Bilkent University

Department of Computer Engineering

CS 319 - Object-Oriented Software Engineering

Iteration 1 - Final Report

Project Name:	Risk Game
Group No:	1D
Group Name:	Risk Takers
Group Members:	Yigit Kutay GULBEN
	Nurlan FARZALIYEV
	Esad Burak ALTINYAZAR
	Burak YENI
	Burak MUTLU
	Anar HUSEYNOV
Supervisor:	Eray TUZUN

Contents

1. Introduction

Before all reports we have played the online version of the game then, decided to implement some classes, with a few attributes and methods, to make it easier for ourselves to understand the whole process and derive some diagrams from it. Basically, we have done a reverse engineering. However after a while things got a little bit complicated, so we decided to draw the rest of the diagrams first and implement the rest of the code according to the diagrams. During the group meetings we first wrote down all possible tasks then assigned roles to those tasks. After all, each group member has chosen some of those roles and then the rest of the implementation started. In order to collaborate synchronously we have used GitHub.

2. Implemented Functionalities

For the purpose of testing our model classes designed through previous report's diagram, we implemented the basic version of the game, in which only primitive but essential mechanism of the game are present. But in visual sense, the standing of game is lack of visual saturation in its current state since our priority is not design and modification of the visual components in very first phase of the project. That is, our system supports the following functionalities with a simple UI:

- The model implementation of player interactions among the game like moving units, capturing other territories
- The implementation of game-specific objects like constant map territories and their properties within each other, and to perform game mechanics over them, their underlying structure
- Loading the game which is the instantiation of the game objects with specified data and updating the data of such objects through incoming signals from UI elements

- UI components with a basic level of interaction and visual appearance in simple manner
- Such basic gameplay and level design that it helps us to implement and modify essential components
- Other image analyzer and functional programs to generate our game-specific data like constant map visual objects and their properties

3. Design Changes

Since we started to implement the game before the design issues in the reports, the first implementation included some deviations from the right way of implementing relations. While modelling the game through UML diagrams and techniques, we used such inspirative implementation to guide us on how to design relations. After modelling the game on the reports, firstly we should’ve fix the wrong implemented and designed parts on the code. After that, as a group, we progressed into much better version of the game and we tried to comply the diagrams while implementing further modules. As the game being progressed into an advanced version, the implementation of game started to fit into an stable state but also immutable form so that inserting additional features into game became much complex. So, we came up with some design changes as a solution for this issue and tried to break such complexity of the software we produced.

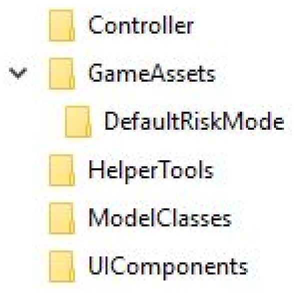
3.1 Extended Dependencies

While we designing our class relations regarding with which objects included in the game, we had a thorough analysis of the game and we established our classes’ relation over such analysis in well designed manner. That's why the implementation of game did not deviated from the designed properties in UML diagrams. That is, we did not face with the need for changing proposed relations as implementing the game by complying our design. Only modification applied differently from design is that extensions of some base classes with child and specific-use purpose ones. For instance, we have class named as Territory, Card and TerritoryGraph for the implementation of game essentials but we realized that we may have different map and game mode assets as a furtherly added feature and also we had to define constants for map and mode specific relations. Such constants that it may indicate whether there is a connection between any territory within TerritoryGraph for the map selected. That is, we have to define the unique properties of the modes and maps in a specialized version of base classes. Currently, DefaultRiskMode, DefaultRiskTerritory, DefaultRiskCard is used for handling such issues. Also, the base classes for those converted into abstract-defined ones since they will not be used for any instantiation but will help a lot for implementing not map or mode unique but common operations with the abstraction they provided. If there is furtherly added features as being alternative for their current ones like different map and its requirements, they will extend the base classes which are ready to use for same purposes, to shape them in same manner.

