CS 319 - Object-Oriented Software Engineering Analysis Report

Infinite Tale

Group 1-H

Mehmet Erim Erdal

Halil İbrahim Azak

Hüseyin Taşkesen

Can Özgürel

Mert Özerdem

Contents

1 - Introduction

Infinite Tale is representing a cult in strategy games. Examples similar to Infinite Tale can be seen in some gaming websites. The main purpose of this strategy game is to conquer as much land as possible and spread your fame and glory across all world with using rational thinking and carefully designed tactics. This game's creation idea is based on famous game Travian which is an online game played by millions. Infinite Tale started with the idea of Travian but evolved in different ways, giving it significantly different ways.

Travian:
https://www.travian.com/tr/game/playstyle

To list several features that makes Infinite Tale different from travian:
1- Game will be turn-based, not real-time playable as in Travian.
2- A difference in calculations of war advantages and different bonuses or disadvantages will be calculated such as weather, height etc.
3- Exotic map design(s).
4- Changed economy understanding. New looting rules and more.

Purpose of this game will be similar to Travian however, conquering as much as conqueror can within the given turn limit. Game will be a desktop application and will be controlled with mouse.

2- Overview

Basically, the map of the game is a hexagonal tiling for the sake of simplicity. Each hexagonal in the tiling will represent a land piece. Conqueror will start the game with 1 piece only and with a small army to give the player a head start. Every piece of land will have a number of villagers on it, which may be different depending on the land. Villagers generate tax, which is the main source of gold of conqueror's empire. Gold is the only currency in game which is used create units. There will be different conditions that change the state of war, including weather conditions or height advantage. Some troops will have significant advantage over each other in these specific conditions even if they are outnumbered. Which means that solid tactical analysis is required in order to be the best conquer among them all. Tiles can be selected by mouse clicks and attacks can be regulated choosing amount of army. There will be different time constraints for farther away tiles to conquer. Overall the game will be played with mouse clicks.

2.a - Gameplay

Game is based on turns as time units. A conqueror or computer can move their soldiers 1 tile in 1 turn, which means sending troops further will take linearly more time, also more resources so if conqueror is sending troops far, it should be worth it. Also in 1 turn, conqueror can buy their troops any item from merchant. In one turn it is not possible to create infinitely number of troops but if conqueror has more tiles, they can produce soldiers faster in total. So simply 1 turn is the base unit of time. With mouse clicks conqueror can decide their next tactical operation. A detailed documentation about the game will be given about gameplay with all details included which might help the conquer to grasp the tactics they will want to obtain.

2.b - Enemies

Since the game is single-player, enemies will choose random classes and will also start in random places but the enemies will be computer-generated and they will have their own choice-making system AI. Enemies will obey the same time constraints as the conqueror do, but they will not necessarily be aggressive against the conqueror. Instead, like in real world situations they might conquer each other as well due to conflicts. Hence conqueror might consider himself unlucky if multiple different empires are co-operating to attack him, or he might have gotten too powerful which draws attention.

2.c - Lands

As described, there will be several different landscapes with different weather conditions. Cities will correspond to each tile but several tiles will make up a continent which might together make a desert with extreme weather conditions, or a rocky which will serve as a natural wall, making it hard to attack and conquer. Cold weathers are also a loss of performance both for attackers and for defenders. Attack and defence bonuses will be determined by game's war calculation mechanism.

2.d - Resource

Main resource of the game is gold. Golds can be gathered by taxing villagers. The more villager conqueror has, more gold income he will have. Also conquering other empires will grant conqueror their gold too. So conquering a big and well defended city might earn conqueror a treasure. But since it is impossible to see how much gold a city has within it, sometimes it is the wrong decision to assume that a big city has lots of gold. These cities might've spent their all gold to training troops

and conqueror will get almost nothing by conquering them except the villagers which will start paying him tax.

With gold conqueror feed his army properly, which is a must because without gold troops will start to starve and go back to villager state, thus wasting the gold conqueror used to train these troops. Gold is also used to train units which conqueror will need in order to conquer other lands, which will be the main usage of conqueror's gold source.

2.e - Strategies

Conqueror should make his decisions wisely in order to be effective in these lands. There are many strategies conqueror can obtain in order to achieve his goal. Passive approach against enemies in not recommended since eventually a computer will have majority of the lands and the source of gold, hence troops. Then they will be incredibly hard to defeat. Hence, conqueror should effectively invade close areas to gain a momentum. On the other hand being way too aggressive and attacking everywhere conqueror can result in loss of many troops and more than one defeat may mean end of the story since you will not have much source of gold, neither troops if you lose multiple wars.

It is best to gather small lands first to increase your gold income, most likely neutral lands will work because they will have less defence to defeat. Having villagers work in neutral lands in the start will give the conqueror a solid and steady gold income. It is recommended to expand as much as one can without taking too much risk. Also one can give a check about other lands of troops with sending a small army, gathering information about other army. Also conqueror should consider height and weather advantages in order to gain an edge on his enemies, counter his opponents in every possible way.

3 - Requirements Specification

3.1 - Functional Requirements

3.1.a - Start Game

Starting game will present conqueror the beforehand designed map with different continents and tiles as cities. Conqueror will have control over a single tile in a continent. Hence this might be advantageous for him or may be a slight disadvantage, for example this situation is advantageous if conqueror starts from a

very high ground. Conqueror can easily see his own tile among others because it will be in a specific color, the color of conqueror's flag.

Conqueror can also see enemy territories which each empire will also have their own specific colors painted in those tiles. Additionally, there will be neutral lands with probably a very small number of troops on it or maybe none and villagers. Neutral lands will not have a specific color.

On top of the map, there will be a rectangle block where you can see the necessary informations about your empire. Your total number of villagers and troops can be seen there, also your gold amount and population capacity will all be written. As each turn progresses this information table will be updated with their new values.

Mouse clicking a tile will open a rectangular information card on right side of the screen, which will give information about that specific tile only. If the tile is conqueror's, then one can see how many soldiers and villagers conqueror has but if tile belongs to an enemy, this information is not visible because of fog of war. One will not know how many soldiers enemy has before attacking it unless conqueror owns a tile closest to enemy tile. Tiles that one does not have a border will not give information except that whom that tile belongs.

3.1.b - Pause / Settings

Change settings option will pause the game when it is clicked and will display a small menu in the middle of screen. That screen will enable the user to stop or start the music played in the background. Also save the game option will be chosen from here, enabling the user to save the game. This will help player to load the game by choosing any saved data before, continuing their adventure.

3.1.c - Load Game

Load game option in the main menu screen will help the player to choose any saved file before and continue the game from the state that he has saved.

3.1.d - Help

Help will have several instructions for the player to grasp a concept. It will explain the basics of playing game, also some detailed instructions such as how military, economy, time management and will explain how to analyze in-game situations to make them to your advantage such as weather conditions and how they affect your troops.

3.1.e - Credits

Credits will display people who had spent effort in developing this game and will give necessary contact information so you can notify them about changes you want to make or additions that will make this game better. We are looking forward for feedback.

