CS319 - Object Oriented Software Engineering

Project Final Report

RISK

Group 2-B

Deniz Çalkan - 21703994
Irmak Akyeli- 21803690
Metehan Gürbüz - 21602687
Murat Sevinç - 21702603
Şebnem Uslu- 21802068
Yusuf Alpdemir - 21803035

1. Introduction

In the implementation of our game we decided to divide the work to three: server, UI/client and server classes. The server part focused on the implementation of our game on the real time working server remote JSP server which is provided by Amazon EC2 (Amazon Elastic Compute Cloud). The client part worked on all the user interfaces and their controllers which also manages the game play by calling the methods provided by the server classes. The server classes are where the main body of the needed functions of the games are and all three of them operate together to ensure a smooth online RISK game play. In our current state all three of the parts are complete separately and the connections between them are established. However, we are still trying to solve all the problems and mismatches created by the connections in an ongoing process. This is a bug solving process that would presumably continue even after the game is finished and launched to the public.

We used NetBeans for our implementations which had Github version control. One of the main problems that all of our team has faced was the process of opening the project as we had a problem with the java version differences. However, we managed to fix them one by one helping each other and worked simultaneously on the project thanks to Github. Everyone had his/her own objectives and we divided the work equally between us by classes or even methods. Yusuf implemented the real-time server, Şebnem and Deniz worked together on the UI/Client side and Irmak, Murat and Metehan did the server classes and helped others. Everyone worked together on the reports and diagrams. Our communication was mostly made on social media platforms like Whatsapp, Discord and Zoom. In the beginning of the semester we created a Discord server only for group members which enabled us to enter when we were working on the project anytime with a screen share option that we needed to solve many of our problems together. As we could not meet each week face to face due to current pandemia we used the discord server to have scheduled meetings as well as going in to just be online while working.

2. Lessons Learnt

We experienced the difficulties of forward engineering which is designing a project before actually starting to implement it. To be more precise, while implementing we started to see how what we have previously thought about the implementations and our assumptions of the classes and methods would be like were incorrect. All of the actual implementations were much more complicated, requiring more methods to access many different properties that we could not imagine before that needed to be added while some of the methods that we thought we would need were not necessary at all. We have realized how we overengineered some unnecessary places while leaving blank spaces for the important gameplay parts. We believe that the problems and shortsightedness that we face are caused by the lack of experience on the subject of designing and coding a game. We took the step for the forward engineering which we will need in the future and hoping for getting better at it.

In addition, one of the biggest things that we have realized was that the game play was not operated by the Game Engine class but from client controller classes. The reason that we have believed that the Game Engine was the main gameplay was that it was the middle of our biggest Façade design pattern that united server classes, having access to all of them directly led us to believe it would be the main class but we realized that how the game should operate from the client side as it is in interaction with the users. Therefore, for the purpose of not changing our façade design pattern, we have turned to a Game Engine class that operated the control and unity of the server layer while having all necessary methods to let the Client layer access every function and property of the Server layer to operate the game.

Aside from the things that we have discovered on our project, we have learnt how to communicate with each other effectively that we lacked in the beginning of the semester. We improved little by little with each reunion. This was a crucial point as working on the same project together, we needed the communication in order to advance rapidly. Another benefit of the communication was solving the problems that we had. As we were working on the same project in the same working environment, we have all faced similar problems in integration and the help we got from each other was much more significant and fast then searching online which led us to communicate more often and more efficiently.

3. User's guide

3.1 Running The Game

a) System Requirements:

Our risk game is runnable in any environment that has JRE (Java Runtime Environment).

The game requires a minimum 256 MBs of RAM and 32 or 64 bit operating system.

b) **Installation**:

Our game will be a single .jar file, therefore, users will only click on this file to open and start the game.

3.2 User's Manual

Goal of the Game

Conquer the world by capturing every territory on the map!

Setup

- Each player is assigned to a specific color to represent them throughout the game.
- Depending on the number of players, each player has given assets.
  > If there are 3 players , each player gets 35 assets.
  > If there are 4 players , each player gets 30 assets.
  > If there are 5 players , each player gets 25 assets.
  > If there are 6 players , each player gets 20 assets.
- Each player rolls a dice.The player who rolls the highest number starts placing their assets first. Then other players start placing their assets on unoccupied territories according to their rolled numbers.
- After all territories are occupied each player adds additional assets to their occupied territories and there is no limit for adding assets to a single territory.
- When there are no armies left, risk cards are shuffled and the first player starts the game.