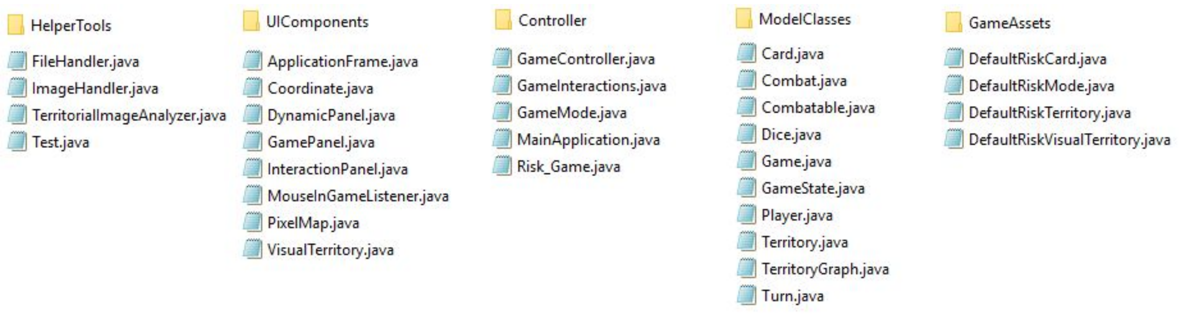
3.2 Utility Functions

The essential mechanisms of the game implemented through model classes and their utility functions. Such utility functions includes the core implementation for game functionalities but again we need other utility functions and side-definitions while implementing those. In the matter of involving side-definitions, in its basic sense interfaces, it may look like such relation does not contribute at all and it can be implemented in other ways. However, defining interface gives us idea about what is designed role for such interface, and then, when we look the classes implements this interface, we easily comprehend what such object refers to in our model space. As being present in current implementation, we have Combatable and GameMode interfaces. But, we decided to convert GameMode into an abstract class for its further relatives but again it supposed to be interface-like abstract class. Additionally, we have some private sub-functions and inner classes with the purpose of helping the implementation of proposed design. Splitting the implementation into sub-components always constitutes fundamental technique of software development and it makes the things seem much clear.

3.3 Packages



```
Controller
GameAssets
    DefaultRiskMode
HelperTools
ModelClasses
UIComponents
```



```
HelperTools
    FileHandler.java
    ImageHandler.java
    TerritorialImageAnalyzer.java
    Test.java

UIComponents
    ApplicationFrame.java
    Coordinate.java
    DynamicPanel.java
    GamePanel.java
    InteractionPanel.java
    MouseInGameListener.java
    PixelMap.java
    VisualTerritory.java

Controller
    GameController.java
    GameInteractions.java
    GameMode.java
    MainApplication.java
    Risk_Game.java

ModelClasses
    Card.java
    Combat.java
    Combatable.java
    Dice.java
    Game.java
    GameState.java
    Player.java
    Territory.java
    TerritoryGraph.java
    Turn.java

GameAssets
    DefaultRiskCard.java
    DefaultRiskMode.java
    DefaultRiskTerritory.java
    DefaultRiskVisualTerritory.java
```

In order not to mix every segment of the program we divided the programs to sub-packages. For instance *UIComponents* package has only contains the programs that handles the user interface and *ModelClasses* package has the main game elements.

4. Lessons Learnt

Since we were doing a simple implementation and meeting when we were doing any changes on the implementation we thought it is not important to use any kind of tool that would help us collaborate. However, after sometime we saw that we had to make some changes but not all of us were able to make it to the meetings especially when there were some unplanned meetings. Even though we were planning to use GitHub in the second iteration because of same kind of problems mentioned above we decided to start using GitHub immediately. Additionally, as I mentioned before we did a reverse engineering. However, since some of the classes inherit some other ones and since the inherited classes were written by different people there were some misunderstandings in the prototype of some methods. Therefore we started drawing class diagrams and then implemented the rest of the classes according to diagrams. Which actually worked, and made things easier.