3.2 - Non-Functional Requirements

3.2.a - Performance

Performance of the game will be high, since our planned methods never use an algorithm which goes exponential. Even in war calculating there will be O(N) time constraint where N denotes the total number of soldiers. With good choices and aesthetic design of algorithms, we expect our game to run fast.

3.2.b - Graphics

We will be designing graphics with JavaFX instead of Java because we care about how graphics look in our game. Normal Java GUI does not have variety of good design options and also it is capped in 8 bits. In order to get smooth running graphics we decided to use JavaFX which is 32 bits and has more variety which leads to a better visualization.

3.2.c - Readability of Documentation

While writing in-game help and documentation about the game, we try to keep it simple and clear so that player can understand how to play and other several instructions without having to know great English.

3.2.d - Extendibility

What makes this game unique is that has extendible parts unlike other strategy games. Player will be able to design his own map, store it in a txt file and upload it to have a custom-created map himself. (Will be in developing phase in first iteration.)

4 - System Model

4.1 - Use Case Model

4.1.1 – MainMenu Use Case Model

MainMenu use case of Infinite Tale game is explained with detailed here.

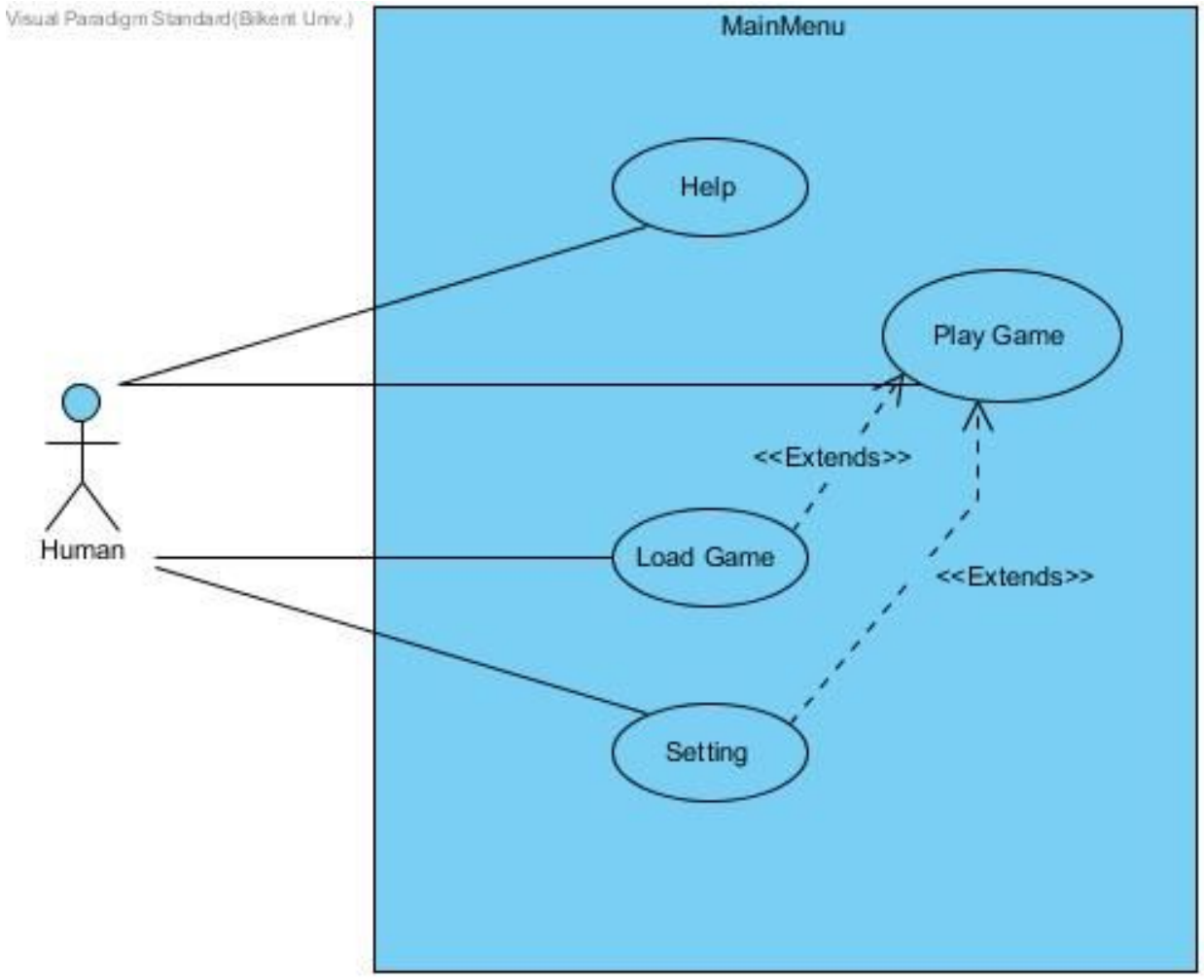


Figure 4.1.1 - Use Case Diagram

4.1.1.a - MainMenu Use Case Definitions

Use Case Name: Help

Including Actors: Player

Pre-conditions: system must keep help instructions.

Entry Conditions: Player Clicks on “Help” button via using main menu interface.

Exit Conditions: Player clicks on back icon.

Flow of Events:

1. Player clicks on help button.
2. System displays pre-recorded help instructions.
3. Player clicks on back icon, then help interface closes.

Use Case Name: Play Game

Including Actors: Player

Pre-Conditions: Music volume is set to default unless it configured via using setting option.

Entry condition: Player clicks on “Play Game” button via using main menu interface.

Exit condition:

- Player can select exit via in game menu.
- Player has to conquer all tiles and must beat all boots.
- Losing all owned tiles ends the game and force player to return main menu.

Flow of Events:

1. Game starts by clicking “Play Game” button.
2. System generates the level by placing all actors to their default state.
3. Player moves and attacks its opponents.
4. Player conquers all enemy owned tiles.
5. Game finishes and returns to main menu.

Alternate Flow of Events:

1. Game starts by clicking “Play Game” button.
2. System generates the level by placing all actors to their default state.
3. Player exits game via in-game exit option.

Use Case Name: Load Game

Including Actors: Player

Pre-condition: There must be a Pre-saved game in system, system saves relevant information

Entry condition: Player selects “Load Game” button via using Main Menu interface

Exit condition:

1. Player selects any saved games and system recreates game from where it is left.
2. Player exits load Game interface by using back icon.

Flow of Events:

1. Player selects “Load Game”.
2. System regenerate the game according to saved game’s information.

Alternate Flow of Events:

1. Player selects “Load Game”
2. System cannot find any saved games
3. System returns to main menu.

Use Case Name: Setting

Including Actors: Player

Precondition: System is holding music files and set them default.

Entry condition: Player clicks on “Setting Button”.

Exit condition: Player clicks on back icon.

Flow of events:

1. Player clicks on “Setting” button.
2. Player changes music setting to off.
3. Exits by clicking back icon.

4.1.2 – InGameTopBar use case model

MainMenu use case of Infinite Tale game is explained with detailed here.