GamePlay

Each turn consists of 3 stages:

1. Getting and placing new armies.
2. Attacking.
3. Fortifying position.

1.Getting and Placing New Armies

At the beginning of each turn players receive new armies based on the number of territories they occupied, value of the continents they hold , value of the matched set cards and other bonuses like the number of capitals they hold.

Territories

Players receive armies according to their occupied territories divided by three. Players can place their armies on their already occupied territories. Players will receive at least 3 armies even if their occupied territories are less than 9.

Continents

Players receive armies for each continent they hold. In order to hold a continent, players must occupy all the territories on the continent. Each continent has its own bonus points and it activates after the player holds a continent at the beginning of his/her turn.

The number of extra armies for each continent:

- Asia: 7
- North America: 5
- Europe: 5
- Africa: 3
- South America: 2
- Australia: 2

Capitals

Each player chooses a capital at the beginning of the game after all the initial soldiers are placed that cannot be changed during the game session. This capital gives 2 bonus points while the player is still controlling the territory. If a player attacks and invades another player's capital, it gains the bonus points given by the capital.

Risk Cards

At the end of a turn in which the player captures at least one territory, the player gains one risk card. The main goal is to collect sets of 3 cards in one of the combinations:

-3 cards of the same type(Infantry, Cavalry, or Artillery)
-1 from each type(Infantry, Cavalry, and Artillery)
-Any 2 cards and a wild card

At the beginning of each turn players who have a set of cards that are one the above conditions can trade their set with armies. Each set has its own value in terms of troops.
Wild cards are like joker cards, if a player has 2 artillery cards, it behaves like the third artillery card and provides a set.

2.Attacking

When a player’s turn comes he/she can attack other players’ territories. The main goal of attacking is to capture an opponent's territory by defeating all the armies that are currently on the territory. The defeater of the battle is decided by roll of dice. The players may choose not to attack when their turn comes but can still fortify their armies.
In order to attack the player should satisfy the following conditions:

-The player can only attack a territory which is next to or connected to his/her territory.
-The player must have at least 2 armies in the territory which he/she attacks from.

The player can attack a territory till there are no opponent’s armies left. Also, the player can attack territories as much as possible and can switch from one territory to another during his/her turn.

Dice rolls

Players will roll two dice which have 10 faces that gives the player a total number of 100 different possibilities. The player that rolls the highest number wins the round and if there is an advantage to one of the players, the player uses the best of two rolls to fight.

Capturing Territories

When there are no opponent’s armies left where the territory that the player has attacked, he/she captures the territory. In this territory, remaining armies from the previous battle are placed. The player must leave at least one army on the territory that he/she has attacked from.

Ending Attacks

The player can end his/her attacks when desired. If the player has captured at least one territory he/she takes a one risk card.

Eliminating an Opponent

When the player eliminates all armies of another player, the second player is eliminated from the game and the first player gets all the cards that the second player has previously owned.

3.Fortifying Position

While always leaving one army behind, the player may choose to fortify their armies from only one territory into only one other territory that is his/hers. There are no requirements to fortify armies.

Winning

The player who captures all the territories wins the game.

3.3 Gameplay

Main Menu Screen

The main menu meets the user when our game is started. There will be 3 buttons in main menu:

- **Help:** On the left top corner, there will be a help button which opens a pop up. It contains information about the goal of the game and the game setup. When the help dialog is closed, the user gets back where he or she was.
- **Play:** Play button is the starter button of the game. It forwards the user to the role selection screen.
- **Quit:** This button closes the game immediately.

