OOP Project

# Related image**Project Analysis Report: Iteration 1**

CS 319 Object Oriented Software Engineering  
Eray TÜZÜN  
Group 1-I: Private Moon Inc.

Project Group Members

• Doğa Oruç – 21602022

• Çağrı Orhan - 21503102

• Fazilet Simge Er - 21602358

• Kaan Ünlü - 21602622

• Ege Turan – 21502441

[Project Analysis Report: Iteration 1 0](#_Toc3146606)

[1 INTRODUCTION 2](#_Toc3146607)

[2.0 PROPOSED SYSTEM 2](#_Toc3146608)

[2.1 Overview 2](#_Toc3146609)

[2.2 Game Algorithm 3](#_Toc3146610)

[2.3 Server 3](#_Toc3146611)

[2.4 Game Objects 3](#_Toc3146612)

[2.4.1 Provinces 4](#_Toc3146613)

[2.4.2 Continents 4](#_Toc3146614)

[2.4.3 Dices 4](#_Toc3146615)

[2.4.4 Card 4](#_Toc3146616)

[2.4.5 GameBoard 4](#_Toc3146617)

[2.5 Gameplay 4](#_Toc3146618)

[3.0 FUNCTIONAL REQUIREMENTS 4](#_Toc3146619)

[4 NONFUNCTIONAL REQUIREMENTS 5](#_Toc3146620)

[5.0 SYSTEM MODELS 6](#_Toc3146621)

[5.1 UML Diagrams 6](#_Toc3146622)

[5.1.1 Use Case Diagram 6](#_Toc3146623)

[5.2.2 State Diagram 9](#_Toc3146624)

[5.2.3 Sequence Diagram 9](#_Toc3146625)

[5.2.4 Class Diagram 10](#_Toc3146626)

[5.2.5 Activity Diagram 10](#_Toc3146627)

[5.2 User Interface 10](#_Toc3146628)

[5.3 Screen Mock-ups 10](#_Toc3146629)

[5.3.1 Main Menu 10](#_Toc3146630)

[5.3.2 Play Online 10](#_Toc3146631)

[5.3.3 Play Offline 11](#_Toc3146632)

[5.3.4 Game Screen 11](#_Toc3146633)

[5.3.5 Settings 12](#_Toc3146634)

[5.3.6 Rules 12](#_Toc3146635)

[6.0 REFERENCES 12](#_Toc3146636)

# 1 INTRODUCTION

Risk is a table-top strategy game by Hasbro, whose purpose is the conquest of a political world map inhabited by different players at the start of the game [1]. Project Risk, done by Private Moon Inc., is an extended implementation of the original game.

Risk is a strategic board game of diplomacy, conflict and conquest for two to six players. The standard version is played on a board depicting a political map of Earth, divided into forty-two territories, which are grouped into six continents. Turns rotate among players who control armies of playing pieces with which they attempt to capture territories from other players, with results determined by dice rolls. Players may form and dissolve alliances during the course of the game. The goal of the game is to occupy every territory on the board and in doing so, eliminate other players.

In our version of the game, there are additional features we are trying to implement as a group, such as playing either single player or multiplayer. Furthermore, we are planning to add chat panel so that users can contact other users while playing online. Shortly said, this will be an extended version of the traditional Risk game, designed so that it can be played alone or with people from far away locations (unlike the original version, which requires a certain number of people to be strictly around the game board), and will have more gameplay content than the original Risk, such as new maps, and depending on time left after the key features are implemented, even more.

Although Risk is a strategy game and does not need much graphic, we are planning to implement well designed graphics and motion algorithms at certain actions. Thus, we can meet the expectations of the users more.

# 2.0 PROPOSED SYSTEM

## 2.1 Overview

We will implement desktop version of Risk game which runs on the Java platform and its properties are as follows:

* Each player registers the game server with a unique name.
* Each game is capable of having two to six players, and single player mode is available and can be played offline without internet connection.
* Chat panel to communicate
* Risk map
  + With forty-two provinces
  + Six continents
  + Edges that shows ownership of provinces
  + Routes between provinces, and continents
* Random card taking
  + There are 4 different types of cards (Soldier, Cavalry man, Cannon, Wild Card).
  + Player takes a random card at the end of his/her turn, if the player had captured at least one province on a single turn.
  + A player can keep maximum of 4 cards. If the player has total of 5 cards, he/she must trade cards.
* Trade cards
  + A player can exchange cards with soldiers
  + Player must collect either
    - 3 same type of card
    - 3 different types of card
    - Wild card may use as desired type so that 2 same type of cards or 2 different kind of card, and a wild card makes a valid triple to exchange
  + Player who has 5 cards must trade cards because 5 cards guarantee having a valid triple card to exchange.
  + Every valid triple card corresponds to number of soldiers to be taken. First trade gives the player 6 soldiers. After each trade, number of given soldiers increase. So, the next trader will get 10 soldiers rather than 6, and the number of soldiers taken per trade will continue to increase.
  + Users cannot see other’s cards, but they can see the numbers of cards the players have.
* Capturing and Defending
  + First of all, the player can only capture an area which is connected by a route or an edge, to at least one of the provinces he/she owns.
  + User must have more than one soldier on the territory he/she is trying to use the armies of, to capture another province.
  + Capturing and defending users roll dices for each soldier, and who rolls the lower dice loses his/her soldier. In order to capture a province, capturing player must beat all the soldiers on the desired territory.
  + There is no capturing limit so a player can attack several times on a single turn.
* Taking Soldiers on each round
  + Every player takes at least (number of province under control / 3) soldiers on each turn and place the soldiers wherever he wants.
  + If the player captures all the provinces in a continent, he/she will get bonus soldiers on each turn. Each continent has different soldier bonus.
* In order to win the game, a player must capture all continents.

## 2.2 Game Algorithm

Our game is played turn by turn. Hence, the game will continue until a player takes the whole world. Our algorithm is going to execute until a player won the game by checking winning conditions on each round. The algorithm must kick the player if the user has no more soldiers to play.

## 2.3 Server

Our version of Risk is played as multiplayer so need to have data transfers among users. Our solution we considered for this issue is using a server rather than a database. Logic of playing as multiplayer is all data that used by our game has to be synchronized on each user. Our way to do this creates a SocketServer and unites the users on that server for data transfer. This feature also allows us to create a chat on Server. When a user starts a new multiplayer game, user has to create a room which means SocketServer object is created to bind people. After constructing a SocketServer, players will connect the server as a Client. A user may decide how many players going to be in a game room so that server can deny the extra requests. After each turn, needed game objects is send to each client in order to update the game process. In order to provide synchronized data on each client, we are using multi-thread class.

