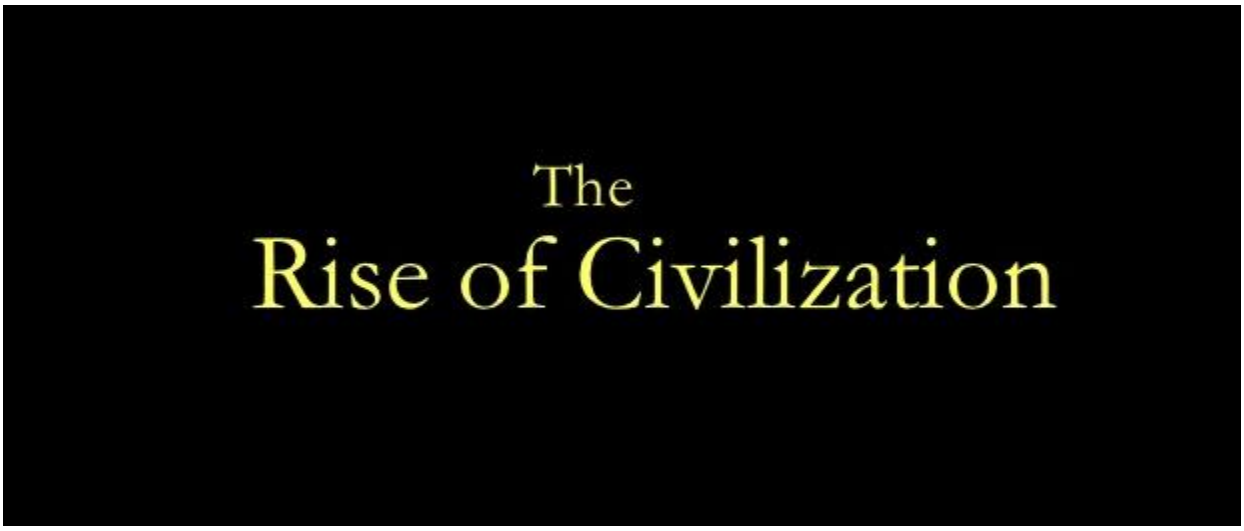
Software Engineering Project Report



Project Development Report for
The Rise of Civilization

Group 13:
Akhil Kateja
Adil Qaisar
Andriy Serafyn
Himanshu Anand

CS 440
University of Illinois at Chicago

09/24/2015

Table of Contents

List of Figures

I Project Description

1 Project Overview

Rise of Civilization is a turn based strategy game. It is an intuitive and easy to play game that is suitable for everyone. As a strategy game, it requires the player to think logically in order to beat his opponents. Once you start the game, you get the opportunity to choose to play either versus an AI opponent or versus players on your local area network.

The objective of this game is to develop and expand the user's empire that can last for ages and resist conquests from other empires. Each player assumes the role of a ruler of an empire and will initially have some resources in the form of manpower, goods, army units and virtual currency to build his/her empire and compete with other empires/civilizations. The player should possess a knack of taking critical decisions regarding exploration, warfare and diplomacy.

The game basically requires players to build a base where they build their buildings and units and eventually defeat all of the other players that are not in their team. That is a really good and fun way for people to develop their logical thinking, because the game requires from you to start thinking a few moves ahead (like a chess game).

As it is a turn based game, each player has a number of things he can do during his turn, which include: create new buildings, spawn units, move units and more. It is important to be noted that in our game, when you create a new soldier for example, that is not only one soldier, but it represents a brigade of soldiers. Movement of units is limited to their range and different types of units have different ranges. Player will be able to find new cities, build buildings and create new units. He/she also can conquer all of the above from his/her opponents (real or AI) through conquests. The player can choose to engage some or all of his/her units for the conquest. The player with greater unit strength will have a greater chance of winning the Conquest. Two emperors can choose to join hands with each to fight jointly with other Civilizations. The limitation to this alliance formation being the fact that this needs to be historically accurate.

There exists virtual market place where player could buy or exchange goods with other players. It uses virtual currency that is earned from each turn that player makes. Enemy's skills depend on the level of adventure i.e. higher the level, higher the skills of an enemy.

There are also different cell terrains, each of which has buffs for units that are onto it. It is very easy to go around the map and there is also a mini map that is a real time representation of the map, which helps you to have a better view of the whole map at any time, but scaled to be fit in the bottom context bar. At the end of your turn, you receive a fixed amount of money and a new "game day" begins. The game is really easy and fun to be played and if a player has difficulty in understanding the game rules, there is an instructions menu that will bring you the instructions screen, where everything that you are required to do as a player is described.

2 The Purpose of the Project

2a The User Business or Background of the Project Effort

Our project, The Rise of Civilization, is a turn based strategy game, wherein the player has an option to play either against an AI or against other players on your local area network. The game basically requires players to build a base where they build their buildings and units and eventually defeat all of the other players that are not in their team. The situation that triggered the development effort of the project is that in today's world, where we are surrounded by technology, students have lost the ability to think abstractly or logically and destroying social interactions. This, in-turn, is impacting their ability of problem solving and abstract thinking. Rise of Civilizations will provide the children with a platform to apply strategies in guiding troops, conquering territories, performing trade and managing resources, which will, in process, enhance their cognitive power. As it is a turn based game, each player has a number of things he/she can do during his/her turn, which includes: creating new buildings, spawning units, moving units and more. The user is presented with a map which gives a good view of different terrains and assists the user in building the units. The user can pan the map with the movement of the mouse. The user can pan the map left, right, up, down by moving the mouse to the left-most, right-most, up-most, down-most part of the map as well as using the mouse to pan the map, the user can use the keyboard to pan the map by using the left, right, up, down arrow keys to pan the map left, right, up, down. The keyboard can also be used for zooming in and zooming out of the map to get a wider view of the map. This is done by the + or - keys, which will increment the zoom accordingly. The buildings and units have stats that determine their effect on the game. For buildings they have stats for Health, Cost and Reward. For units they have stats for Health, Cost, Number of Units and Range. The health stat track how much damage a building/unit can take before it is removed from the game. The cost tracks how much money the player needs to create that building/unit. The reward stat tracks how much money the enemy player gets from destroying the building/unit. The damage statistics for units track how much health is removed from other units or buildings. Game ends when there are no opponents left in the game, or when last of player's "settlements" is captured.