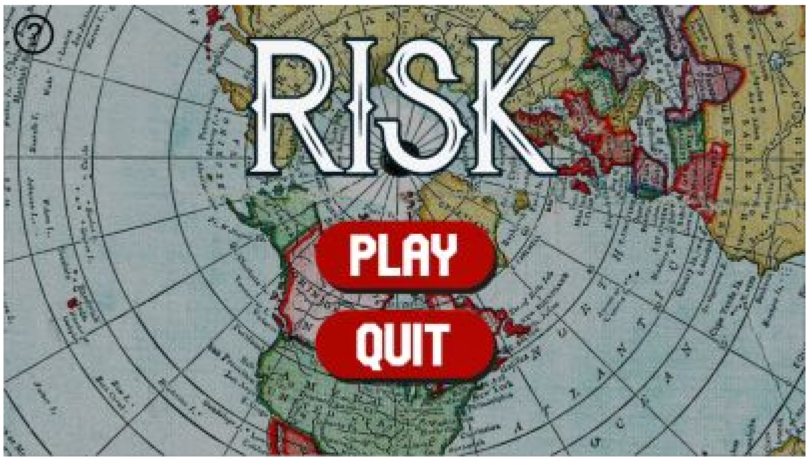
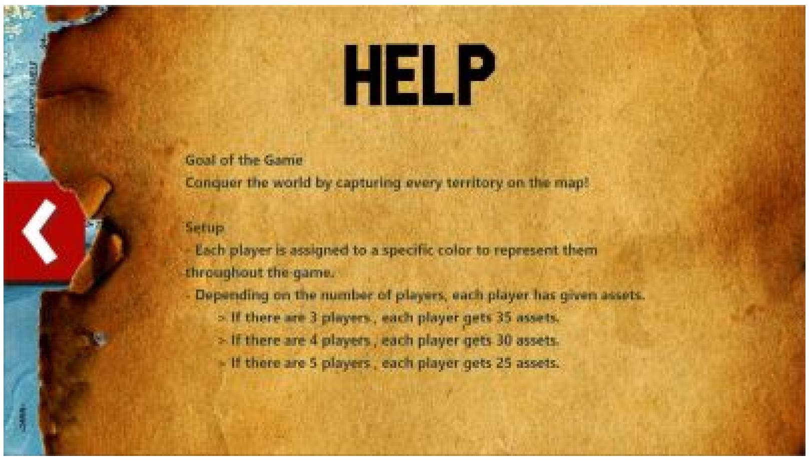


figure 1: Main Menu Screen


HELP

Goal of the Game
Conquer the world by capturing every territory on the map!

Setup
- Each player is assigned to a specific color to represent them throughout the game.
- Depending on the number of players, each player has given assets.
  > If there are 3 players , each player gets 35 assets.
  > If there are 4 players , each player gets 30 assets.
  > If there are 5 players , each player gets 25 assets.

figure 2: Help Screen

Role Selection Screen

This is where the user picks his/her role. There will be two options: Host and Join. When the host button is clicked, the user is directed to the game configuration screen. When the join button is clicked, the user will be directed to the join room screen.

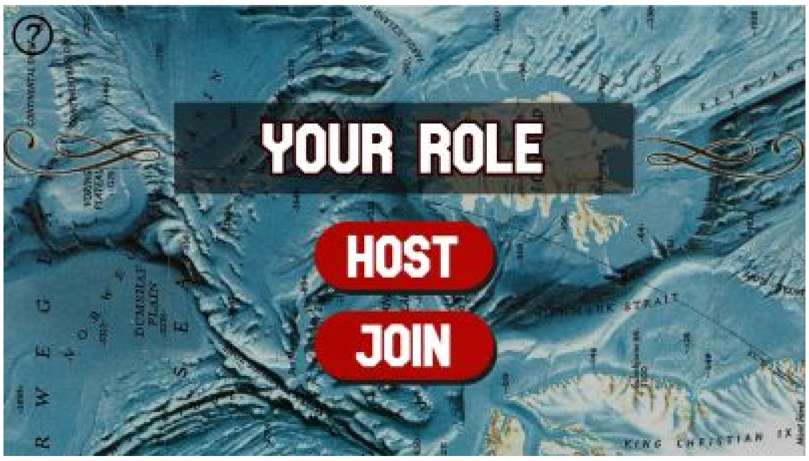


figure 2: Role Selection Screen

Game Configuration Screen

In this screen, two bars and two arrow buttons will be shown. One bar will contain a toggle button that will determine the troop distribution of the game; automatic or manual. The other bar will contain a text field for the user's name. This field will show an error pop-up if there is no input provided, or the user name is larger than 14 characters when the forward button is clicked. Backward arrow will return the user to the main menu, and the forward arrow will direct the user to the room screen.