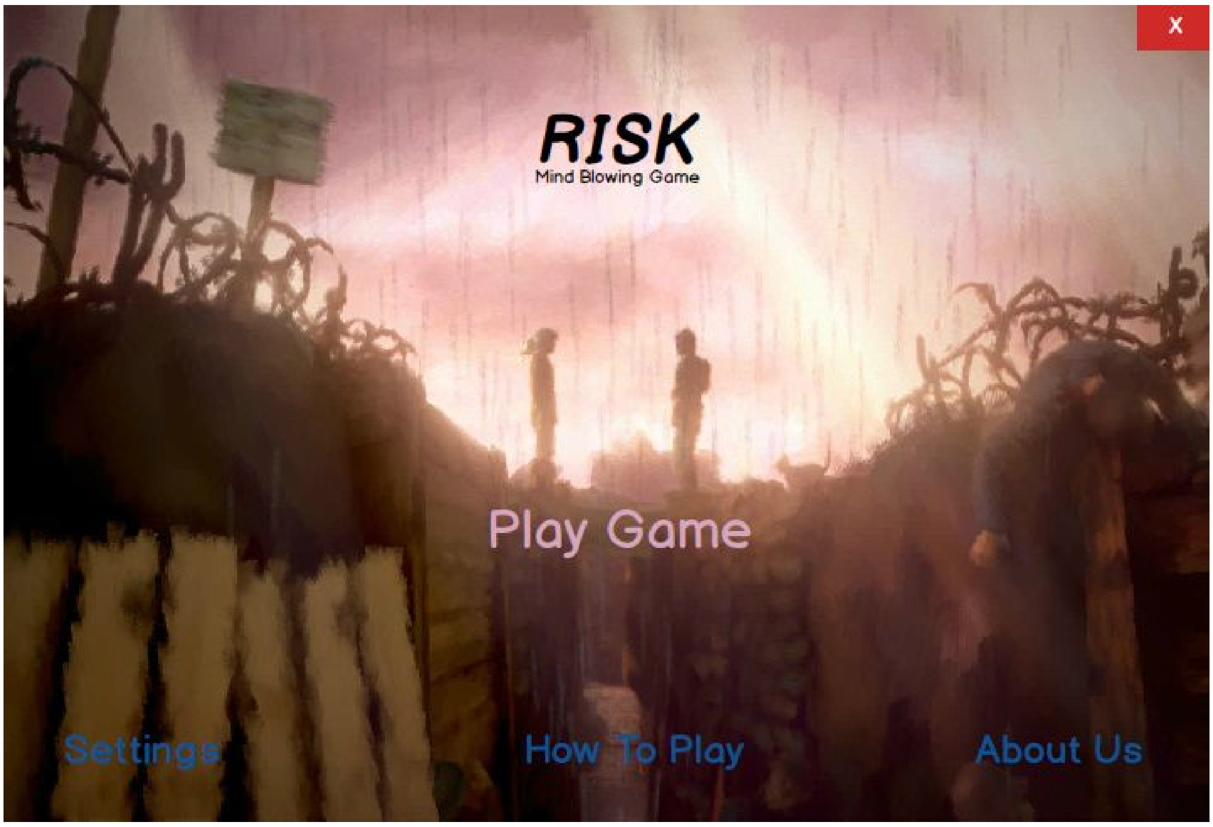
Through using the methods of CS319 we had chance to see the power of better communication among the group members and planning the group works. In such manner, we distributed the tasks among the related group members dependently on that person’s skills and then we progressed through the project towards a better version of the game.

5. User’s Guide

5.1 System Requirements and Installation

In order to activate the game, user needs to have the operating system that supports Java runtime environment. In terms of hardware requirements, both keyboard and mouse are needed. After that, all files for establishing the game should be downloaded. Memory requirement for installing the game is negligible (about 20 mb in Local Disk).
For running the game, firstly users should download appropriate Java for their operating systems. After establishing java on the system, users should execute RiskGame.jar file by double-clicking on it.

5.2 How to Use



5.2.1 Start a new game

In order to start the new game, user should firstly click on “Play Game” button. After that, player should choose the option solo or multiplayer. In solo option player will play against the AI and on the multiplayer option players will play a turn based game. After the choosing option new game will start.

5.2.2 Start a saved game

For starting to the previously saved game, user should firstly click on ”Play Game” button and then “Load Game” button in new screen. In that case, user will start to the most recent saved game.

5.2.3 Pause the game

In order to pause the running game, user can click on the “Pause Game” button in the left upper side of the screen. In that case, game will stop and another screen with in-game menu will appear.