2b Goals of the Project

The main goal of our project is to provide an opportunity to the people to apply strategies and conceptualize things. It will involve substantial and well defined planning, together with right actions which will lead to the strategy's success. The

players will engage in battles, conquer territories, perform trades and forge relationships. The Rise of Civilizations will compel and urge the users to think. It will also increase the social engagement among the players as the game is highly interactive. Hence, the players will be completely engaged and engrossed.

2c Measurement

The goals of our project The Rise of Civilizations are measurable. Our main goal is to simulate the civilizations so that people can apply different tactics to compete against one another in order to build a big empire by forging relations, conquering territories and utilizing resources. This will enhance their power to think logically. This goal can be measured in a way that if our project is really helpful, then there will be increase in users using the project. This will also attract people who are curious to know about the activities that used to take place in various civilizations. The success of the game will also attract different organizations to post their advertisements.

3 The Scope of the Work

3a The Current Situation

Currently there are many desktop based and web based historical games based on similar themes of building civilizations but our game will have game scenarios modeled around real historical facts which include land, army units, virtual currency and goods associated with that civilization and time period. The user will able to experience the trade and warfare practices of the time period he wishes to play in. The game will serve as a great platform for the user to learn about different civilizations and time periods and have fun at the same time. Unlike the other games available in market, the progress of the player in our game will not be based on his prior historical knowledge but on his/her knack of taking critical decisions regarding exploration, warfare and diplomacy. Moreover, the game basically requires players to build a base where they build their buildings and units and eventually defeat all of the other players that are not in their team. That is a really good and fun way for people to develop their logical thinking, because the game requires from you to start thinking a few moves ahead (like a chess game).Unlike previous versions of games, our game will give the user, the option to alternatively collaborate with other player(s) in order to increase their strength in terms of the land and manpower he owns. This will help the player improve his standing in the game data.

4 The Scope of the Product

4a Scenario Diagram(s)

