Bilkent University

Department of Computer Engineering

CS - 319 Object-Oriented Software Engineering

Term Project - Final Report

Iteration 2

Project Name : Risk

Group No: 1F

Group Members: Rumeysa Özaydın
Merve Kılıçarslan
Ahmet Serdar Gürbüz
Elnur Aliyev
Osman Burak İntişah

Table of Contents

1. Introduction

The design stage of the Risk game was already begin in the time frame of the first iteration of the Analysis Report in order to choose a suitable programming language and libraries to be used in the implementation. After the end of the first iteration, the implementation of the additional functionalities and bug fixing began. Implementation was performed on different IDE's and a Github repository is used for collaboration of the team members. In addition to GitHub, the social media programs such as Skype and Discord is used in order to communicate among group members.

For the visuals used in the game, a revision has been made to improve the resolutions. The implementation of the promised functional requirements and the additional functionalities that are presented in the Analysis Report were done by the end of the second iteration. The implemented functionalities are discussed further in the following section.

2. Implemented Functionalities

For the second iteration, our aim was to complete the promised functional requirements with a neat UI.

Currently, our system supports the following functionalities:

- Screen transitions
- Risk game map and zoom in zoom out functionalities
- Mouse and event listeners for converting selected regions to coordinates
- Soldier Replacement
- Building a castle to the selected province
- Buying soldiers
- Attacking another province
- Getting and using a bonus card
- Background music with adjustable volume
- Distinguishing between provinces
- Playing the game from different computers from the same network
- Sending the current state of the game after each turn via sockets

3. Design Changes

During the implementation of the game, our team have decided to make some design changes in the implementation of the game. Even though these changes are not major, there were small effects to the design however general design remains the same.

3.1 Extended Dependencies

While we were designing our program, we have encountered that it is not easy to write independent classes while keeping the whole code simple and easy to understand. Thereby, in order to be on the safe side and keep our design simple we have introduced many dependencies between our classes. For instance, ProvincePanel were not dependent on the Game. However, they should display according to result of the battle, the current situation of that province and update the views. So they dependent on Game to get the CurrentData.

3.2 Private Utility Functions

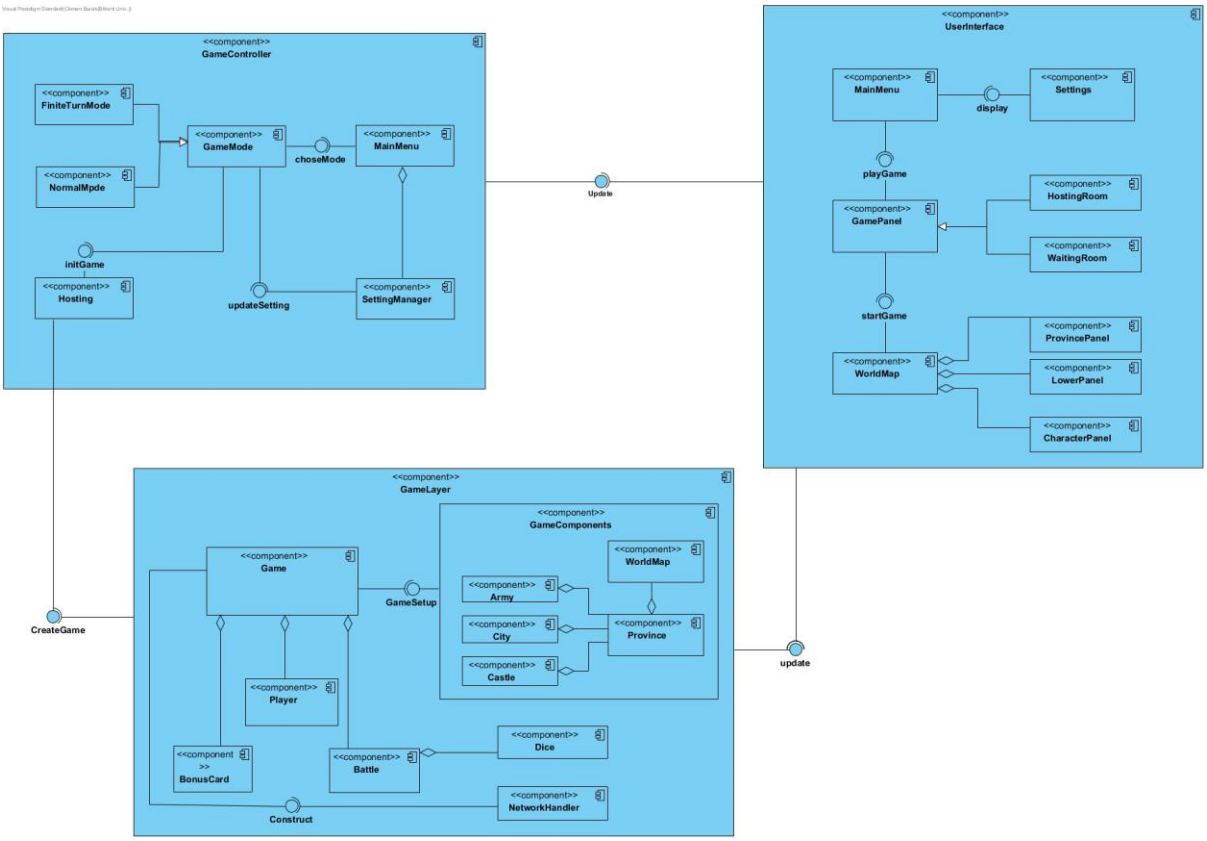
We had little knowledge on the functions of the classes and their parameters. When implementing the Risk game, functions have became more clear and and their algorithms were structured. Sub designs and private functions were not featured in the diagrams.