## 2.4 Game Objects

Game objects are the visual components on game screen and users can interact with those objects.

### 2.4.1 Provinces

Provinces objects are territories on the Risk map. There exist forty-two provinces on the game map. Also provinces have ownerships that are seen on screen with user’s colours. Every province has a unique name. Only ownerships of provinces may change by capturing.

### 2.4.2 Continents

Province clusters generate Continents object with the total of six Continents with unique names. Players try to conquest an entire Continent in order to take bonuses.

### 2.4.3 Dice

Dice objects are traditional dice with the number on each side. When a player rolls a dice, random generator gives a number (1-6) and prints on the screen. The result of battles between users is determined by rolling dices.

### 2.4.4 Card

A card object is given to a player when a player captured a province. Card objects are exchanged with soldiers. A player only can see the other player’s cards numbers.

### 2.4.5 GameBoard

GameBoard object is the main connection with user. GameBoard object has all the objects that a user may want to interact.

## 2.5 Gameplay

After entering game, user is going to encounter with a screen with multiple choices. Choices are as follows

When a user enters the play button, panel offers 2 option

* Single Player: In single player mode user will play against computer.
* Multiplayer: Server will be created and other player will be waited for connect the server.

Next step is deciding which provinces to have turn by turn. After selecting provinces, users deploy armies on their provinces. Now game is ready to be played.

# 3.0 FUNCTIONAL REQUIREMENTS

Following properties are required to implement a proper traditional version of the Risk game

* Single Player Mode
  + We need to implement well design game objects and their connection so that we can render the images according to executing code on the background.
  + Single player requires computer decisions to play against computer. Thus, a decision-making algorithm will be implemented so decisions varies on different situations.
  + Game mechanics are essential for better game experiences and it cannot have any bug
  + Effects are fundamental components of a game. Thus, we need to have attractive effects.
* Multiplayer Mode

Making the game multiplayer takes the importance of the game to bigger point because Risk is a table game and played with group of people while communicating. Making the game multiplayer allows us to construct a customary game. We want to supply game experiences like traditional version of the game and perpetuate the emotions of the game which forces us to build a chat in game.

* Chat

There will be a chat panel on the game screen. People may communicate with other users. This allows us to reduce the risk of user to be bored and makes the game more playable.

* Settings

There will be background music on our application so users can listen while waiting for turn.

* + Audio Settings: Users can change the background track or just adjust the master volume of the playing track

Our application supplies game mechanic’s sound. For example, while capturing a province, effects and its sounds will occur. Users may want to change effect’s sounds.

* + Effect Audio Settings: Users can adjust or close the effect’s volume

User can also disable the effects of the game.

* + Effects Settings: Close the effects to get better performance
* Options
* Help

This screen will provide information to the user about fundamental rules of the Risk Game and common hints for success in game.

* Contact Us

Users can find contact information of our group. They can contact us in order to give feedback of the game or any bug they had found.

* About Us

This window will supply the game contributors to the user.

* Reset Game

Reset game function allows user to launch a new game by deleting current process of the game.

# 4 NON-FUNCTIONAL REQUIREMENTS

Non-functional requirements increase the significant of the game. Ways of well coding can be done by adapting important concepts. Non-functional requirements may help to meet the customer’s expectations more.

* Usability

Usability is most often defined as the ease of use and acceptability of a system for a particular class of users carrying out specific tasks in a specific environment. Ease of use affects the users’ performance and their satisfaction, while acceptability affects whether the product is used [2]. Thus, it is of great importance that every software practitioner not only be aware of various usability methods, the human-computer interaction community aims to increase the awareness and acceptance of established methods among software practitioners. Indeed, awareness of the basic usability methods will drive an Information Society for all.

Even if a new user of Risk game must play the game without any obstacle. User interface must be simple. Users may be familiar with the game even they are playing for the first time

* **Performance**

In order to get through more wide audience, game must run at optimal performance. Optimizations will be done in order to make the play more playable even in insufficient computers.

* Reliability

We need to handle with any possible crashes and bugs in game. We will test our game over and over again with the help of testers.

