. Model can change with the effect from other sources such as user inputs or game operations. Controller is responsible for stimulating model service. Model updates its data with its own update methods which is triggered by controller. Furthermore model has connections with both the view and the controller as it is seen from figure 1.

3.2 View Services

The view sub-system contains all the panels that is in the interaction with the user. It is used to display the changes in the models. Model notifies the view (since they are in connection as in figure 1) . According to the given notifications the view updates to the user.

3.3 Controller Services

In order for the whole sub-system to work, the entity objects and other important classes need to be managed. This management is done through Controller -Manager- classes. For example, at the start of the game, the map will be loaded safely. In order to safely load the game the MapManager class manages the loading of the map. Furthermore, we have designed our controller sub-system in such a way that all our objects are controlled through such manager classes. These manager classes also interact with the user interface subsystem since the states of the objects are reflected on the user-interface throughout the game.

4 References

[1] "CS 319 Object-Oriented Software Engineering." Accessed November 11, 2016. http://www.cs.bilkent.edu.tr/~ugur/teaching/cs319/.

[2] Object-Oriented Software Engineering, Using UML, Patterns, and Java, 2nd Edition, by Bernd Bruegge and Allen H. Dutoit, Prentice-Hall, 2004, ISBN: 0-13-047110-0.

[3] MVC Process, Viewed 12 November 2016, <https://upload.wikimedia.org/wikipedia/commons/thumb/a/a0/MVC-Process.svg/2000px-MVC-Process.svg.png >.