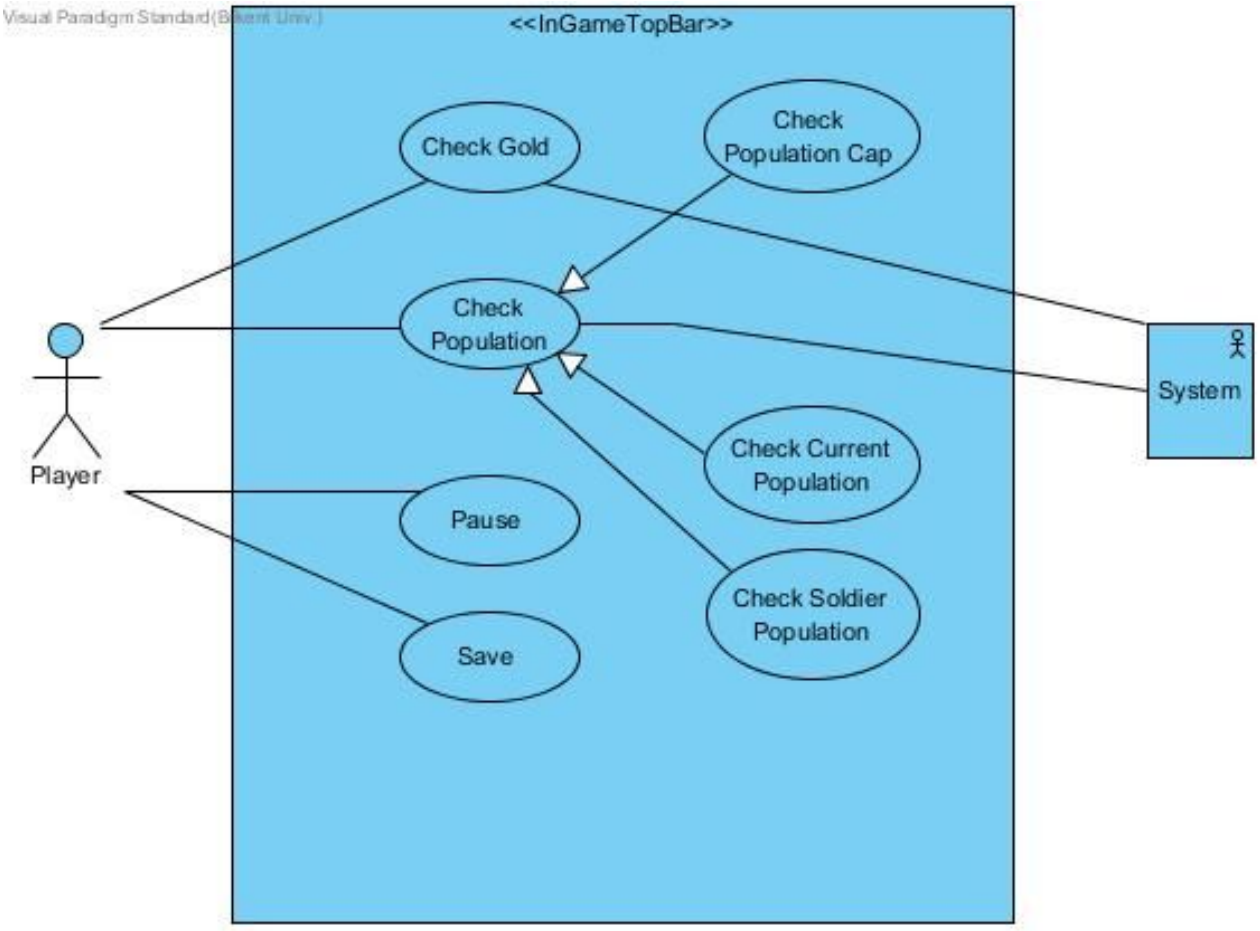


Figure 4.1.2 - Use Case Diagram

4.1.2.a – InGameTopBar Use Case Definitions

Use Case Name: Check Gold

Including Actors: Player, System

Precondition: System contains gold variables of each player.

Entry condition: Player clicks on “Gold” button via using InGameTopBar interface.

Exit condition: Player stops pressing on “Gold” Button.

Flow of Events:

1. Player clicks and holds on “Gold” button.
2. Player releases the button.
3. Pop-up gold interface closes.

Use Case Name: Check Population

Including Actors: Player, System

Pre-conditions: System contains population variables of each player.

Entry condition: Player clicks on “Population” button via using InGameTopBar interface.

Exit condition: Player stops pressing on “Population” Button

Flow of Events:

1. Player clicks and holds on “Population” button.
2. Selects check current population.
3. Player releases the button.
4. Pop-up gold interface closes.

Alternate of Events:

1. Player clicks and holds on “Population” button.
2. Selects check soldier population.
3. Player releases the button.
4. Pop-up gold interface closes.

Use Case Name: Pause

Including actors: Player

Pre-condition: Game has to be running.

Entry condition: Player clicks on “Pause” Button via using InGameTopBar interface.

Exit condition: Player clicks on “Pause” Button via using InGameTopBar interface.

Flow of events:

1. Player pushes “Pause” button.
2. Game stops receiving input until it gets unpaused.

Use Case Name: Save

Including Actors: Player

Precondition: System has to have space to save game variables, game must be running.

Entry condition: Player clicks on “Save” Button via using InGameTopBar interface.

Exit condition: After save game returns to normal state.

Flow of events:

1. Player clicks on “Save” button.
2. System saves game variables.
3. Game returns to normal state.

4.2 - Dynamic Models

Dynamic models provide detailed information and some must-faced situations that the player will experience in Infinite Tale with the help of sequence diagrams. There are many important objects that will interact throughout the game such as tiles and troops. Since it is impossible to analyze all these situations, we will provide a couple sequence diagrams for easier understandability.

4.2.a - Sequence Diagrams

4.2.a.1 - Start Play

Following sequence diagram explains the scenario below:

Scenario: Conqueror Kaan requests from the system to start playing by mouse clicking the required button which starts the game. After this request, system first goes to fullscreen mode, meanwhile gets the graphics objects to create visuals of the game. System loads its objects by calling object creator methods, since the map will be pre designed system will be reading a properly formatted txt file. Placing these objects to their locations, system starts the game by making conqueror able to give specific inputs with his mouse clicks.