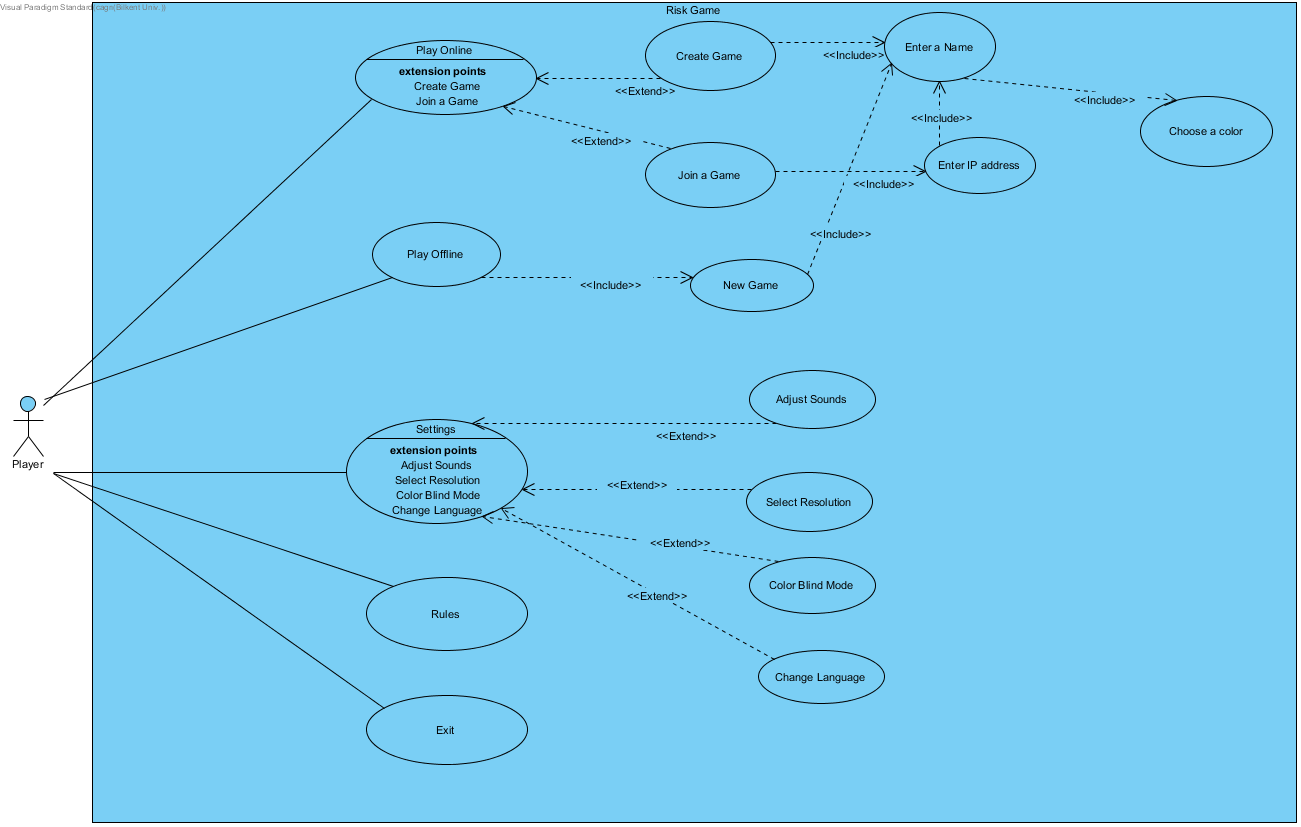
* Platform

Game must be run at any Windows 8 or 10 environments. Also Java environment is required to execute our program.

# 5.0 SYSTEM MODELS

## 5.1 UML Diagrams

### 5.1.1 Use Case Diagram



##### Use Case #1

Unique Name:

* Play Game

Participating Actor:

* Player

Entry Conditions:

* Player must press the Play Game button

Exit Conditions:

* Player must press Quit button from Menu in the game
* Player had won the game

Flow of events:

* Player has to click on Play Game button
* Game type selecting screen will occur on the screen

##### Use Case #2

Unique Name:

* Multiplayer

Participating Actor:

* Player

Entry Conditions:

* Player must press the Multiplayer button

Exit Conditions:

* Player must press Quit button from Menu in the game
* If there is no internet connection, system kick user out from multiplayer screen.

Flow of events:

* Player has to click on Multiplayer button.
* Game type selecting screen will occur on the screen either multiplayer or single player.

##### Use Case #3

Unique Name:

* Single Player

Participating Actor:

* Player

Entry Conditions:

* Player must press the Single Player button

Exit Conditions:

* Player must press Quit button from Menu in the game
* If player had finished the game.

Flow of events:

* Player has to click on Single Player button
* If you press the New Game button, system directs you to select a unique name
* After selecting a name, users have to choose the colour
* New Game will be launched by system.

##### Use Case #4

Unique Name:

* Create a Game

Participating Actor:

* Player

Entry Conditions:

* Player must press the Create a Game button
* Users have to be connected to the Internet

Exit Conditions:

* Player must press Quit button from Menu in the game
* If there is no Internet connection

Flow of events:

* If you press the Create a Game button, system directs you to select a unique name
* System will construct a Server to be connected by other users.
* After selecting a name, users have to choose the colour
* New Game will be launched by system.

##### Use Case #5

Unique Name:

* Adjust Settings

Participating Actor:

* Player

Entry Conditions:

* Player must press the Adjust Settings button

Exit Conditions:

* Player must press Quit button from Menu
* Any changes is adjusted and applied on settings

Flow of events:

* System shows user to accessible setting to adjust. Choices are as follows
  + Track Audio Settings
  + Effect Audio Settings
  + Effects Settings
  + Chat Settings

##### Use Case #6

Unique Name:

* Options

Participating Actor:

* Player

Entry Conditions:

* Player must press the Single Player button

Exit Conditions:

* Player must press Quit button from Menu in the game

Flow of events:

* System shows user to accessible options. Choices are as follows
  + Help which gives information and hints about game
  + Contact Us which allows users to contact with our group
  + About Us which lists the contributors of the game

##### Use Case #7

Unique Name:

* Quit

Participating Actor:

* Player

Entry Conditions:

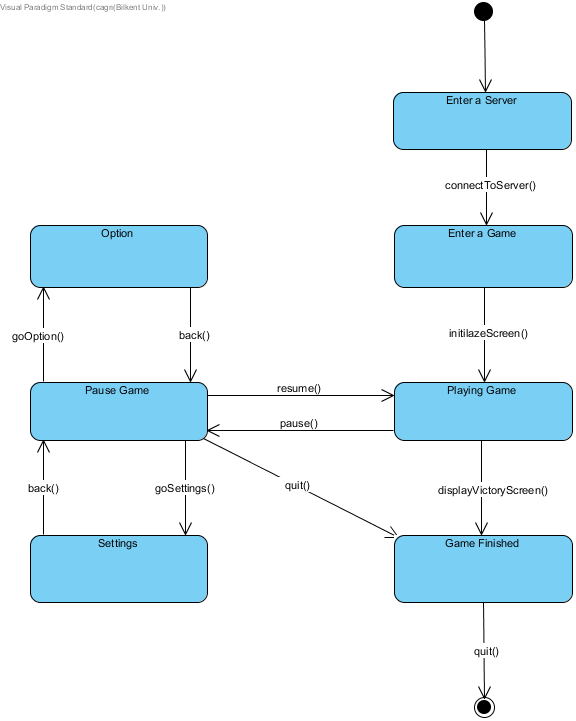
* Player must press the Quit button

Flow of events:

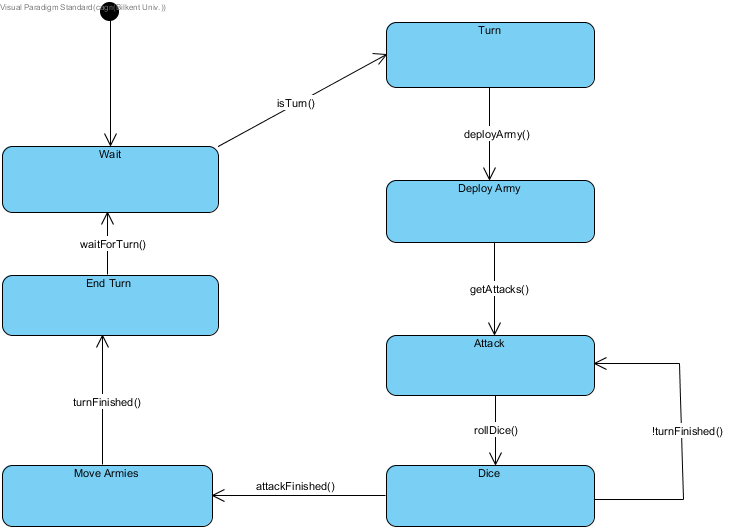
* Player pressed the Quit Button
* Program stops executing and close the game window