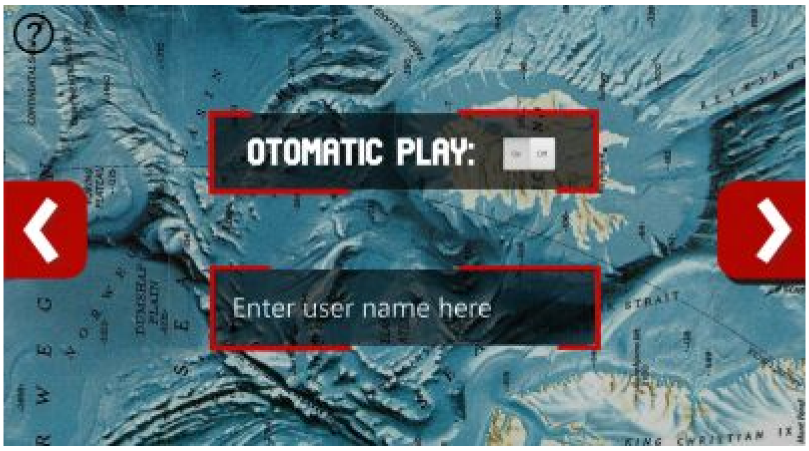
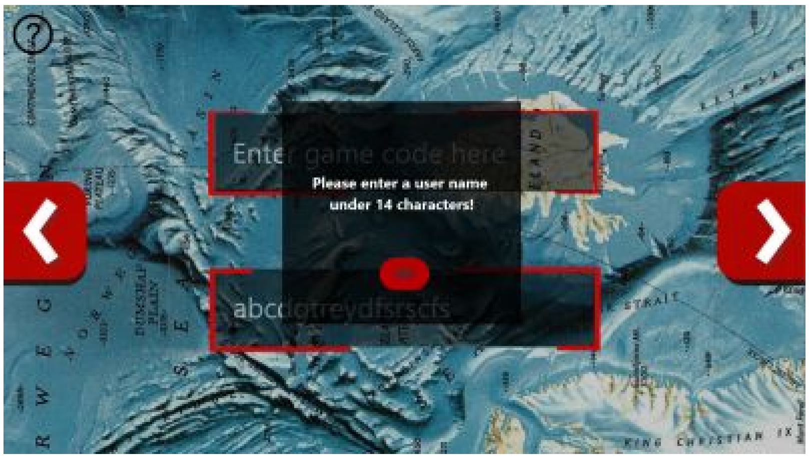
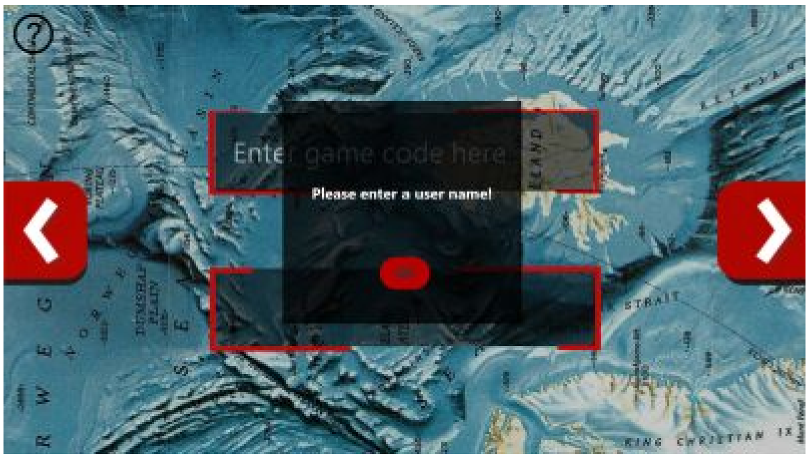


figure 3: Game Configuration Screen



Please enter a user name under 14 characters!

figure 4: Error pop-up 1



Please enter a user name!

figure 5: Error pop-up 2

Join Room Screen

The bar at the upper part will contain a text field for game code. This game code will be used to connect the user to an existing room. Bottom bar will be used to get the user name. The backward button will return the user to the main menu. And the forward button will check whether the game code and the user name is valid or not. Accordingly, the user will be forwarded to the room page, or one of the error pop-ups in figure 4-5 or 7-8 will be shown.