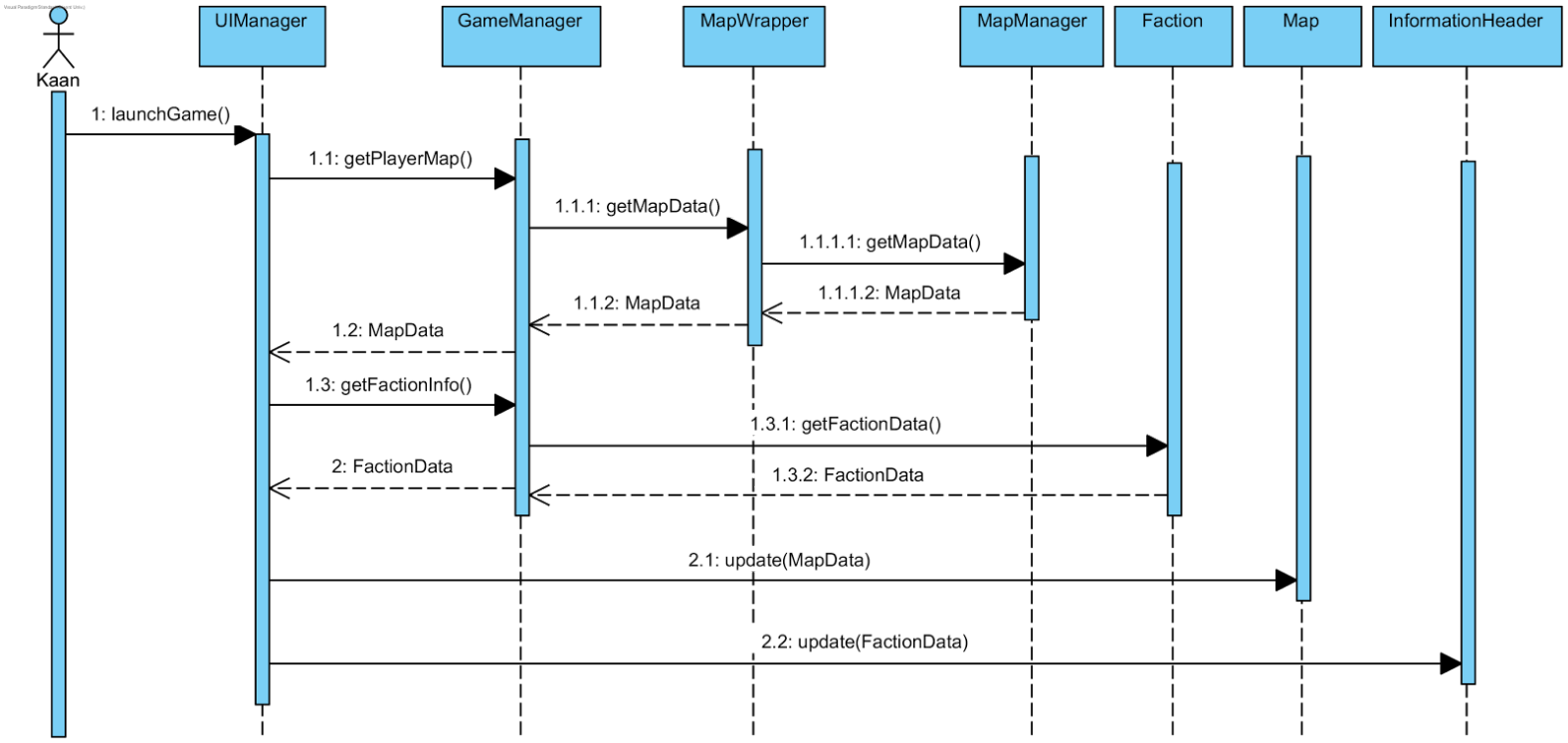


Figure 4.2.a.1 - Sequence Diagram

4.2.a.2 - Successful Conquer

Following sequence diagram explains the scenario below:

Scenario: After conqueror Kaan successfully started the game, he plays a while, trains some troops and he sees an opportunity to conquer a tile which he sees as an advantage. Kaan clicks on that tile and sends his chosen troops to that tile. After troops reach their invading place, Kaan conquers that tile with an incredible victory. He sees that color of the tile has changed and the tile belongs to him now. Also he can check how many casualties he had during the war from the Battle Info Window opened after war.

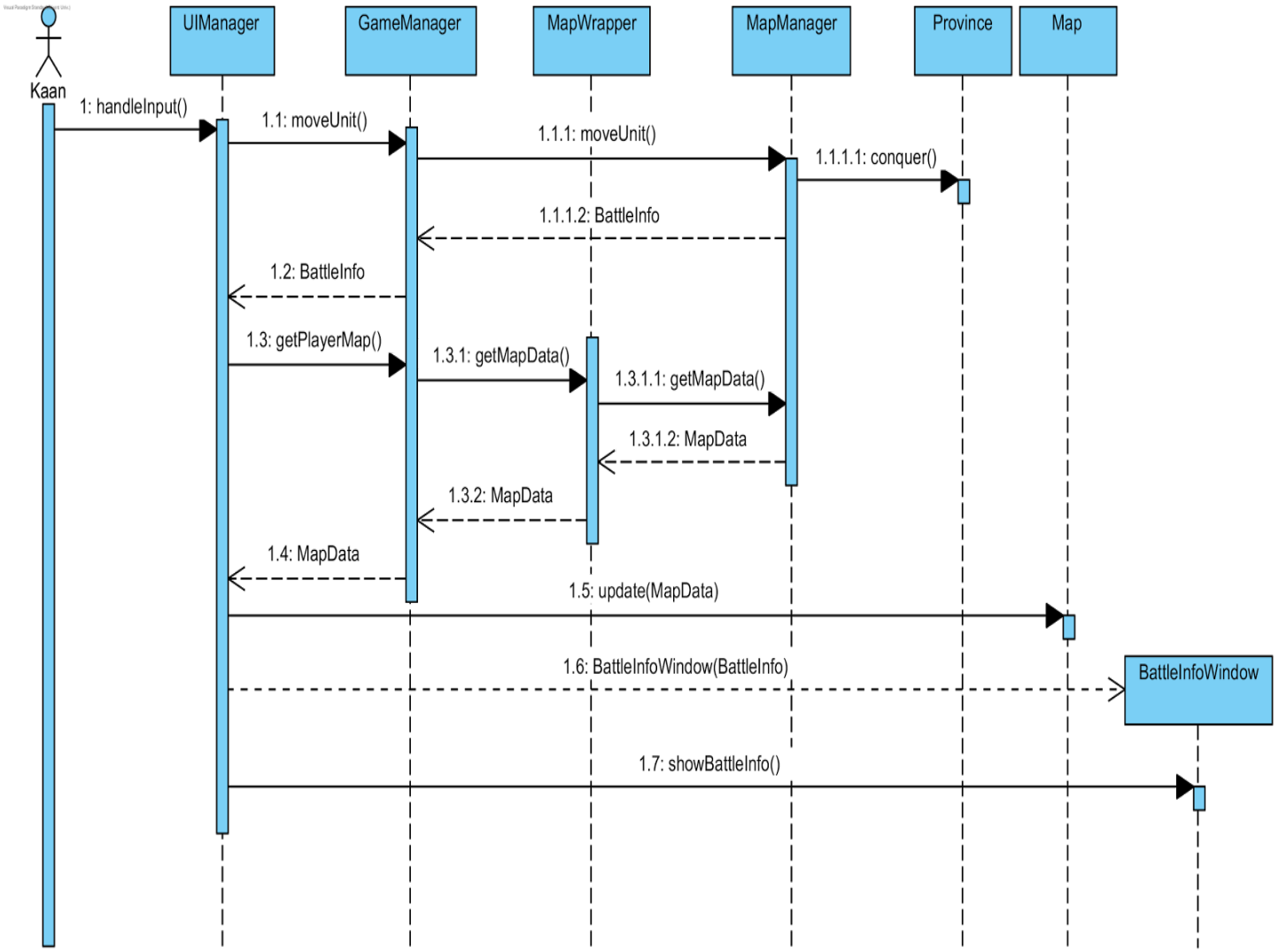


Figure 4.2.a.2 - Sequence Diagram

4.2.a.3 - Saving a Session

Following sequence diagram explains the scenario below:

Scenario: After playing a while, Kaan decides to save his session to play later. He opens the Pause Menu and selects “Save Game”. Then he saves the current session in the opening Save Game Menu.