### 5.1.2 State Diagram

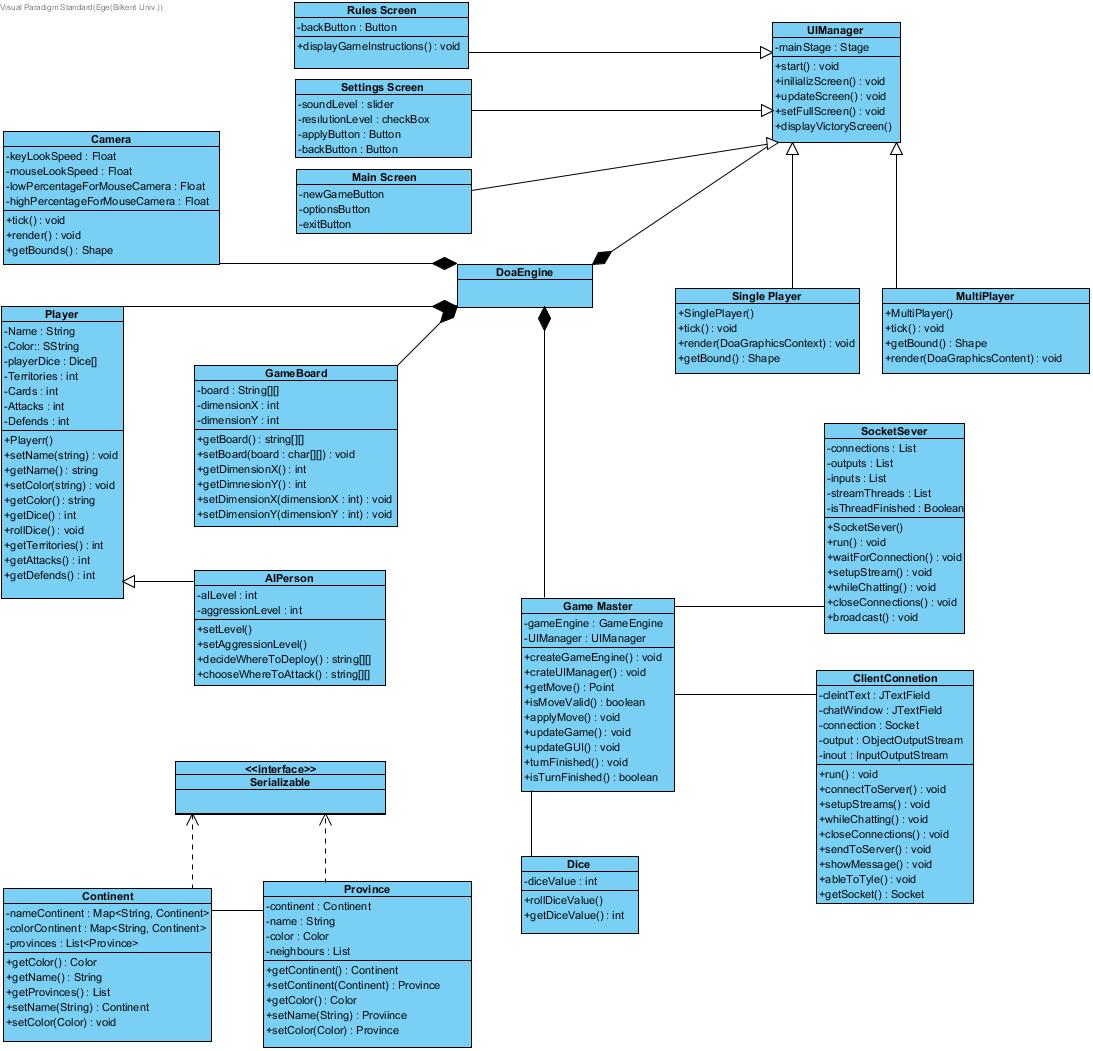
The diagram below is the state diagram of multiplayer screen. If a user wants to play as multiplayer, they have to enter valid IP address of player who creates the game with server successfully. After connecting server, users have to enter a valid name as last process before start playing game. GUI class renders a new game screen which means game is playable now. While playing games, users can pause the game. Pause screen provides options and settings to the users. If a player wins the game, victory screen will be displayed on the screen which means end of the game.



The diagram below is the state diagram of attack turn. When the turn comes, user gets armies to deploy to any provinces. User may do not want to attack so user can pass the attacking process. Otherwise, user has to roll a dice for battle. User can attack several times. After attacking session, player may want to move armies. When the turn finished, user has to wait for other turn.