Fig. 1.0: User Starts Game and initiates virtual marketplace trade

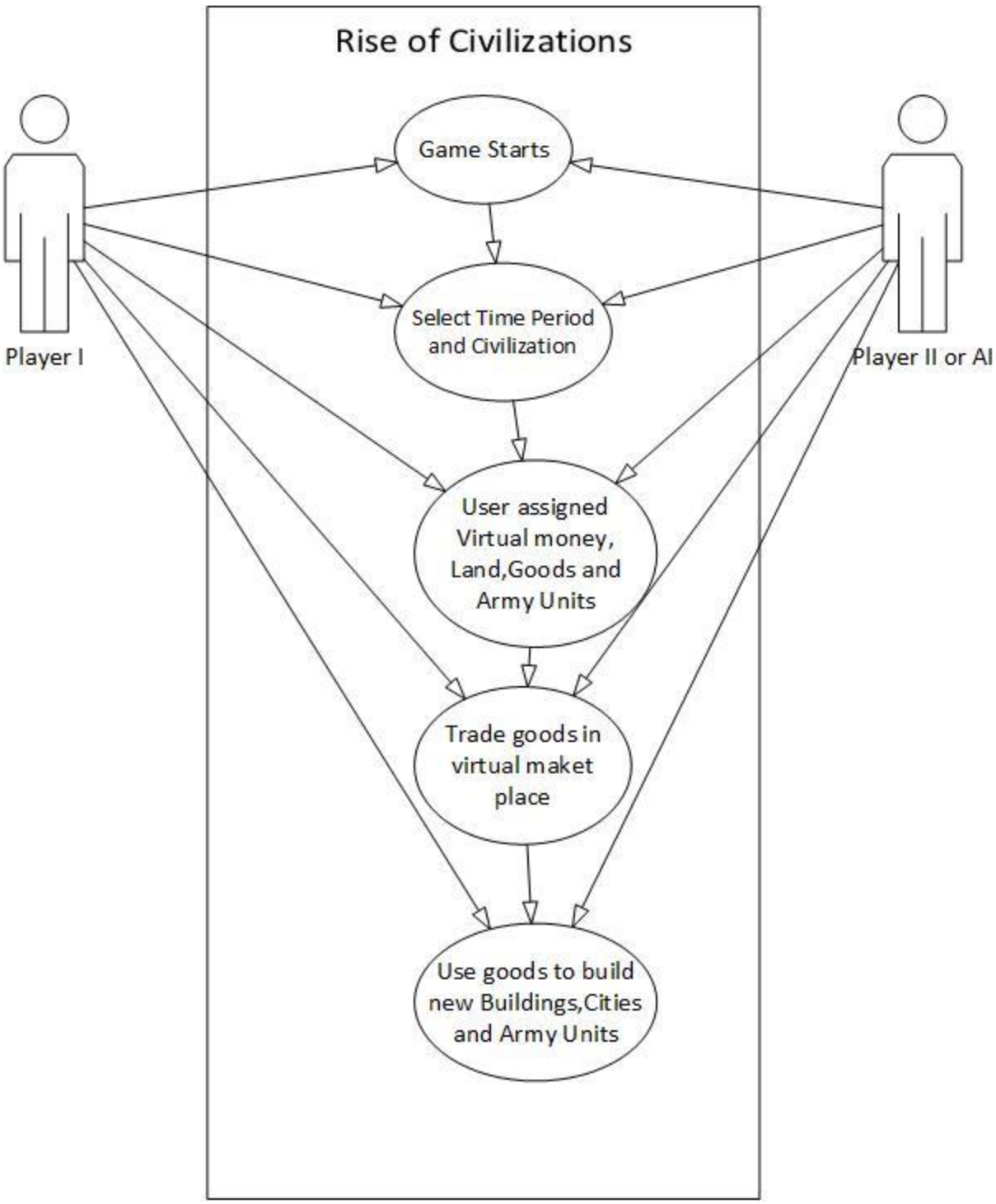

Rise of Civilizations
Player I
Player II or AI
Game Starts
Select Time Period and Civilization
User assigned Virtual money, Land,Goods and Army Units
Trade goods in virtual maket place
Use goods to build new Buildings,Cities and Army Units