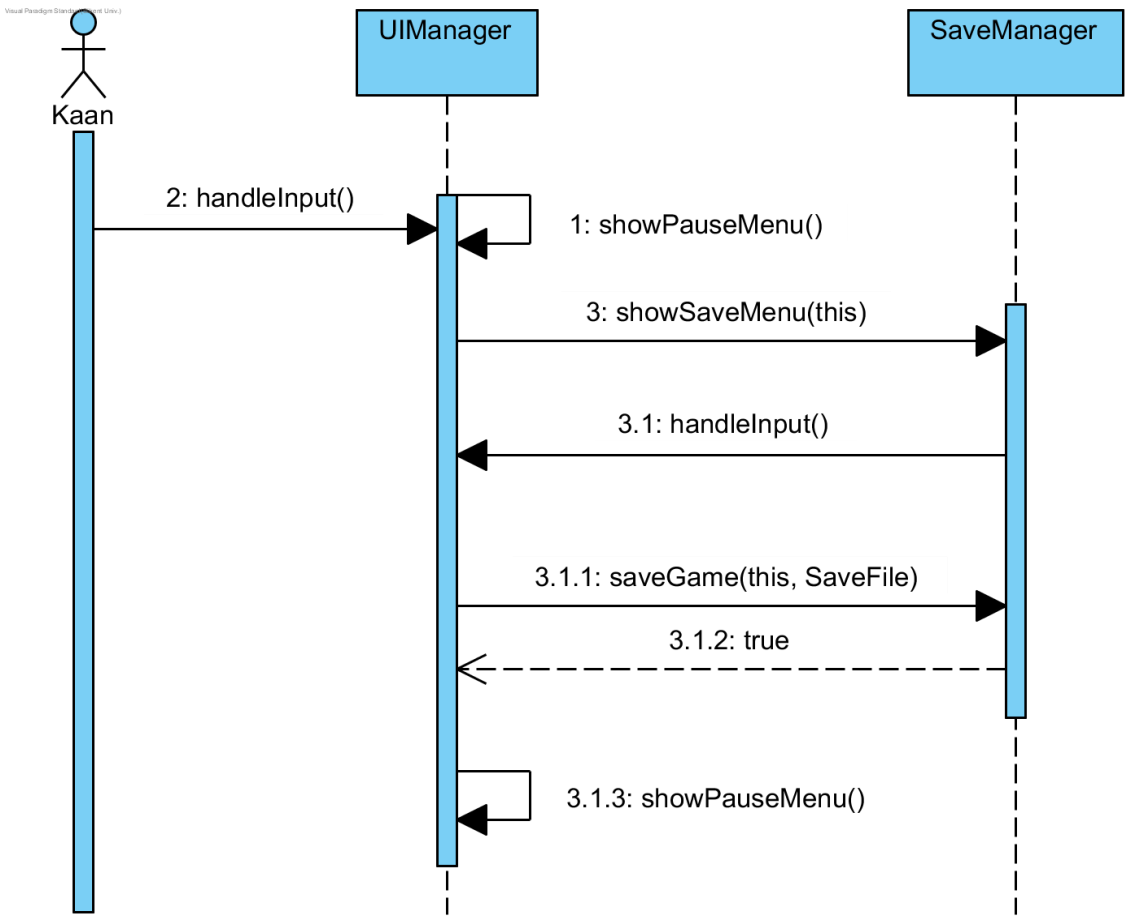
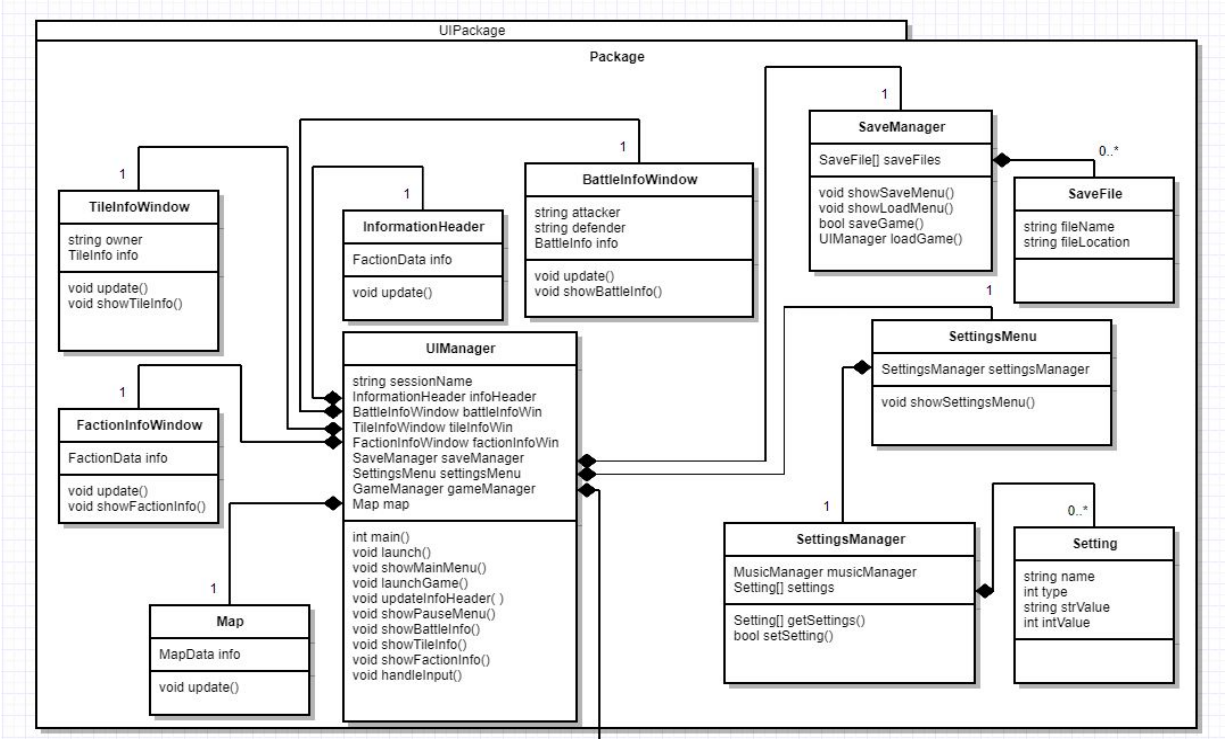