3.3 Sub-Packages

In our design we have splitted our classes into subsystems they belong. Every subsystem has its package which are actually; managers, game, menu and network packages. And we have used SFML library as an external package. The packages we have used are sfml-audio.lib, sfml-window.lib, sfml-graphics.lib. These packages help us to integrate the audio into the game, creating windows and implementing user interface design.

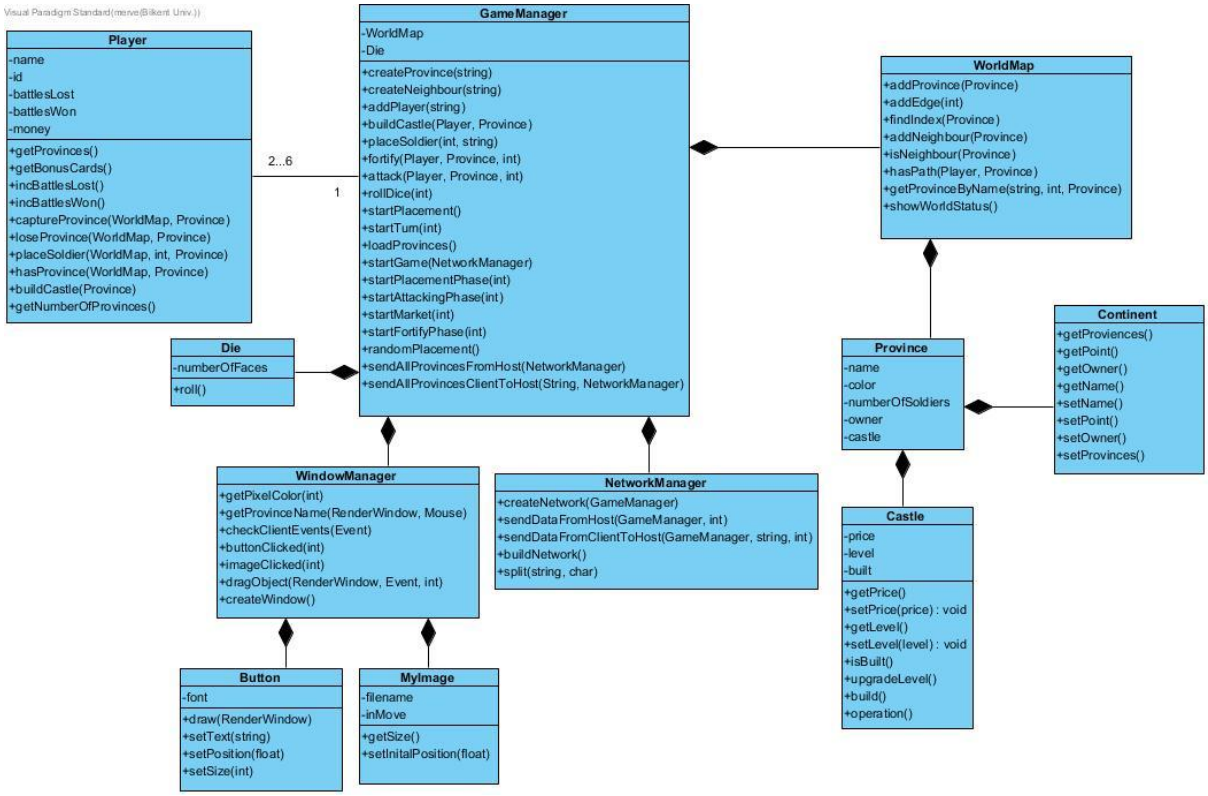
3.4 Design Decisions and Patterns

In the first iteration of the design report, it can be seen that the classes were not properly ordered. Thus, in the second iteration the system decompositions was re-done. We have realized that MVC Design Pattern is the most suitable design pattern for our project since we can divide our software with respect to model, view and controller. Since our decomposition is changed, our packages were re-arranged. In addition, we applied “Singleton Design Pattern” for WorldMap class since we wanted only one handler/controller for active map. We also applied “Factory design Pattern” for City, Province, Dice and WorldMap classes since they are entities that are limited and known beforehand. Overall, our design is improved as follows:



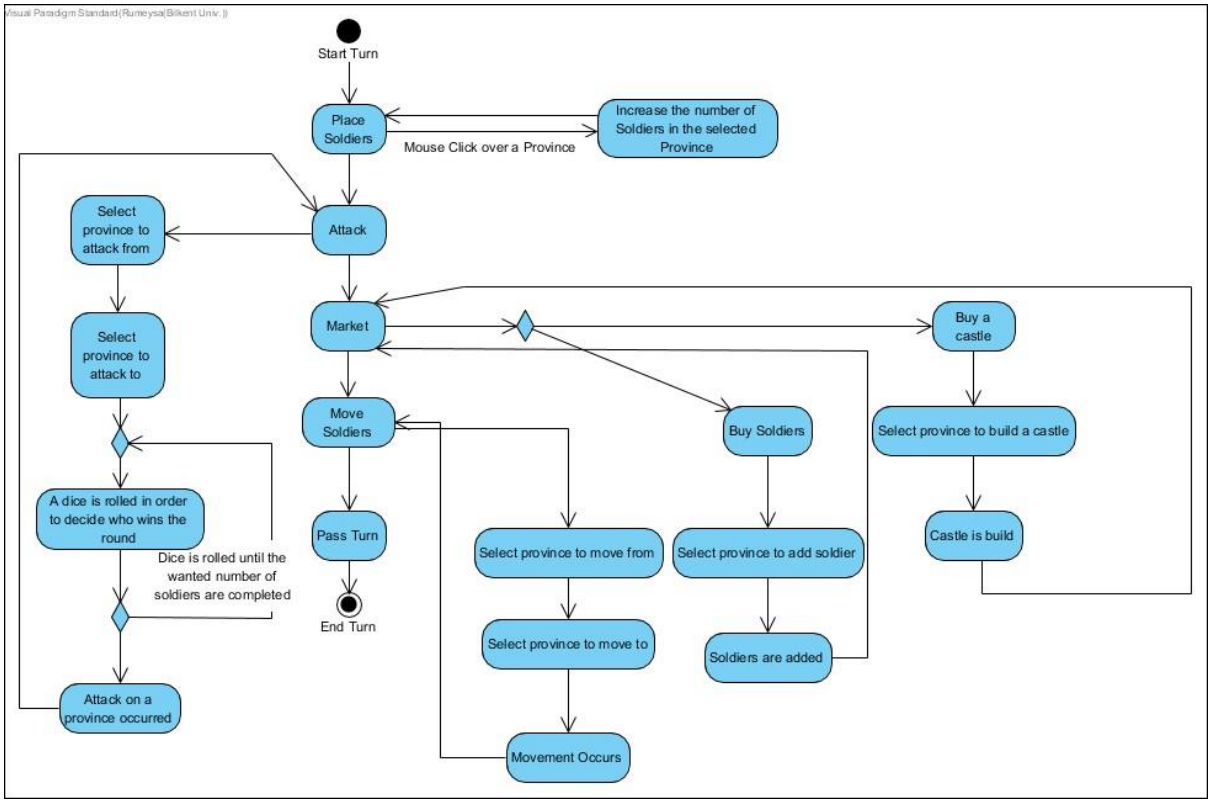
3.5 Object-Class Diagram

Since the process for this project is dynamic, the changes can be done even after the second iterations. After the second iteration, we realize that some of our methods were unnecessary and can be done more efficiently. So, we change the structure and the methods of the classes. Overall, our UML diagram is improved as follows:



3.6 Activity Diagram

After the second iteration, we decided that the system for buying soldiers and moving soldiers were a little bit vague since there was no order between them. We decided to add an order so our activity diagram for a turn has changed as follows:



4. Improvements

4.1 Fortune Wheel

Normally, after conquering a province, users were only given a bonus card. In order to make the game more interesting and exciting we have added a fortune wheel. A user can decide either to turn the wheel or just pick a bonus card. Our fortune wheel is represented below.

5. Lessons Learnt

When our team was implementing the game, the experience revealed many new lessons. First of all, we have learned the importance of partitioning the work and act as early as possible for the project. We also had an experienced in which we have to examine pros and cons of the different object oriented languages and choose an appropriate programming language for our project implementation. We also used forward engineering for the project where we have constructed schemas and diagrams for the project and we have written code according to the outline. Even though this is a favourable application, there were changes have been made since our opinions and specifications changed along the progression of the project.

Another lesson we have learned is the importance of using GitHub. Because of the lack of experience and knowledge, the system had slowed us down. It would be better for us to know the system beforehand.