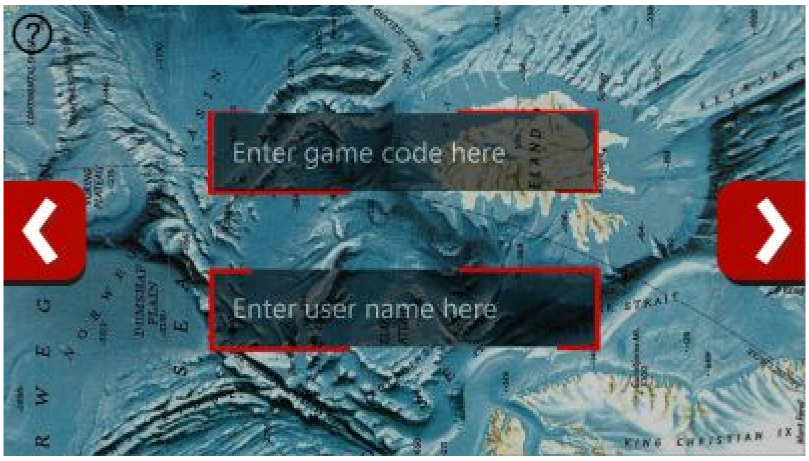


figure 6: Join Room Screen

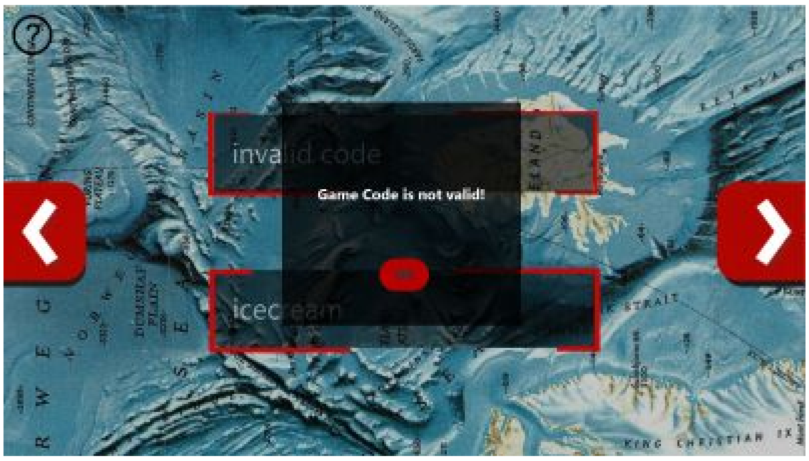
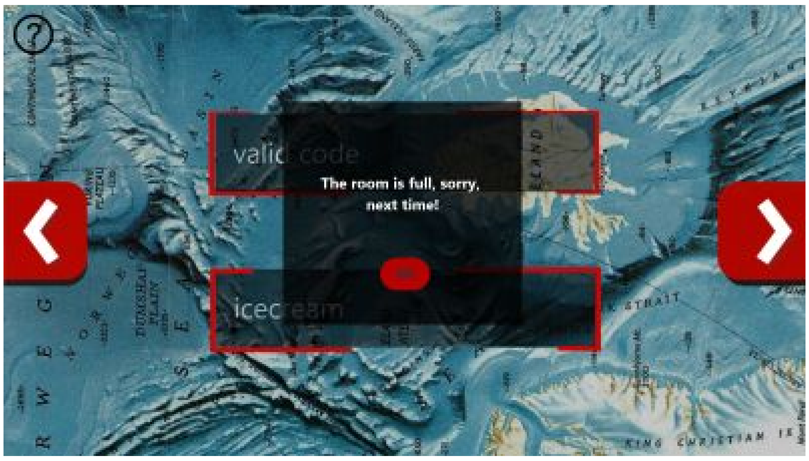


figure 7: Error pop-up 3



The room is full, sorry, next time!

figure 8: Error pop-up 4

Room Screen

The room screen contains one backward button, one forward button and one text dialog that shows the current players' name in the room. The Backward button will direct the user to the previous screen (in this case either join screen, or game configuration screen). The forward button will direct the user to the board screen, to the actual game.