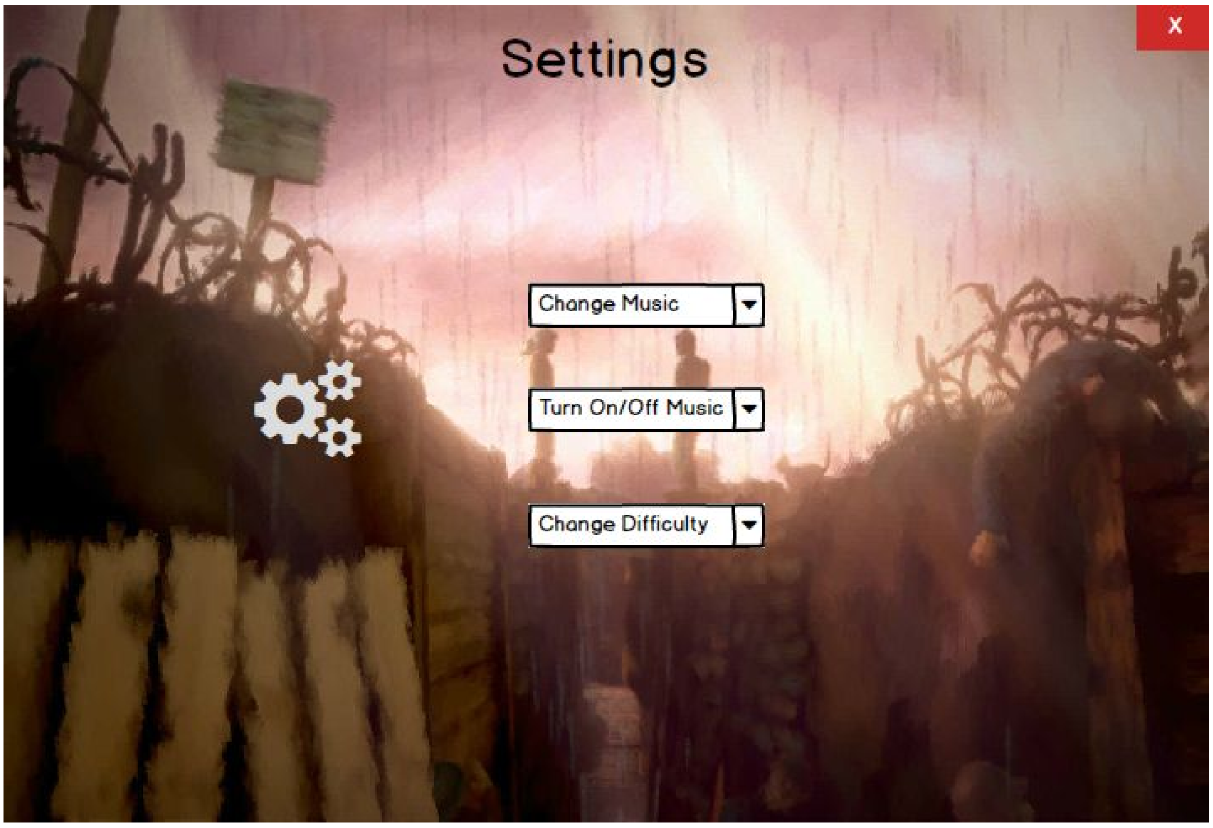
5.2.4 Resume to the paused game

After pausing the game, in-game menu will appear. In order to resume to the game, users just need to click on “resume game” button on the top of in-game menu. In that case, game will start from where it was paused.

5.2.5 Exit from the game

To exit from the game, player should firstly pause the game. After that, he should click on “Exit Game” button in order to leave the game. The game will be saved automatically after player exits from the game.

5.2.6 Go To Settings



User may go to settings menu from main menu or in game menu. User may change the game’s difficulty level, change music and change volume.

5.2.7 How to Play

In this section users will be informed on how to play the game, nothing related to hints will be given there. The information will be only about how to move units to the possible areas, explaining the cards and basically what is supposed to when the game starts.

5.2.8 See Credits

User may go to credits in the main menu of the game. In the credits sees the information about the producers.