Implementation phase:

Before the implementation of the game, our team had to decide which language to use. In this decision our team determined the programming language together in a brainstorming activity in which all members identified their objectives and reasoning. At the end, a decision that everybody agreed on was made according to opinions of the group members. A functionality we have decided to implement was a network system for multiple player however none of our group members were familiar with the concept. As a result, at the beginning we have decided to divide the tasks according to the time frames and gave specific people specific tasks in order to finish the project before the provided deadline. We have choose the set small goals for every week and medium goals for the months and added or deleted tasks after each goal according to the requirement.
The implementation was divided into the following tasks:

- Implement a basic map to get the coordinates correctly
- Implement a zoom in and out mechanism
- Come up with a UI design and basic functionalities
- Make soldier and castle objects draggable
- Implement objects and methods of the GameMap
- Implement independent classes like Player, Bonus Card and Dice
- Implement the regions of the WorldMap
- Implement a network system
- Add sound effects
- Implement the lowerPanel of the Risk Game
- Debug and test the Risk Game
- Implement Settings screen
- Implement how to play screen
- Implement credits screen

The distribution of the tasks have been done with respect to ability and desire. To summarize, our project had 3 main goals. First goal was to decide the functional and nonfunctional requirements of the game, we have decided the new functionalities and how to implement them. Second goal was to implement the basic functionalities of the game. Third goal is to implement the network system of the game.