Fig. 2.0: New User and Returning User Gameplays

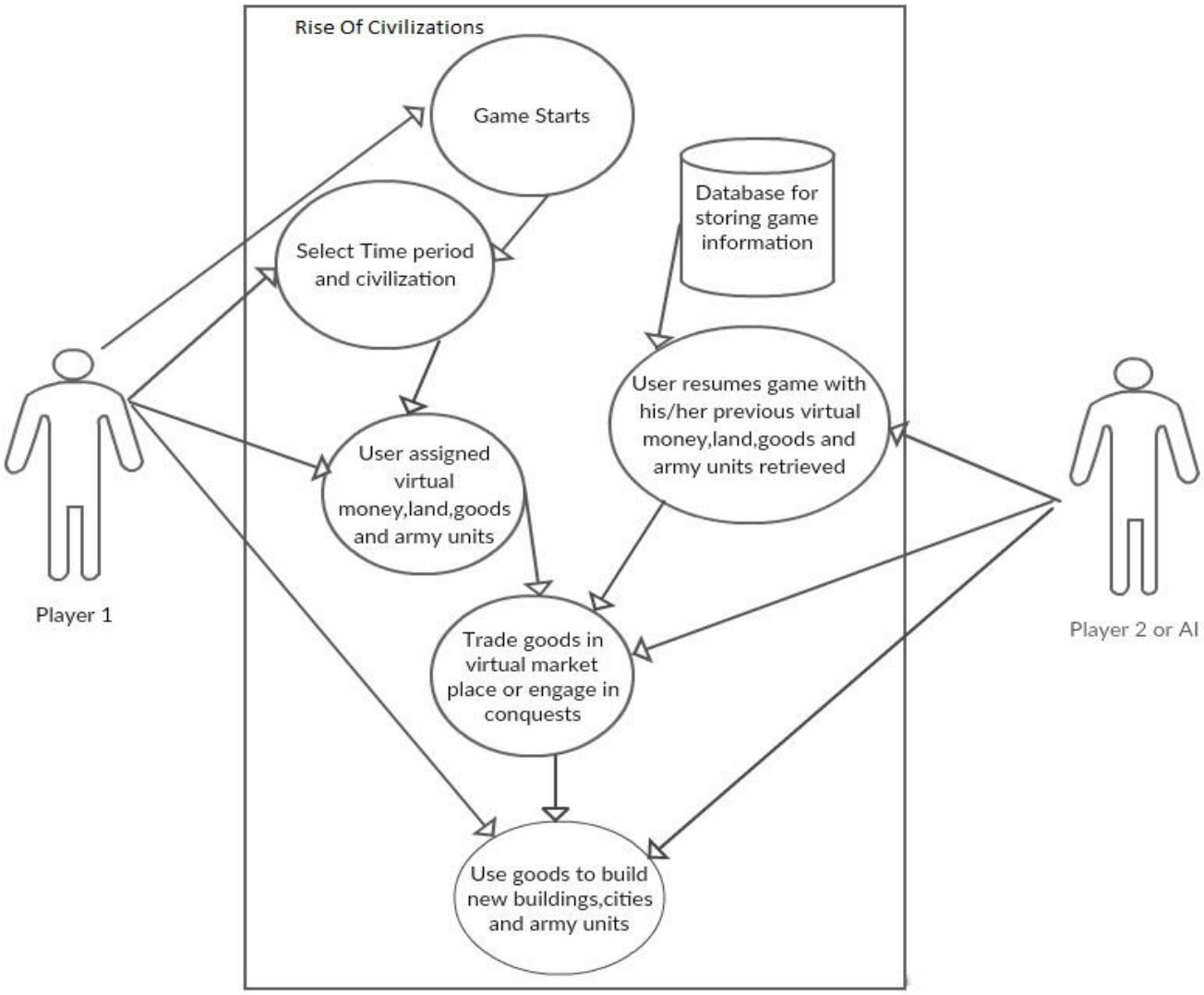


Fig. 3.0: User engages in conquest with opponent