Figure 4.2.a.3 - Sequence Diagram

4.3 - Object and Class Model



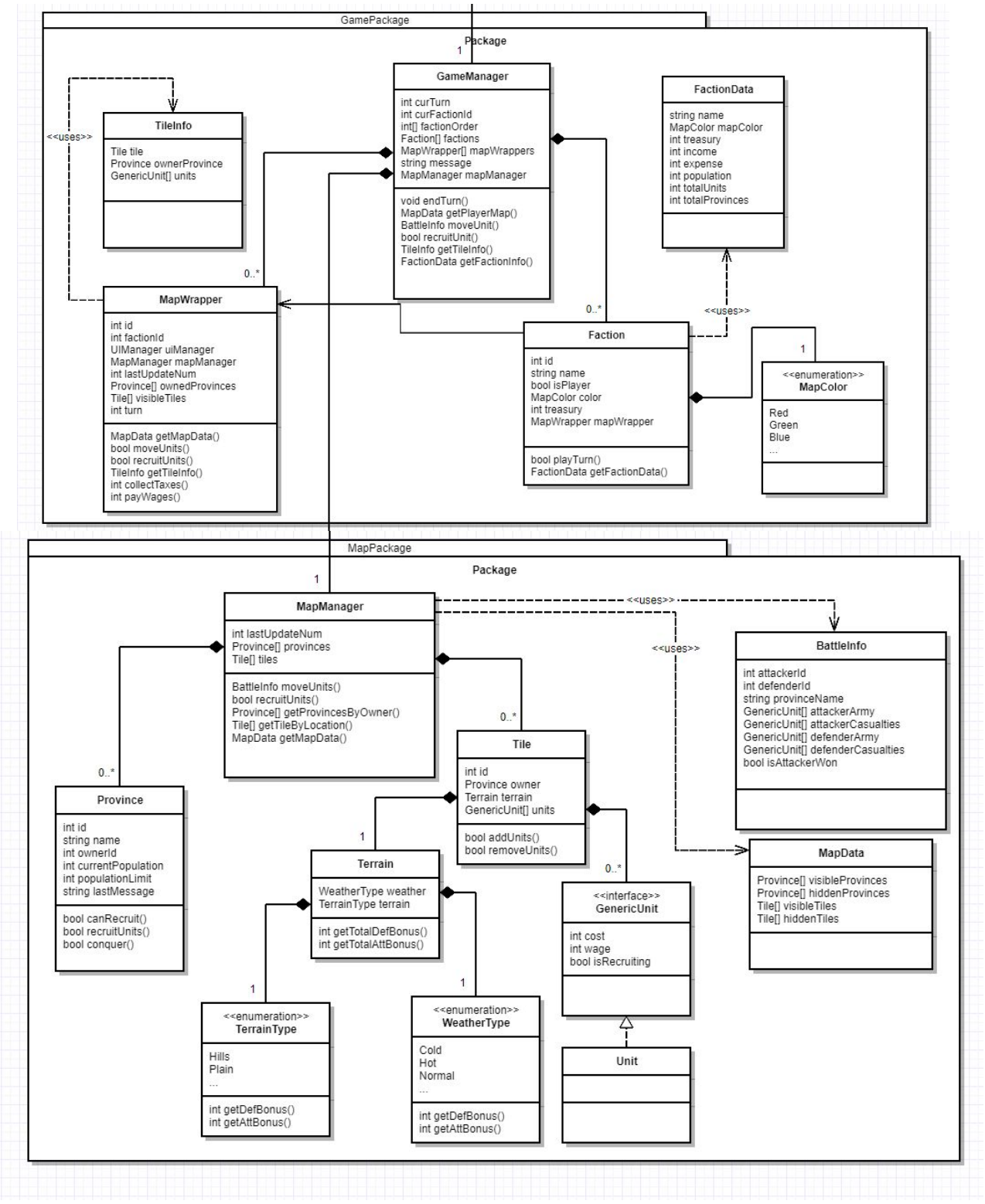


Figure 4.3.1 - Class Diagram

The class diagram of the game is as shown in **Figure 4.3.1**.

The main method of the game is located in UIManager class. When the program is started, UIManager shows Main Menu to the player. If the player choses starting to a new game or loading a saved game, UIManager class will create other classes such as Map, InformationHeader UI classes and GameManager class.

UIPackage: Consists of classes related to game interface and file management. UIManager is also a part of this package. When a game starts, UIManager creates all the classes in UIPackage and updates them accordingly through the game.

UIManager: Creates and holds UIPackage classes and is responsible for the connection between them and other classes of game. UIManager also handles input and calls relevant methods of other classes.

SaveManager: Class is responsible for handling saving and loading. It also shows save or load menu when needed.

SettingsMenu: Handles settings for users.

Map: Class controls the part of the UI where game map is shown. Map class is once created at the beginning of a game and then updated by UIManager when needed.

InformationHeader: Controls the information bar at the top of game UI where the player's resources are shown. Like Map class, InformationHeader is created at the beginning and updated by UIManager when needed.

FactionInfoWindow, **TileInfoWindow**, **BattleInfoWindow** classes are responsible for showing related information by opening a window. These windows do not stay open always like Map class but are hidden after information has shown.

After creating UIPackage classes, UIManager also creates GameManager class of GamePackage. GameManager class is the control class of GamePackage.

GameManager: Class controls the game. At the start of a game, GameManager creates required classes. And through the game it controls the connection between both GamePackage classes and also between them and UIManager.

Faction: Class represents a faction in game and plays as AI. Player faction is also represented using a Faction class but GameManager class does not call AI methods of it.

MapWrapper: Class is a wrapper class for the game map and used by Faction classes. MapWrapper class control the access of different factions to the game map. At the beginning of a game, GameManager creates a MapWrapper class for each Faction class and assigns them. MapWrapper class holds if of related Faction class and limits access to the game map accordingly.

GameManager also handles the connection between MapPackage classes and other classes of the game by creating and holding a reference to the MapManager class. MapManager class in the controlling class of MapPackage.

MapManager: Class controls map state of the game. It creates required information classes and controls them through the game.