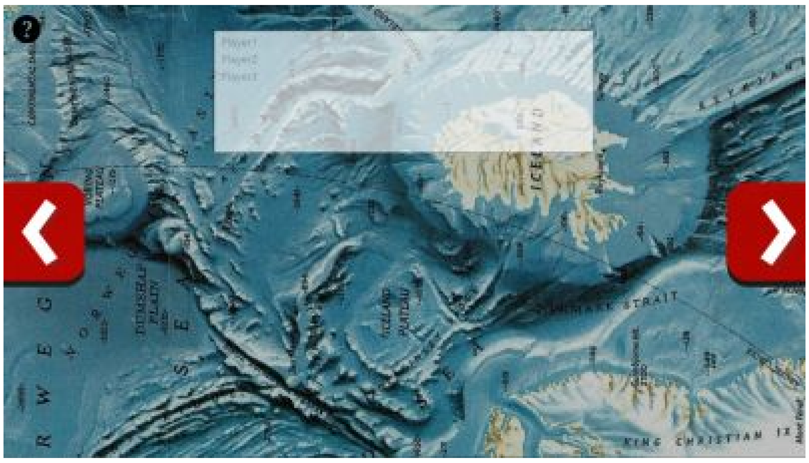


figure 9: Room Screen

Board Screen

In the board screen, there will be the game map, which consists of 47 regions. This will be a dynamic screen. Fortification, attack, reinforcement phases will be shown on this screen. On the right bottom corner, there will be a resign button which shows a resign dialog, if the player clicks on ‘yes’, the user is directed to the main menu. If the player clicks on ‘no’ the user will keep playing. On the left bottom corner, there will be a small text box that shows the turns of players and when the turn is the player itself, there will be figures to indicate which phase the player in. Also, there will be white card figures and this button will open car dialog to get extra troops. When the game is over, a big dialog indicating the winner will be shown. When pressed “ok”, the players will be directed to the main menu.

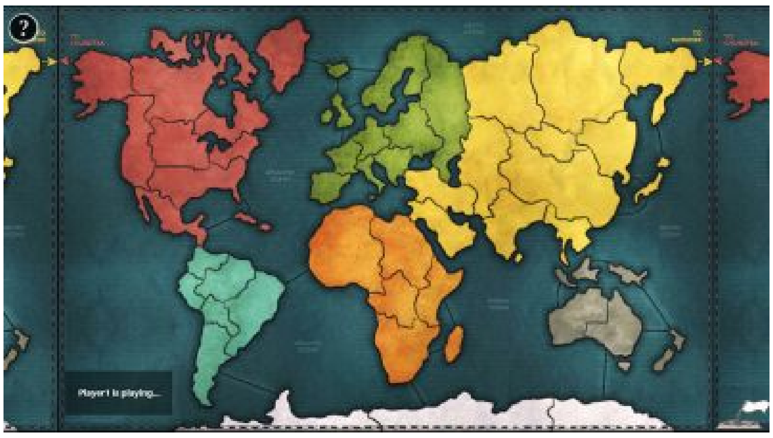


figure 10: Board Screen when somebody else is playing

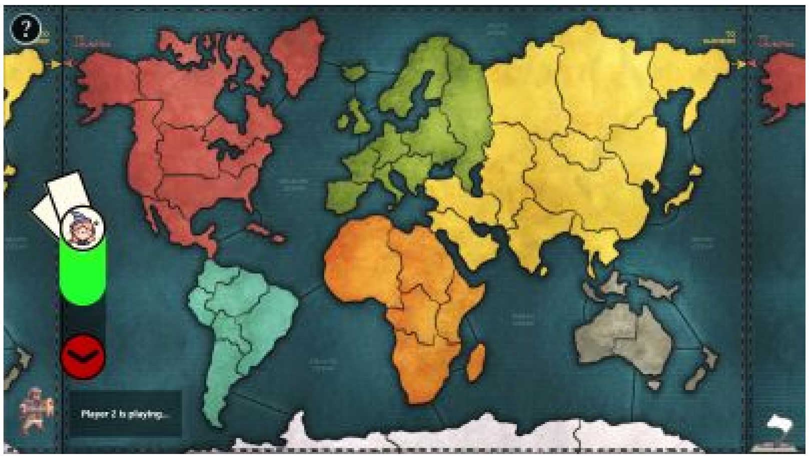


figure 11: Attack Phase of the player

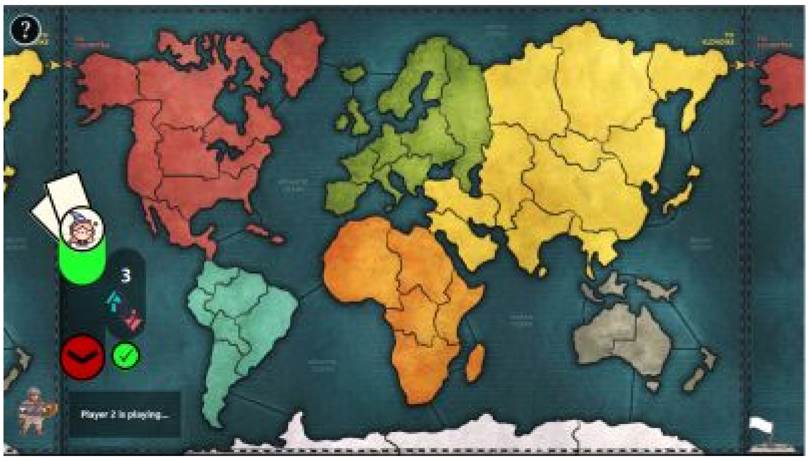

Player 2 is playing...