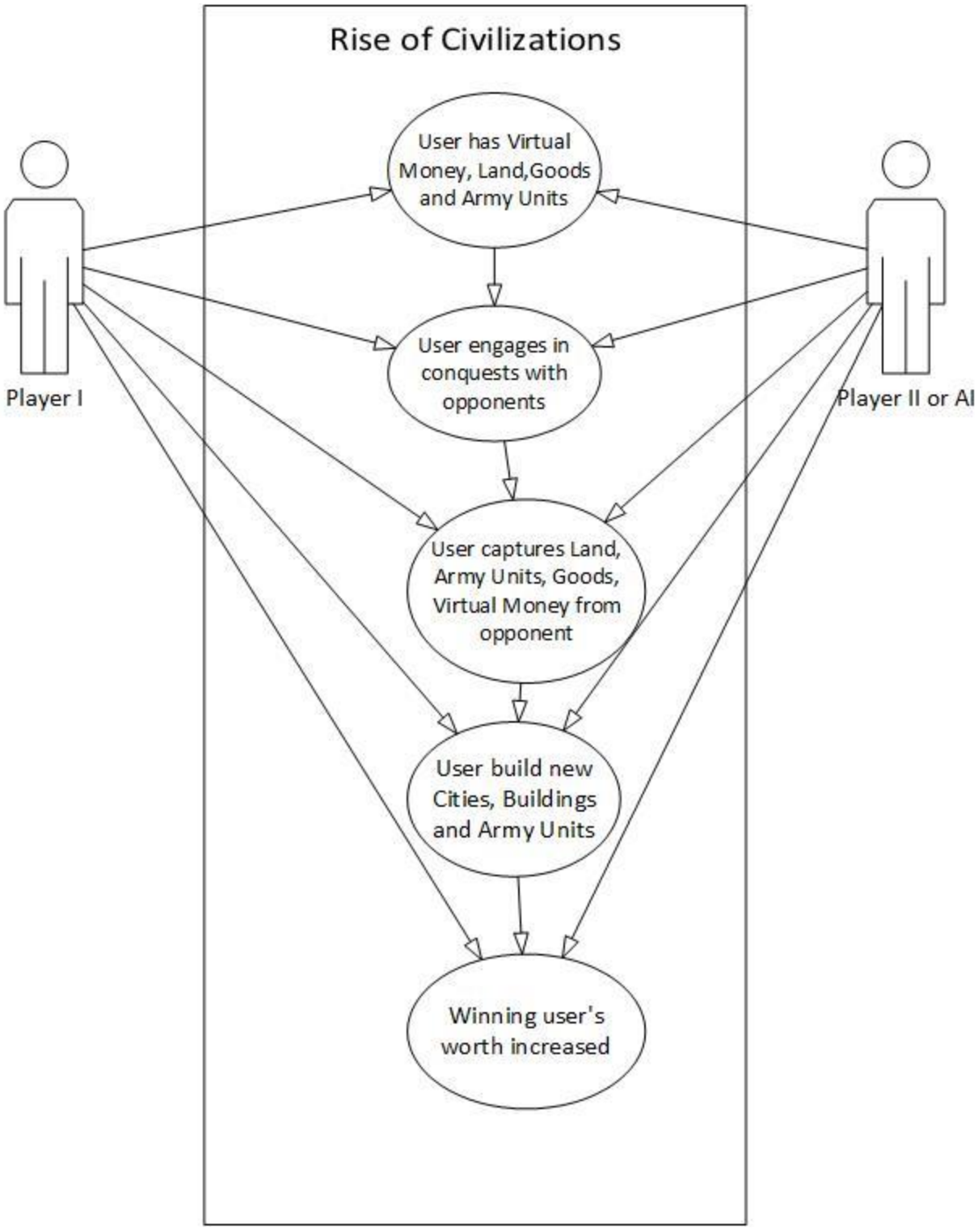


Fig. 4.0: Game summary from start to end of gameplay

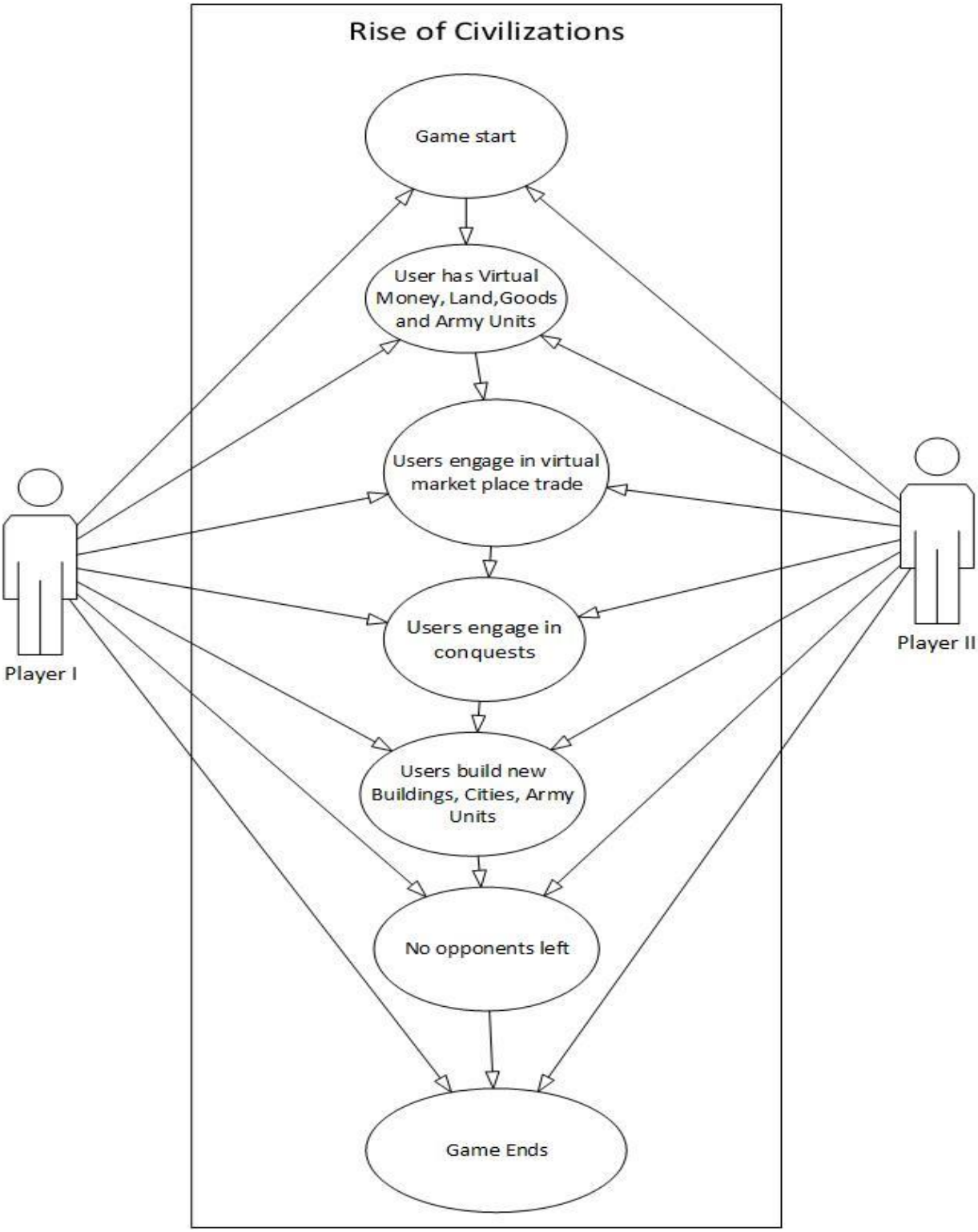
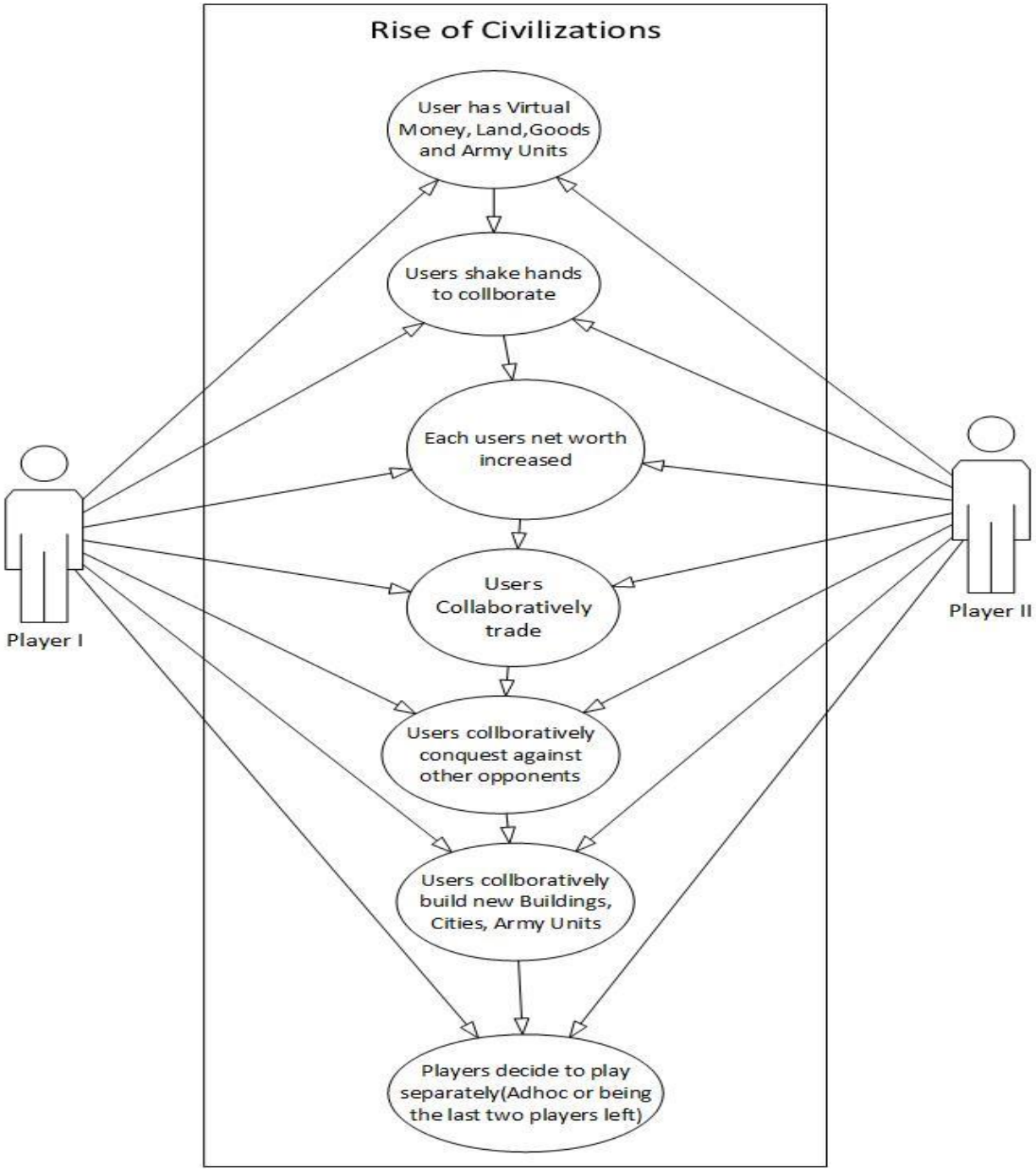