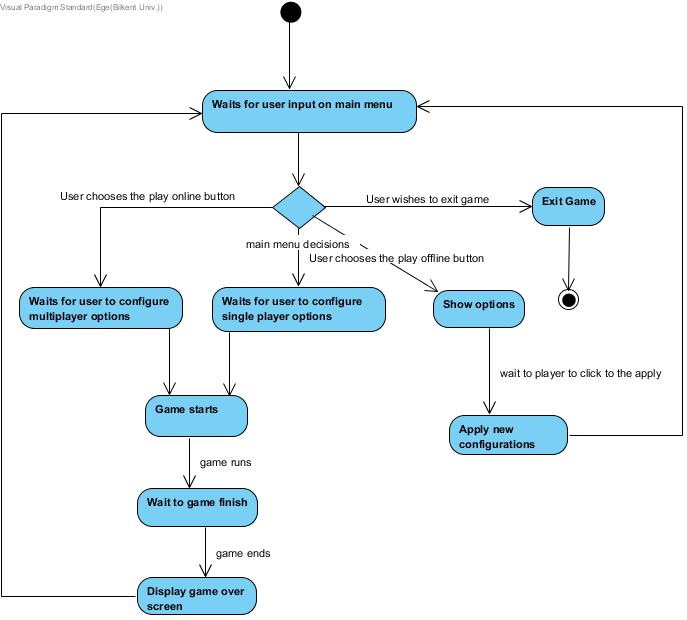


### 5.1.3 Class Diagram



* **Controller Class**
  + **Game Master**: This class controls the whole game system. It enables other classes to talk with each other and connects necessary components to maintain the game. It controls the game system by using the capability of DoaEngine while sustaining and ending the gameplay.
* **Packages**
  + DoaEngine is the game engine that we have used to configure our game system, it is written by our member Doğa Oruç. It enables us to create Camera, GameBoard, Player, UIManager and Game Master Objects to configure the game. Also, it optimizes the features of our system. Briefly, it is fundamental structure of our game.
* **Entity Classes**
* **Continent**: This class creates the continents and decides on which province is belonging to which continents and defines the features of continents.
* **Province**: This class creates the provinces that are the fundamental part of the game and defines the features of those provinces.
* **Player**: The person who is playing the game is defined by this. This one specifies the capabilities of person, such as; sequence, colour, the number of soldiers, the type of playing (defensive/offensive).
* **AIPerson**: This class extends players and converts them to AI players. Also, chooses the aggression, playing type, or belongings of AI person.
* **Dice**: This class defines the result which is calculated by the dice and game master uses this class during the gameplay to result the game master’s action.
* **Camera**: This class is used to define the zoom in and zoom out positions. Briefly, it is used to arrange the perspective of game players.
* **Gameboard**: This class is used to define the playground by specifying the dimensions of X and Y axis. Thus, we can upload new boards with this class.
* **Connections Classes**
  + **SocketServer**: This class is used for the connection of the system. It works as a server and uses the UPnP to provide port connection. After the multiplayer game is called it starts waiting do the connections and it creates new connections for the desired person capacity**.** After connections are established, it sustains the connections among the users and manages the messaging and data transfer among the players.
  + **ClientServer**: This class is used to provide a connection with the created servers into the system. It takes the server IP and attends to the waited connection between the server and that client.
* **View Classes**
* **UIManager**: This class is created with the help of DoaEngine (Game Engine of that game), it is used to manage and arrange the other view classes and use them to show desired information to the user.
* **SinglePlayer**: This view class is controlled by UIManager and it creates the Single Player Screen. We can choose our single player game option with the help of that view.
* **MultiPlayer**: This view class is controlled by UIManager and it creates the Multiplayer Screen. We can choose our multiplayer option with the help of that view and it uses the SocketServer or the Client connection protocol via Game Master.
* **Rules Screen**: This view class specifies the rules of the game and displays desired information and rules to the player.
* **Main Screen**: This class is the welcome screen of our game and it is managed by the UIManager. It is the starting view of our view flow.
* **Settings Screen**: This class is used to show Setting of the game and it has slider, checkbox, and buttons to arrange the options and the features of the game.