figure 12: Fortification Phase of the player

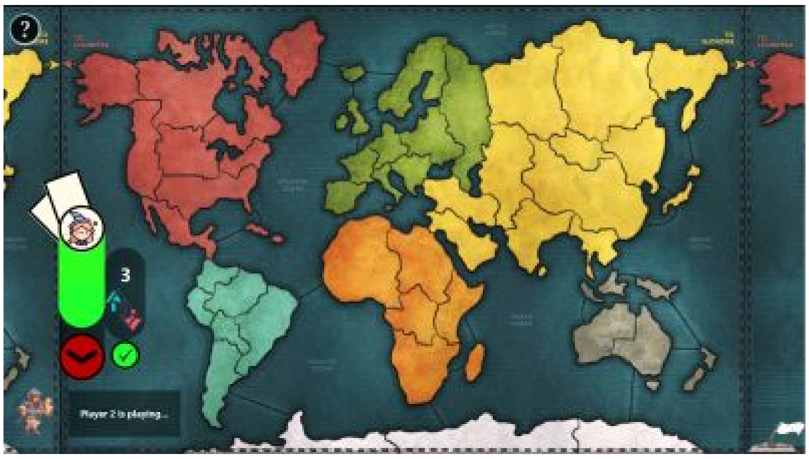
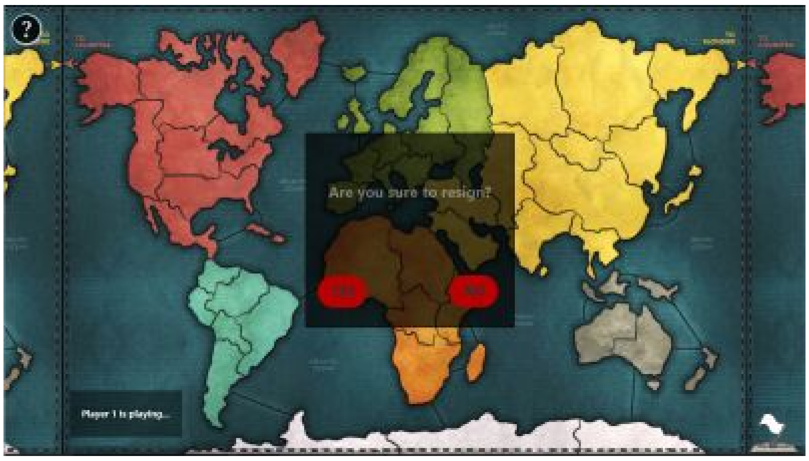

Player 2 is playing...

figure 13: Reinforcement Phase of the player


Are you sure to resign?

Player 1 is playing...