6. User’s Guide

6.1 System Requirements & Execution

“Risk” will have an executable .exe extension. We will provide corresponding libraries for the user in installer so that they can run our game in a Windows Machine

6.2 Build Instructions

The game can be compiled with an IDE (preferably Visual Studio) by following the below instructions:

- Download the git repository
- Create an empty project with the name Risk in your IDE.
- Copy the folders you have downloaded into the folder for your empty project
- Change the project property in C/C++ part. Add the include folder of sfml folder to “Additional Include Directories”
- In input part of the linker part, add the following dependencies to “Additional Dependencies”: sfml-main.lib, sfml-audio.lib, sfml-window.lib, sfml-graphics.lib, sfml-system.lib, sfml-network.lib
- In the general linker part, add the lib folder of sfml folder to “Additional Library Directories”
- Then copy the files that ends with 2.dll in the sfml/bin folder.
- Click run in your IDE.

6.3 How to Use

6.3.1 Become a Host

In order to be able to play the game multiplayer from different computers, one should click “Create Game” button and set the settings of the game and became the host. Once a host exists, other players can join the game from the same network.

6.3.2 Join a Game

Players can join a game if a host already exists. They can choose a game from available servers list and they determine their nicknames. Than they can join a game.

6.3.3 Start the Game

The host can start the game if the wanted number of player is completed. If the game is going to be played from one computer, the player doesn't have to wait for other users to attend, s/he can start the game immediately.

6.3.4 Play the Game

After the settings are done and the number of people to play is reached, players can start playing the game and directed to the game page. Before starting the game, reading rules is recommended. “Risk” is a turn based game. In the beginning of the game turns of the players will be selected randomly. Starting from the first player, each player will choose their territories one by one. After every territory belongs to a player, game with start with the player who started first in the beginning. In a turn, firstly, player will collect soldiers and place them on the map to the territories that belongs to the player. Secondly, player can choose to attack other territories with a unlimited number of attacks. If the player conquers a territory a bonus card will be given to the player. Afterwards, player can choose to activate the bonus card for the next turn or can buy soldiers and castles. In the end, player will change the placements of the soldiers or add new soldiers then the turn will be finished. Turn must be completed in the respective order some stage can be passed without any action however ordering can’t be change. After one player conquers all the provinces, the game ends and that player wins.

6.3.5 Change Settings

User will be able to access the settings from the Main Menu. User will click to the Settings to change the Volume, Music and will be able to control the Full Screen options.

6.3.6 See Credits

User will click on the “Credits” button on the main Menu to see the credits. Credits will appear on the new window.

6.3.7 How To Play

The user will be informed by a tutorial which will be either in video format or gifs. Also, the rules of the game will be written.

7. Contributions

Merve Kılıçarslan

- Analysis Report- Dynamic Models (Iteration 1)
- Design Report (Iteration 1)
- Final Report (Iteration 1)
- Analysis Report (Iteration 2)
- Design Report (Iteration 2)
- Final Report (Iteration 2)
- Demo Slides (Iteration 1)
- Demo Video (Iteration 2)
- Managed GitHub repository
- Implement the regions of the WorldMap
- Added sound effects

Osman Burak İntişah

- Analysis Report - Functional and Non-Functional Requirements (Iteration 1)
- Design Report (Iteration 1)
- Analysis Report (Iteration 2)
- Design Report (Iteration 2)
- Implemented the network system for the game

Rumeysa Özaydın

- Analysis Report- Use Case Diagram and textual description (Iteration 1)
- Design Report (Iteration 1)
- Final Report (Iteration 1)
- Analysis Report (Iteration 2)
- Final Report (Iteration 2)
- Demo Slides (Iteration 2)
- Demo Video (Iteration 2)
- Implemented some functions for the logic of the game
- Implemented credits screen content

Ahmet Serdar Gürbüz

- Analysis Report- User Interface (Iteration 1)
- Design Report (Iteration 1)
- Analysis Report (Iteration 2)
- Design Report (Iteration 2)
- Class design and system architecture design
- Implemented an algorithm in order to link the provinces
- Implemented functions for the logic of the game
- Fixed various kinds of bugs
- Designed the map for the game

Elnur Aliyev

- Analysis Report - Object-Class Model (Iteration 1)
- Demo Video (Iteration 1)
- Design Report (Iteration 1)
- Analysis Report (Iteration 2)
- Drag and drop interactions
- User interface for game screen and menu screen
- Fixed several kinds of bugs
- Screen transitions