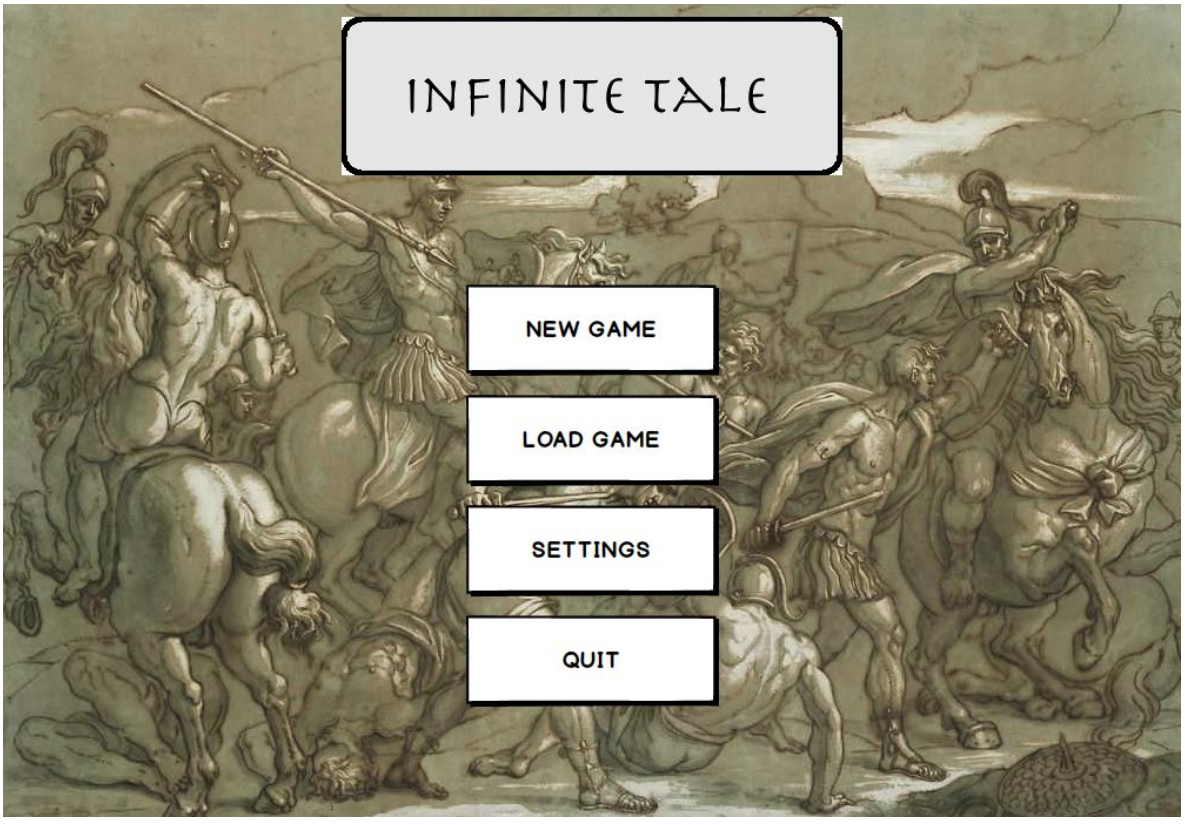
Province: Class represents provinces in the game.

Tile: Class represents a single map tile in the game. A tile belongs to a province and Tile class holds a reference to the owner Province class.

5 - User Interface

5.1 - Main Menu

Figure 5.1.1 - Main Menu



The game will start with the Main Menu. The user will have the freedom to choose from 4 options that are

- New Game
- Load Game
- Settings
- Quit

These options will be preceded by the logo of the game (design subject to change) which includes the title “Infinite Tale”.

Each option will be controlled by their corresponding buttons. The “Quit” button will allow the user to close the game by terminating the program.

5.2 - New Game

Figure 5.2.1 - New Game

If the user chooses the option “New Game” on the Main Menu, they will be taken to the New Game screen. Here, the user will be prompted to enter their name

and start playing the game by providing a text input for the name and clicking on the “Play” button.

If the user wishes so, the “Cancel” button can be pressed at any time to take the user back to the Main Menu.

5.3 - Load Game

Figure 5.3.1 - Load Game

If the user chooses the option “Load Game” on the Main Menu, they will be taken to the Load Game screen. Here, the user can load a saved game by choosing the game to be loaded from the saved games list and clicking on the “Load” button.

Another option is to delete a saved game. The user can select a saved game as if they are loading the game but click on the “Delete” button instead to delete the chosen saved game. The purpose of this option is to save memory space by getting rid of unnecessary files.

Lastly, the user can click on the “Cancel” button to cancel the loading/deleting operation and go back to the Main Menu.

5.4 - Settings

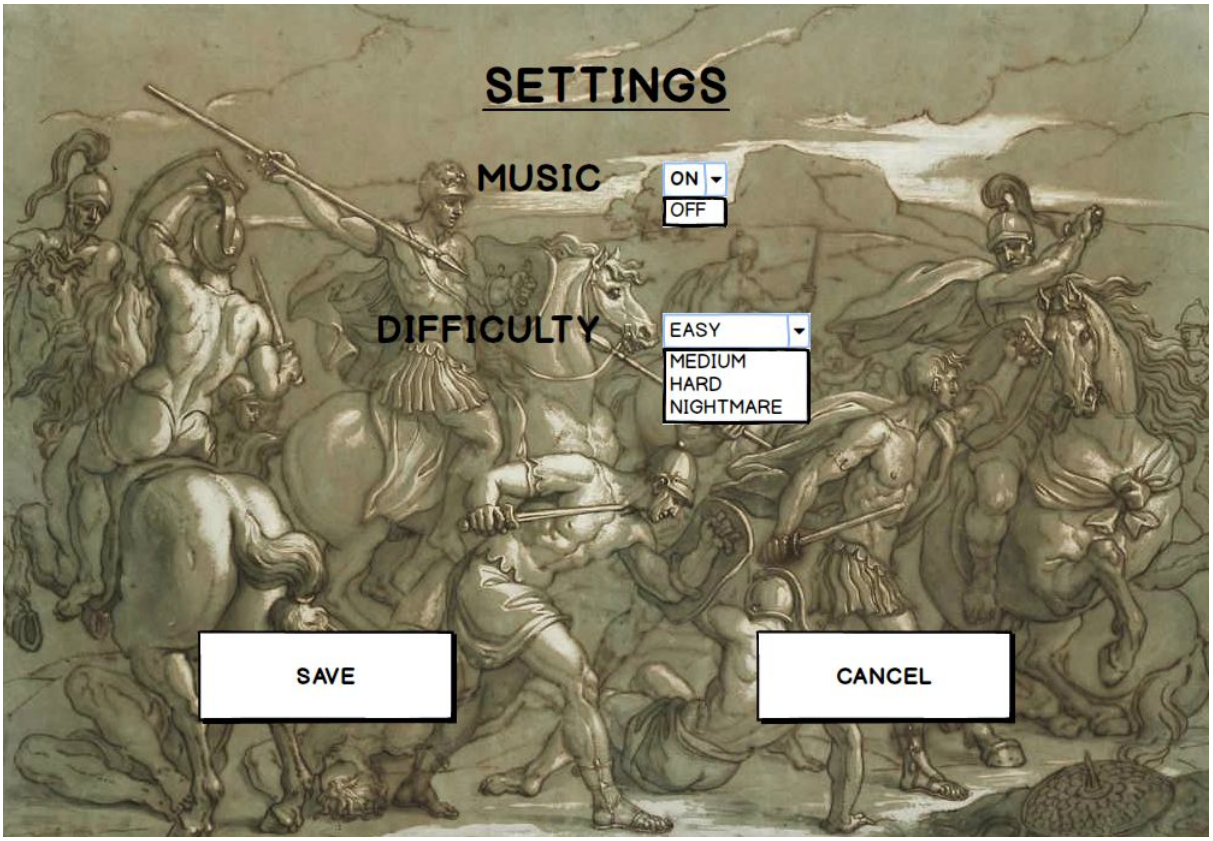


Figure 5.4.1 - Settings

If the user chooses the option “Settings” on the Main Menu, they will be taken to the Settings screen. Here, the user will be able to change 2 (subject to change) options about the game before starting to play.

The user can decide whether or not to listen to music whilst playing the game by choosing from the drop-down menu options. The options “On” and “Off” each correspond to the implied actions for the game music.