Fig. 5.0: Scenario where multiple users join hands to play collectively



4b Product Scenario List

1. Game Start
2. Players Engage in Trade
3. Players Engage in Conquests
4. Multiple Players Shake Hands
5. End of Game

4c Individual Product Scenarios

(1) **Game Start:** Rob is a new user and logs into the game for the first time. Rob is a given the option to select the time period and civilization in which he would like to begin his gameplay in. On selecting the time period and civilization, the game will load with the look and feel specific to that time period and civilization and the user is assigned with some virtual money, land, goods and army units specific to that time period. John is a returning user, who when logs into the game, the game retrieves his current holdings in the game and displays the virtual money, goods, land and army units that were owned by him at the end of his last game play.

(2) **Players Engage in Trade:** Rob and John after beginning their respective games may wish to engage in trade via the virtual market place. This gives the players an opportunity to strategically involve in trade with their opponents who help them build new buildings, cities, army units, expand their current holdings and hence improve their current standing in the game.

(3) **Players Engage in Conquests:** Rob and John after beginning their respective games, may wish to increase their current holdings by the means of attacking each other and acquire the other person's holdings. Attacking the other player will be strategic decision and will involve a thorough analysis of not only the player's own worth in terms of army units and manpower but also a prediction of the opponents power in terms of the same. The game will give Rob and John an opportunity to use their knack in terms of how many and which army units to deploy at appropriate fronts. During a conquest, a user may wish to use his current holding and engage in trade with other players in order to enhance his position in terms of army units and manpower. The winner of the conquest acquires the holdings of the opponent, thereby, increasing his worth in the game.

(4) **Multiple Players Shake Hands:** During the gameplay, Rob and John may strategically predict the worth of another user Daniel to be much greater than theirs. Rob and John, in case of inability to be able to do any new trades or conquests, may decide to shake hands and use their collective holding from here on to build new cites, buildings, army units or involve in trade collectively with other players. They may also strategically plan to attack Daniel and engage in a conquest with him. From there on, Rob and John may wish to play the game collectively, collaborating in each of their strategic decisions and putting their collective worth to improve their standing the game

(5) End of Game: Rob and John can continue playing the game till there are no more opponents or no more lands, goods, army units left to trade or conquer.

5 Stakeholders

5a The Client

The client for this game will give be anyone who is willing to invest in the gaming arena. The client will be in charge of maintaining the game and enhancing it whenever necessary. They will also be responsible for maintaining the website and ensuring that it runs seamlessly at all times. The client will also be responsibility of marketing the website and making it available to the customers. The onus lies on the developers to ensure that product meets the requirements set out by the client and to ensure a quality product in a timely fashion.

5b The Customer

This game will be available to anyone who is willing to purchase the product. It is intended to serve the purpose of enabling people to use their thinking and intelligence and to enhance their knowledge of different civilizations. Whether a person is an adolescent or an adult, this game caters to everyone who has a desire to improve their logical or abstract thinking. A key tenet of game-based learning is it can potentially motivate students to learning by making learning fun. This will be an effective tool in preparing the people to better plan their future and to take critical decisions. This will also inculcate leadership qualities in individuals and they will be better prepared to face-off the challenges. The students will be able to appreciate the geography in historical development, the impact of trade and economics, resource availability and expenditure.

5c Hands-On Users of the Product

The Hands-On users of the product can be anyone in the age group 10+.The game offers hours of play, challenging AI, and the ability to play and compete against remote users. The game will enhance people's knowledge about trade and economics and makes an effort to educate the users about the civilizations and the time periods it cover. It can also include adults who are interested in strategy games and who love adventure. This game can also be included in the college fest as one of the events.

User name/category: People in the age group 10+ who like playing strategy games.

User role: Actively involved in taking critical decisions, applying strategies and managing resources. They need to wisely utilize their resources and make decisions about which all units to seize in order to gain maximum benefit.

Subject matter experience: No prior knowledge is needed. It will be possible to learn the game by playing a few times or by carefully reading the rules.

Technological experience: The user must be capable of using a basic web connected computer and web browser.

Other user characteristics:

Physical abilities/disabilities: Person with weak eye sight is advised not to spend long periods of time in front of the computer as it will strain their eyes. As the game is highly addictive, the users are advised to take breaks after every hour.

Intellectual abilities/disabilities: The game is intended to be designed to be played by anyone for education/leisure purpose. Therefore, everyone is free to experience the game. No prior intellectual ability is needed. However, it is advised to see the doctor in case itching is experienced on playing the game for hours together.

Attitude toward job: People playing the game should have a positive attitude and should exhibit a desire to learn and improve. As the game is highly competitive, users are advised to curb their anger and not to hold any grudges, as it is just a game after all. They should cherish the opportunity to compete with others and strategically win over their empires.

Attitude toward technology:

Education: No prior education is required, but people with good logical ability will have an upper hand over their opponents.

Linguistic skills: No special linguistic skills are required to play this game at a high level.

Age group: This age group will most likely include people in the age group 10+.

Gender: This game is for both males and females.

5d Priorities Assigned to Users

- **Key users:** This game can be beneficial to all the users who have a desire to learn about different civilizations and the time periods they cover. This game will enhance people's knowledge about economics and trade. Also, this game can be played by strategy game enthusiasts and people who wish to compete against their friends. Therefore, our key users will comprise of all such people as they will expand and popularize our game worldwide.