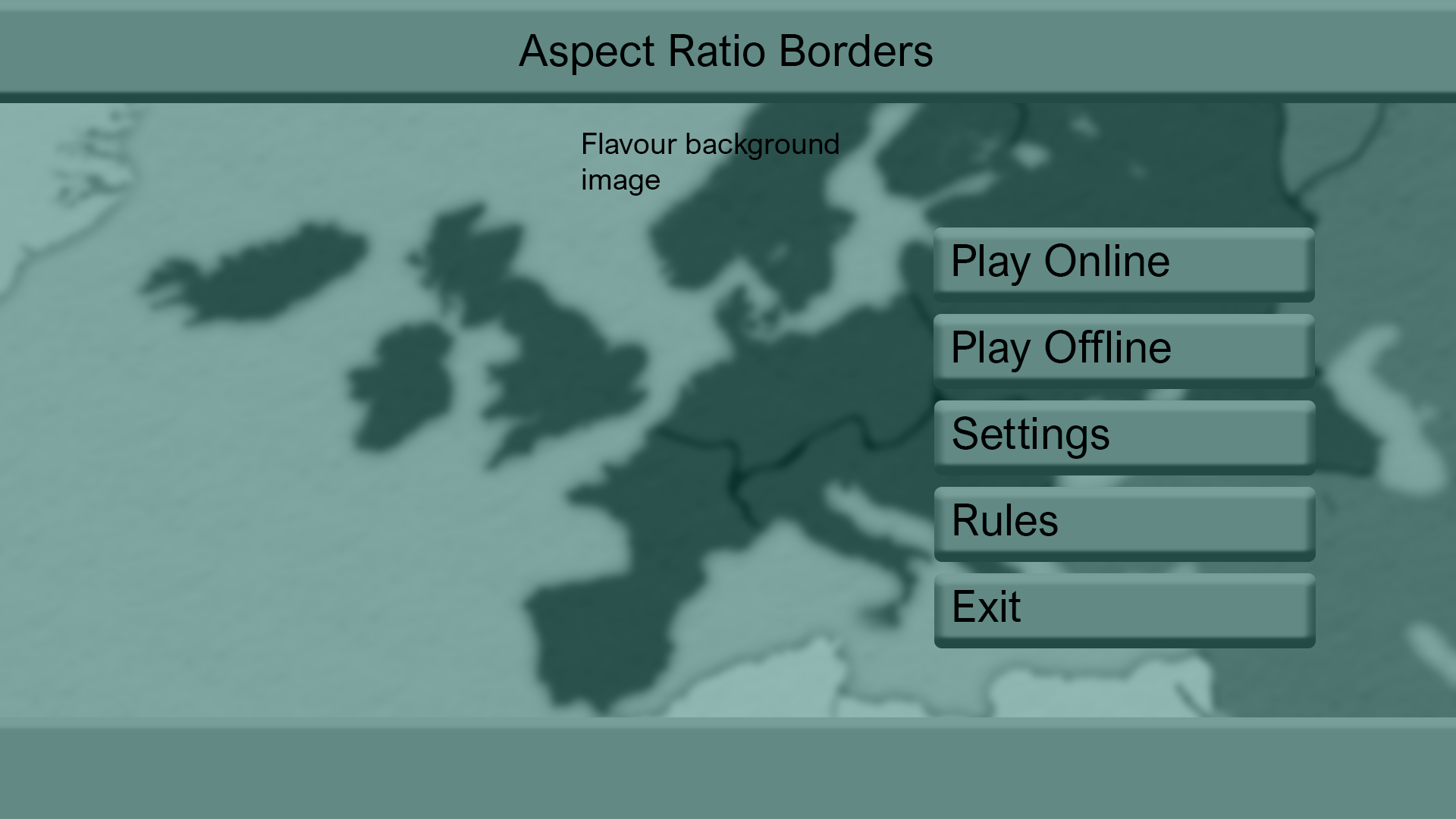
### 5.1.4 Activity Diagram



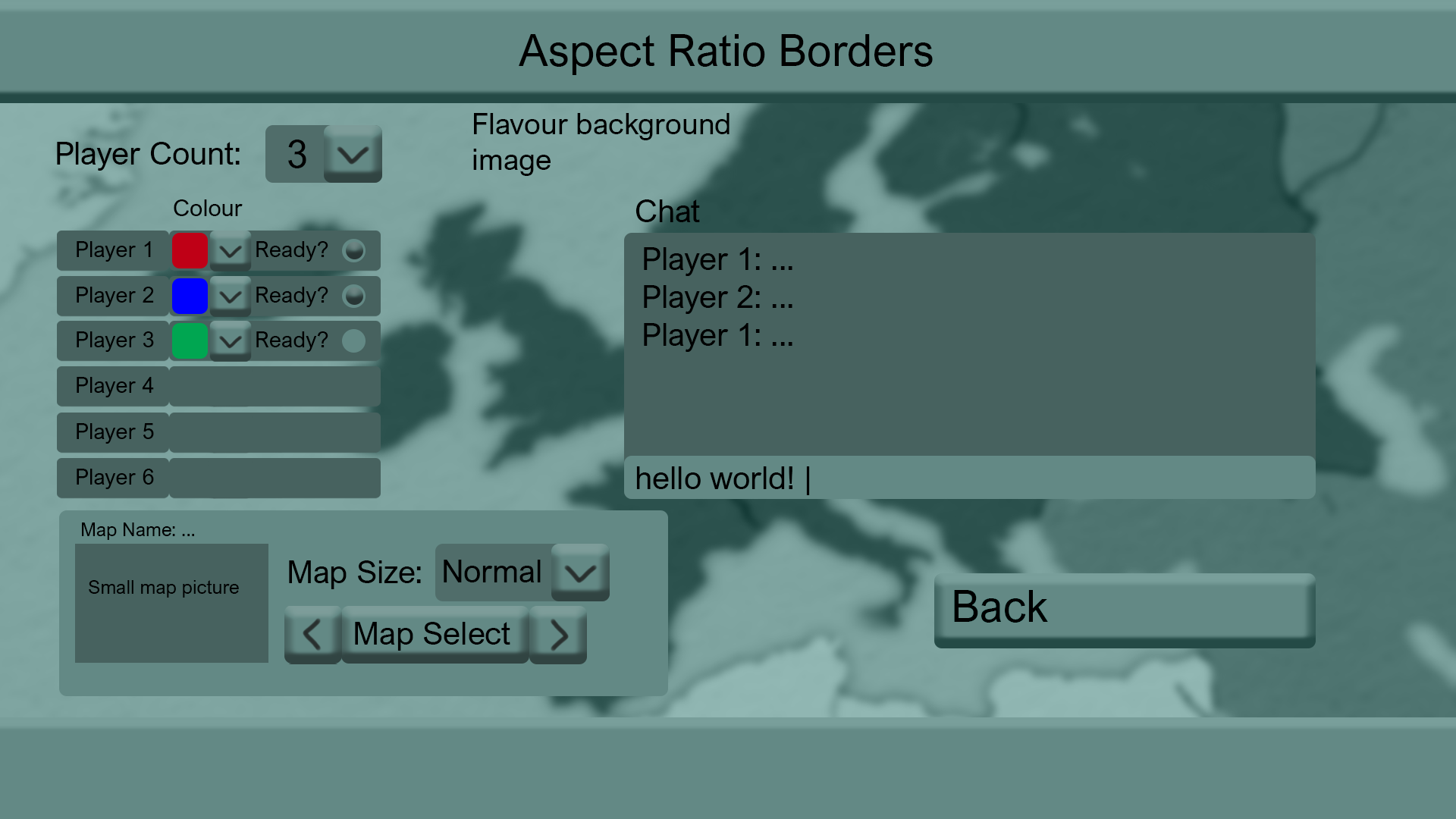
This diagram shows how our activities work. Firstly, we have main menu decisions and with the help of this main menu decisions option you can choose multiplayer game, single player game, options or you can quit the game. After game starts, it waits for game to finish. After that it shows the result and turns back to the main menu.