The user can also change the difficulty setting of the game to play on their suited level of hardship. How challenging the game can get is under the control of the user this way.

The user can click on the “Save” button to save all of the changes made to the settings of the game or click on the “Cancel” button to revert the settings back to the way they were before the user’s modifications.

5.5 - Gameplay

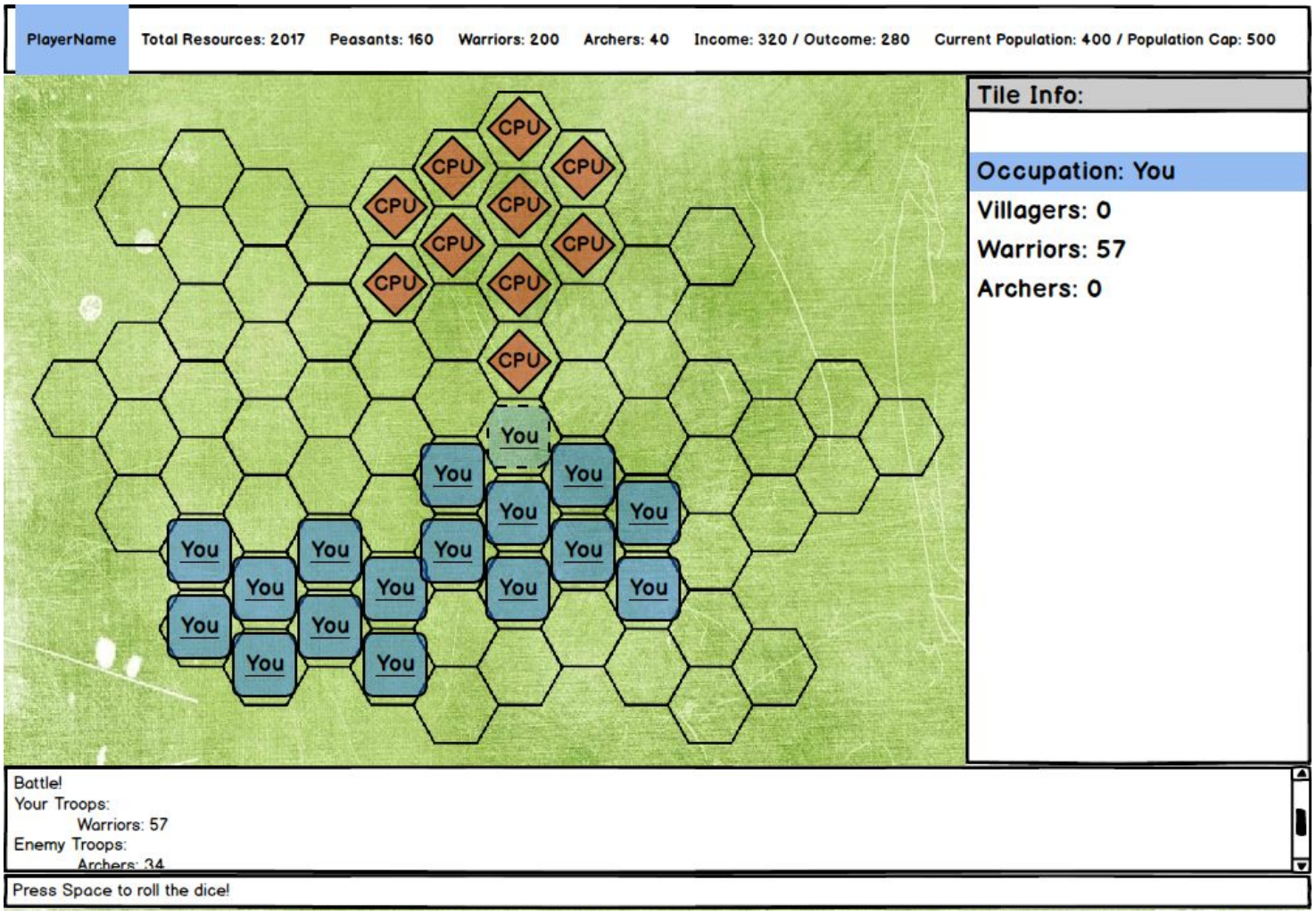


Figure 5.5.1 - Gameplay

After typing the player name on the New Game screen (Fig. 5.2) and clicking on the “Play” button, the user will be taken to the start of the game where they will be matched against a CPU (Computer Processed Unit). Both sides will have a turn to take an action which can consist of moving units, attacking enemy units, converting unit types into each other for a certain cost and so on. The gameplay screen will include a top menu bar (displays the information of the player), a side tile info bar (displays the information of the tile that the user clicked on) and a console with an input bar (to take an action on the chosen tile).

As 5 strategy game enthusiasts, our team decided to create a map made up of hexagonal shapes to maximize the efficiency of the space that we can use.

6- Conclusion

In this analysis report, we specified our design and implementation methods to create a turn based strategy game, named “Infinite Tale”. Our analysis report analyzes our game in two main parts. First of the main part is requirement specification and second one is system model. The focus of the analysis process is the decisions that we make as a team to create a well-organised game.

Requirements specification is the part where we determine functional and nonfunctional units of the game. In functional units, we represent the parts of the game that must work correctly in order to have a playable game. Start game is the core of this project and it is analyzed deeply in use case and sequence diagrams. Also other options must be available for an operating game. Nonfunctional requirements are mostly about the implementation process of our project such as performance, graphics and extendibility.

Our report includes the basic gameplay elements, rules and features of the game. As second, system architecture has been specified with help of use case models, dynamic model and objects and class model diagrams.

Models used are given below in order:

1 - Use Case Model

2 - Dynamic Models

3 - Class and Object Model

4 - User Interface Mockups

After considering which cases we should be using in which conditions, we have found out the most important cases that are a must to include in this report. Use case reports have two use cases, first of them is display of main menu. Main menu is the heart of our game since the most important option, start game lays here. Also one can change settings or display help from this screen. Second one is ingame top bar where player can see all his resources and construct a strategy from it.

In dynamic model, we included important sequence diagrams. One of them is starting the game. Actually this sequence diagram is complicated because starting the game nearly needs all components of the game in use, such as creating a small

number of troops, creating enemies and drawing the visualization of the game, also dealing resources. Dynamic Models help us to create the heart and core of the game. For this reason we took dynamic models most seriously, in case we might face some problems in implementation it should not be caused by poor modelling.

As the last crucial part we have mock-ups. Mock-ups are indeed crucial because the visualization of the game is one of the most important parts. A game with very bad graphics might be frustrating to play with and when dealing with designing objects one needs to find creative images of these objects to represent within game.

To conclude, we think this analysis report is very detailed in terms of explaining many different cases in the game. Also this report helped us to create a solid underground for the start of our project.

7 - References

1 - Object-Oriented Software Engineering, Using UML, Patterns, and Java, Bernd Bruegge and Allen H. Dutoit, 2010/3rd, Pearson

2 - analysis (Analysis report of Crazy Ball, considering the fact that a brick breaker and a strategy game has nothing in common, this report helped us to create a neat template structure.)