figure 14: Resign Dialog

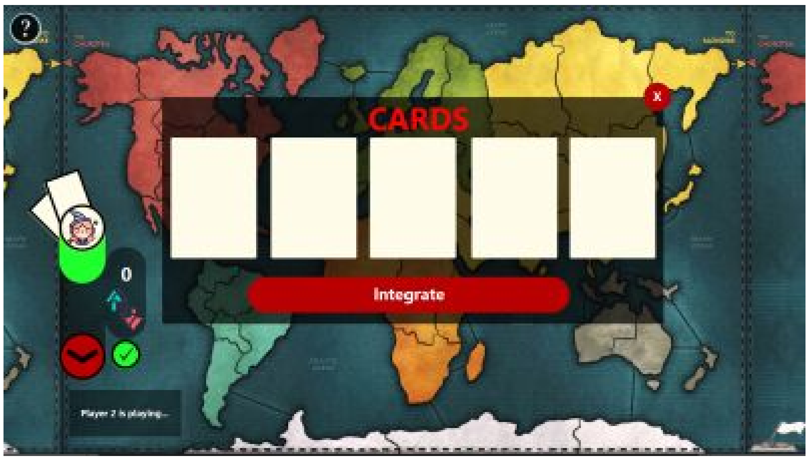


figure 15: Card Dialog

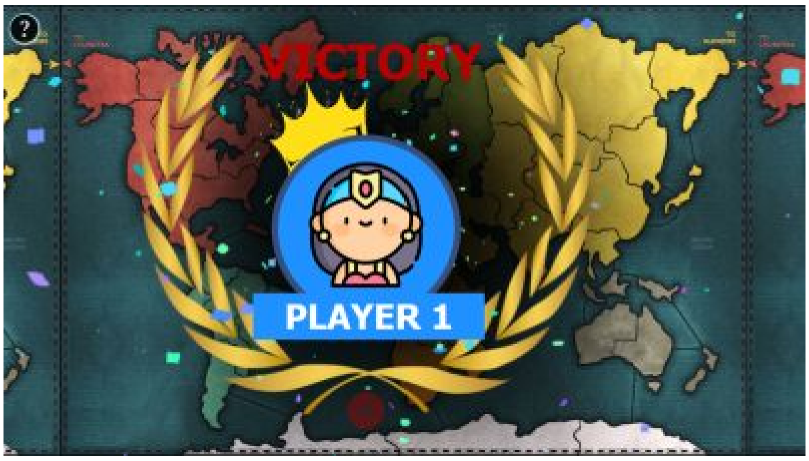


figure 16: Gameover Screen

4. Build Instructions

Since the server is running in the cloud as an Amazon Web Service, the client (player) only has to run the jar file and the application will be running as intended, getting all the information necessary from the server as long as the server is running. As an alternative the main class can be run after compiling the project using the Apache Netbeans IDE.

The IDE used to create the project was Apache Netbeans (Above version 12.0.0), so in Netbeans, opening and selecting the client project, right clicking and choosing “Build” will be enough to build the project. The version of JDK used is 11, however, the project was also tested in versions 10 and 15. Any additions can be added to the code and rebuilding will be enough to see the changes. Since our UI was made in javafx, the user should have a javafx sdk in case Netbeans IDE cannot automatically find the javafx sdk. If there are any changes made to the server project, “stubs” will need to be created to update the client side. Thanks to an automatic web service client created in Netbeans, these stubs are automatically updated.

The server project is made with Maven which exists in Netbeans. To modify the code of the project, after the modifications are made in the code, the project will be packaged into a .war file by right clicking “pom.xml” and choosing “package”. After this .war file is created, it will be uploaded to the server using PuTTY and overwriting the .war file in the server given by Tomcat. The version of Tomcat is 9.0.4 and the Java EE Web version used is 7.

For some users, the compiler in Netbeans may not see the javafx library. If this happens, adding the javafx sdk by choosing “libraries” under “tools” and showing the location of the javafx sdk. The version of javafx sdk used is 11.0.2. Another error we have encountered that sometimes happened, was that the stubs created from the server sometimes did not update after the methods in the server were modified. If such a thing happens, re adding the web service client should solve the problem.

5. Work Allocation

Deniz Çalkan:

- Worked on the functional and non-functional requirements, design goals, introductions and overview on the reports.
- Wrote a manual for the game.
- Created and implemented all the controllers, fxml files and the ViewFactory class of the client layer.
- Implemented all the user interfaces together with Şebnem.

Irmak Akyeli:

- Made a sequence diagram and worked on the reports.
- Made an attack package and implemented them on the player.
- Worked on the classes: Player, Region, GameEngine, Attack.* and Dice.
- Worked with other members to integrate their parts to GameEngine and added needed methods by the client part.

Metehan Gürbüz

- Made the state diagrams and worked on the reports.
- Made the Card class and worked on Player class.
- Worked on methods on the client side of the project.

Murat Sevinç:

- Prepared activity, system decomposition, and deployment diagrams in the reports.
- Software Architecture & Subsystem Services parts are provided for reports.
- Made DistributionFactory and Distribution classes using AbstractFactory design pattern.

Şebnem Uslu:

- Made the class diagram and worked on the reports.
- Created the first designs for the user interface and implemented them.
- Created and implemented all the controllers, fxml files and the ViewFactory class of the client layer.
- Implemented all the user interfaces together with Deniz.

Yusuf Alpdemir:

- Made the Use case diagram and explanation for reports.
- Implemented the server-client architecture and added the Façade class GameEngine for the project.
- Worked on and modified the classes: GameEngine, Region, Player, Attack package, Automatic package, ServiceTest and MainGUI for testing the server client connection.
- Helped with some controller classes and ViewFactory class on the client side.
- Added methods and a polling system to establish communication between server and client.
- Checked for errors and debugged the code, solving errors.