## 5.2 Screen Mock-ups

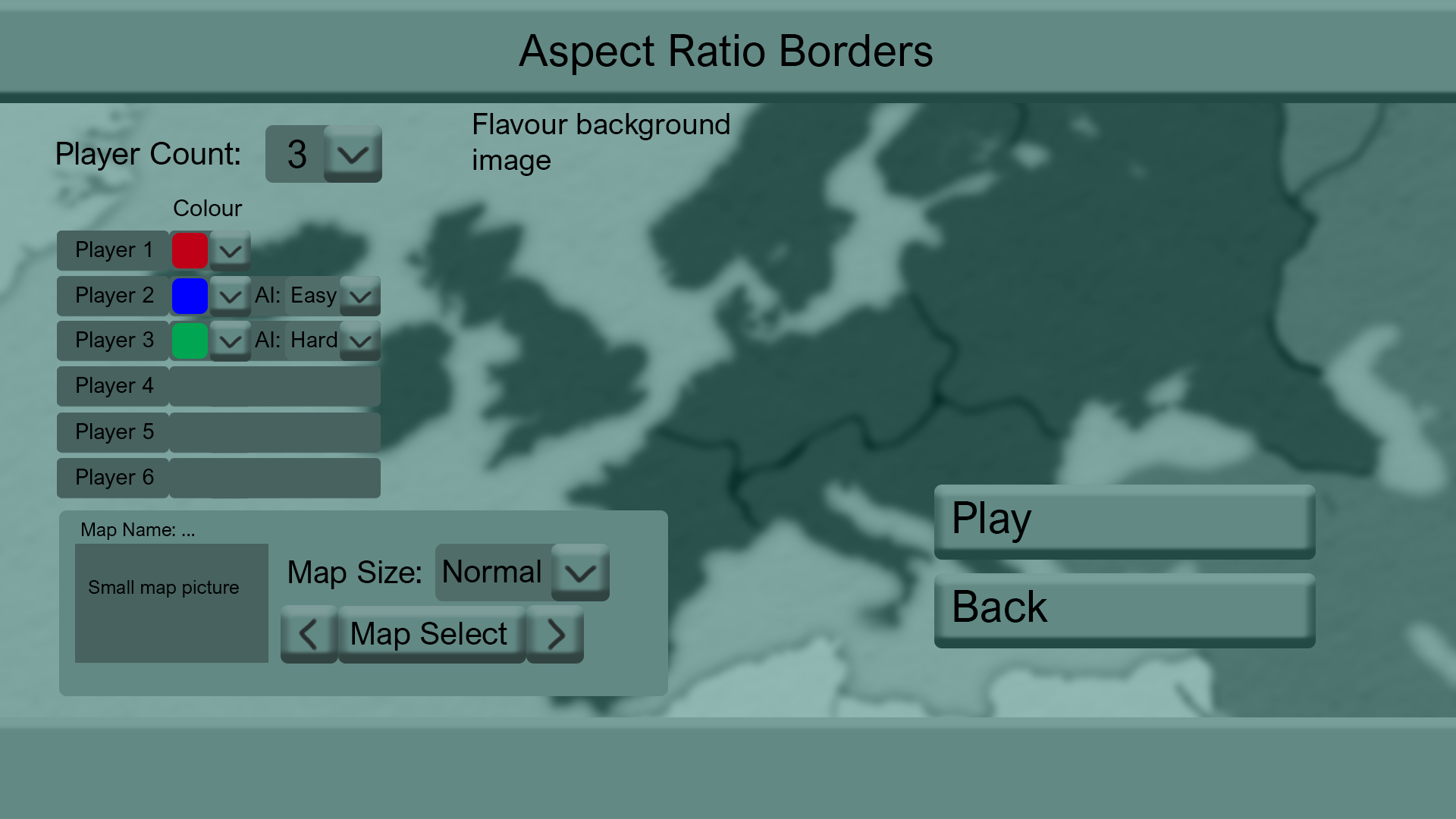
### 5.2.1 Main Menu



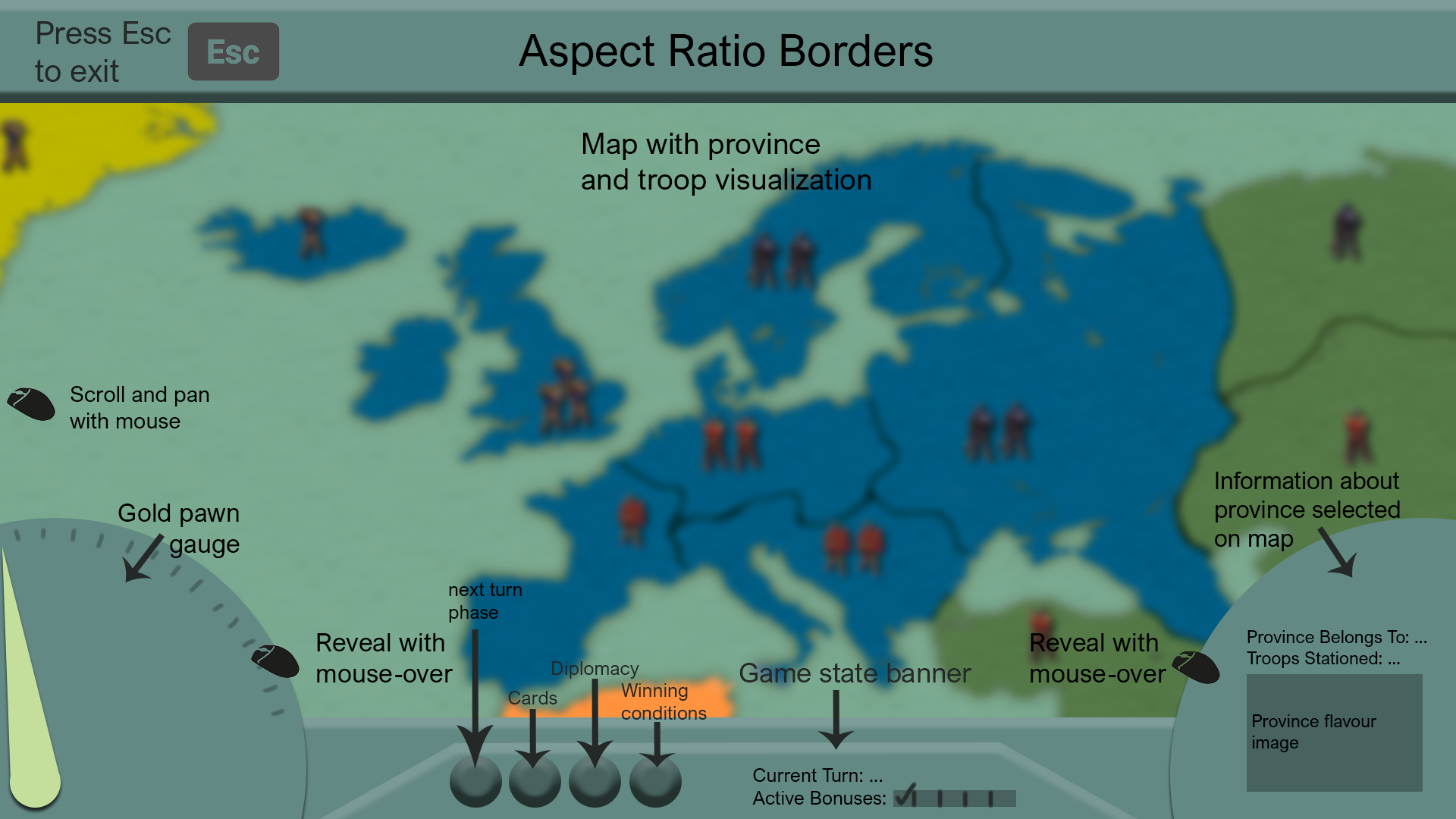
### 5.2.2 Play Online



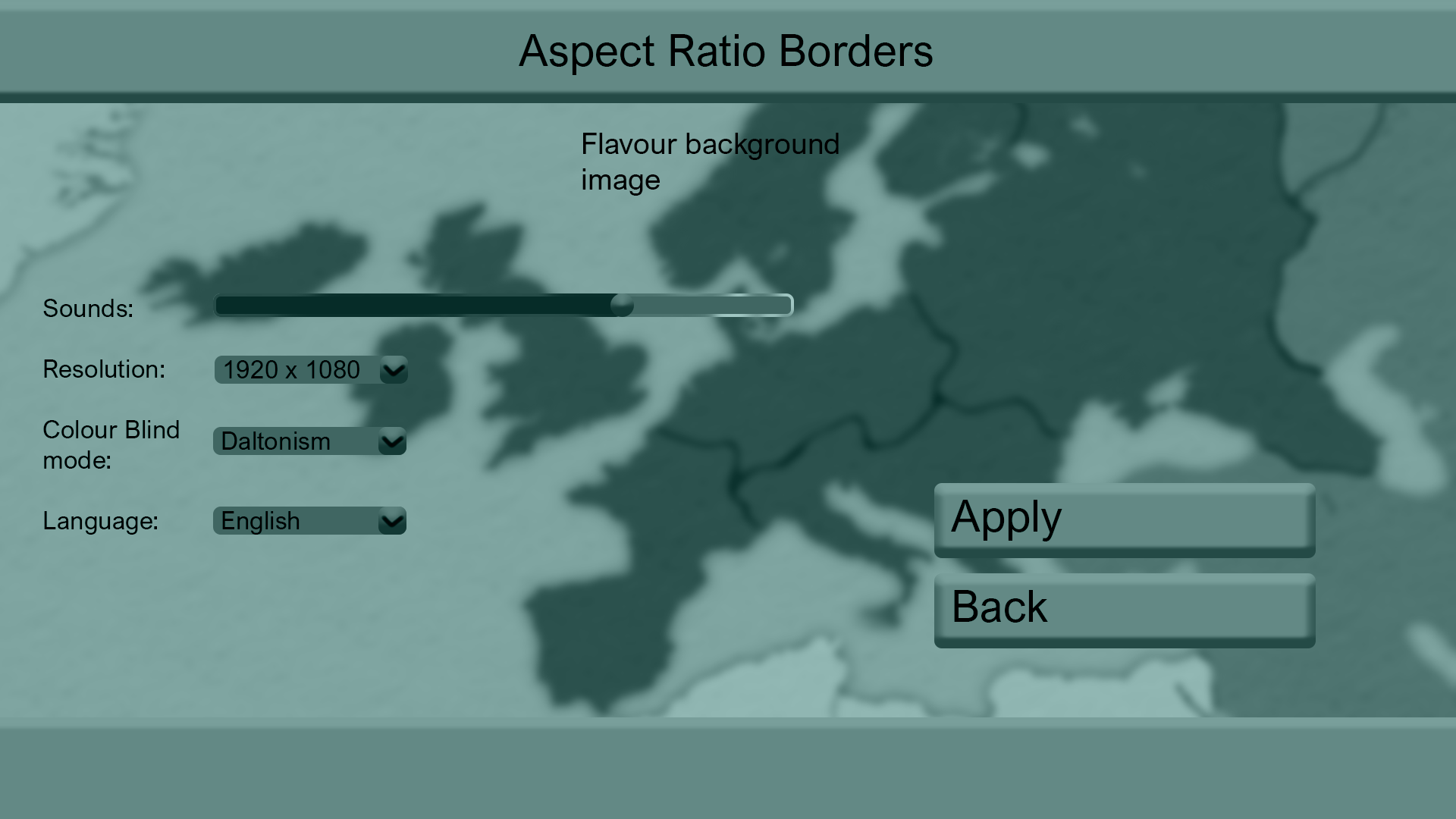
### 5.2.3 Play Offline



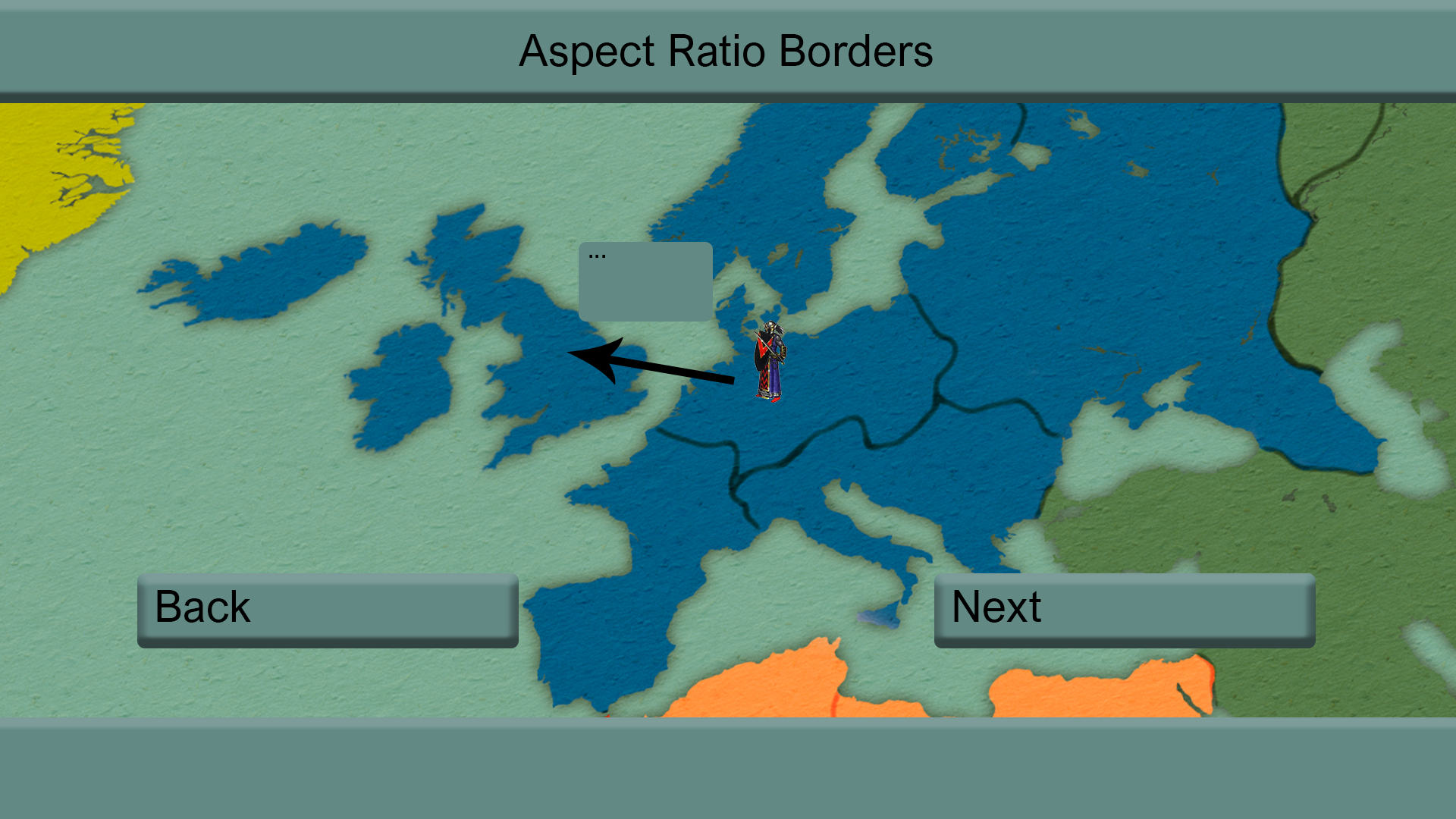
### 5.2.4 Game Screen



### 5.2.5 Settings



### 5.2.6 Rules



# 6.0 REFERENCES

|  |  |
| --- | --- |
| [1] | D. Gordon, «TotalDiplomacy,» 4 October 2013. [Çevrimiçi]. Available: https://www.cardboardrepublic.com/classics/risk-vs-diplomacy. |
| [2] | N. Bevan, «Measuring usability as quality of use,» *Software Quality Journal,* cilt 4, no. 2, pp. 115-130, 1995. |