5e User Participation

One of the things that the users can help with is the testing of the game. This will require the users to play the game for an hour during the day and report any glitches they experience with the functioning and performance of the game. The users can also

report any issues with the user interface. This will help the developers in fixing the bugs and improving the quality of the product.

5f Maintenance Users and Service Technicians

Maintenance users will include a group of technicians who will work together to fix and remove any defects discovered by the users. They will provide technical support to customers in a timely fashion and ensure that the website is up and running at all times and that there are no obstacles or impediments.

5g Other Stakeholders

- **Testers:** Testers will ensure to discover any bugs that may exist in the program. This will help keep the game updated and enhance user's experience.

- **Technology experts:** This group will include service technicians who will provide technical support to the team.

- **Marketing experts:** This group is extremely important as they will be responsible to market the game throughout the world. Their job will be to advertise the game, whether it is with online ad's or billboards.

- **Graphic Designers:** This group will maintain the graphics across different resolutions and operating systems. The group will be responsible for updating the game with the latest graphic drives across different platforms.

- **Developers:** This group will be responsible for maintaining the code across various release cycles, refactoring the code at times and keeping it as maintainable as possible for future releases.

6 Mandated Constraints

This section describes constraints on the eventual design of the product. They are the same as other requirements except that constraints are mandated, usually at the beginning of the project.

6a Solution Constraints

Content

This program must be solved primarily with Java. The program should be compatible on both Mac-OS and Windows (7, 8, 10). This is so that all users will be able to use this program.

Motivation

The constraints put on this program are too making it as accessible and easy to use as possible. Most of the users of this program will be young students that may not have the greatest ability to navigate difficult to understand user interfaces.

Examples

Constraints are written using the same form as other atomic requirements (refer to the requirements shell for the attributes). It is important for each constraint to have a rationale and a fit criterion, as they help to expose false constraints (solutions masquerading as constraints). Also, you will usually find that a constraint affects the entire product rather than one or more product use cases.

Description: The product shall use Artificial Intelligence technology.

Rationale: The client will be able to play against a machine that is able to compete with the client.

Fit criterion: There shall be levels to the Artificial Intelligence, so all the expected clients will have a suitable level of difficulty.

Description: The product shall operate using Windows 7, 8, and 10 as well as Mac-OS.

Rationale: The client base for this product is wide and this program should be able to run on all of these platforms.

Fit criterion: The product shall be approved as Windows 7, 8 and 10 compliant by the MS testing group as well be approved as Mac-OS compliant.

Description: The product shall be on a laptop and or desktop computer

Rationale: The product is to be marketed to people from all backgrounds looking to be educated in history.

Fit criterion: The product shall be able to run the following platforms, Windows 7, 8 and 10 and Mac-OS.

6b Implementation Environment of the Current System

Content

The product should be installed on desktop computers such as a Windows machine or Mac machine. The product should be compatible with Windows 7, 8 and 10 as well as any running version of Mac-OS.

Motivation

The product must be fully maximized on the screen as to not show anything else in the background for the full learning experience. Similar to how Microsoft PowerPoint slideshows are run. Once the client opens the product, the screen will be taken over by the product itself. The only way to get out of the product environment is to exit the program itself. This will allow the clients to get the full learning experience and be presented with challenges that cannot be referenced on the internet.

Considerations

The client should not be able to navigate away from the program as long as the program is running.

6c Anticipated Workplace Environment

Content

This product will be used by the clients on computers at schools, homes and libraries. The client crowd is mostly for students looking to learn more about history and this environment correlates with the above.

Motivation

The product must be school appropriate as to not show anything that might not be permissible on school grounds.

The product is to be used in a library or school; it must be extra quiet.

6d Scheduled Constraints

Content

This product must be completed by the start of the next summer so that teachers and supervisors may get accustomed to the product before the students.

Motivation

The client wants this product to be ready for the next school year for new students.

Considerations

This product must be finished by May 29th. It is critical for this product to be finished by this date so that the client may get used to it and be able to give it to their students.

If this deadline is not met, then the client will not be able to get accustomed to the software in time.

.

6e Budget Constraints

Content

The budget for the project is 1,000,000 USD.

Motivation

The requirements must not exceed the budget. The client fully expects all requirements and constraints to be met within this budget.

7 Naming Conventions and Definitions

7a Definitions of Key Terms

The glossary produced during requirements is used and extended throughout the project.

Player: The client on the user end of the game.

Computer: The artificial intelligence element of the game.

Simulation: Refers to gameplay situations carried out automatically

Flat: The basic land that takes up the majority of the map. Flat allows any unit or any building to be placed on it or move across it.

Water: Part of the map that only tanks can move pass through

Bridge: A way for units to cross the water terrain. Only marines can move across this terrain

Barracks: The building is used to create soldiers. Only four soldiers can be spawned around the barracks giving preference to the x-axis and then the y-axis

Factory: The building used to create machinery, only four machines can be spawned around the factory.

Soldiers: The basic unit, marines have no limit on how many can be created

Time Limit: The game is a turn based game. There will be 5 minute turns. After this amount of time has ended, the turn will be ended

Day: A day has taken place when all users have clicked the end turn button once

Player Money: This is how much a user has to build new units and buildings.

Current Player: This is the player who is currently playing the game. This is also shown in the team color.

Team Color: This is a reference so a user knows what their team color is.

Forest - Only marines can move across this terrain.

Mountains - Only marines can move across this terrain

.

7b Data Dictionary for Any Included Models

AI (Artificial Intelligence)

There are three main objectives that that AI attempts to do when it is its turn to play these are to maintain buildings, recruit units and attack enemies. This is the order in which the AI prioritizes its actions, maintaining buildings are top priority because buildings are required to train units and thus maintain an army. Secondary priority goes to unit recruitment so that the AI is always spending money on troops and finally attacking is done last.

When it comes to creating buildings the AI will attempt to maintain a set amount of them. At the start of each turn the number of factories and barracks are checked and if they are below a threshold then they are replaced at the next available opportunity (when money can allow). Each AI player tries to keep 2 barracks and 1 factory in operation so that they can recruit new tanks and marines. The number of each building was decided because of the unit limit on tanks (this being 5) means that a player can quite easily sustain this number through one factory. As marines are effectively unlimited two barracks are enough to create as many as allowed by the player's economy. Through testing we concluded that this arrangement was optimal and as such is the strategy that the AI implements.

Unit recruitment is a major part of the game and is something that the AI manages well and therefore effectively maintains and expands its army. Which units the AI recruits are decided by 3 factors. Firstly the amount of money that the AI player has at its disposal, it will check that there is more money available than the cost of the most expensive unit. If it turns out that one particular unit is too expensive for the player at that time then it is not considered. This is also where the AI determines if it has enough money to purchase any units and if not then it will skip to the attacking stage. Secondly the number of limited units on the map is assessed. The current maximum number of tanks allowed by a single player is 5, this is where this is checked and if the maximum number is reached then that unit is removed from consideration by the AI. Thirdly the priority of the individual unit is assessed; tanks are recruited in preference to marines. This is because of their increased strength and durability therefore they are prized higher. Marines are second choice as they are comparably less strong both in attack damage and durability.

Attacking is perhaps the most important aspect of the AI as without it there is no chance of winning the game. Each unit is controlled individually and chooses its own target to attack. This allows the AI to engage multiple targets in one turn which is important when playing a free for all against 3 opponents.

The way that a single unit attacks is as follows, the unit performs a breadth first search in a radial pattern on the map centered at the unit's position. It looks around it for an enemy unit or building and keeps expanding the search until a unit is found or it exhausts all cells of the map. This search only includes cells that are valid for that unit's travel, so a marine's search will not take it into a water terrain tile. This means that it only searches tiles that are accessible by that unit, this avoids the situation of a

unit selecting an enemy to attack that is unreachable and then getting confused about its route to it.

Once an enemy unit is focused as the target a route is plotted from the AI's unit to the enemy's one. The unit then proceeds to advance along this route until one of three states occur. It is possible that the route is blocked (by an allied unit), this can often happen with marines crossing water as the only way for them to do this is via a bridge. If this happens then the unit will move into the closest available cell and cease movement. Another situation is that the unit does not have enough move points to reach the target, if this happens then it just moves as far as it can and then stops. The final situation is that it manages to reach its target. If this happens then the unit attacks the target and then stops movement.

The game has a difficulty setting that gives the AI an advantage over the player. This is implemented through increasing the starting money of the AI player. This means that they have an advantage over the player from the start and as such are able to quickly mass units into an army.

Human Computer Interaction (HCI)

There are many strategy games out on the market at the moment, all of which function with similar, if not the same, human-computer interaction principles. This consists of making use of the mouse and keyboard input devices, along with various buttons on the user interface to aid the functionality.

The mouse input device is strategy games is used for the majority of the interactivity. The left-click of the mouse is used for selection, the right-click of the mouse is used for moving or attacking units, depending on whether the right-click was on an enemy unit, and the movement of the mouse is used to pan the map. The keyboard input devices allows the user to pan the map and zoom in and out of the map, if the functionality is there. The buttons on the user interface of the game itself are used to interact with the functionality of the selection. For example, when a unit is selected, the user will have buttons available for moving and attacking.

These principles haven't changed. They have been the same with the early strategy games compared to the top strategy games today. Rise of Civilization is no different. The user will use the left-click of the mouse to select a unit or a building. If a unit was selected with the left-click, the user can right-click in an empty cell to move a unit, since Rise of Civilization is a grid based strategy game, or the user can right-click on an enemy unit and attack it. The user can pan the map with the movement of the mouse. The user can pan the map left, right, up, down by moving the mouse to the left-most, right-most, up-most, down-most part of the map respectively. As well as using the mouse to pan the map, the user can use the keyboard to pan the map by using the left, right, up, down arrow keys to pan the map left, right, up, down respectively. The keyboard can also be used for zooming in and zooming out of the map to get a wider view of the map. This is done by the + or - keys, which will increment the zoom accordingly.

Fig. 6.0: ***Standard keyboard controls for gameplay.***

These two images are from the popular strategy game StarCraft 2. The image on the left is the panel which is shown when you select a unit, showing available orders to the unit. The panel on the right is shown when you want to build structures. Again, this is similar, if not the same, across almost all strategy games and Rise of Civilization is no exception.

Fig. 7.0: ***Standard gameplay buttons***

These two images are from Civilization. The image on the left is the panel which is shown when you select a unit, where you can choose to attack or move, and the panel of the right is shown when you select a Base building, which you can choose to build a Barracks or a Factory. Each panel has a help button which will show the instructions and the end day button which will end the turn of that player.

At a first glance, is not clear which building is which in the image on the right of the StarCraft 2 panels. This is why we decided to use text rather than images of the buildings as this is more informative to a user that is not familiar with the game. Although a help button isn't very popular on the panels in strategy games today, we wanted to make sure the user has access to help options whenever available. The same goes for the end day button, this needs to be available at all times. However, StarCraft 2 is a real-time strategy game and therefore, does not need an end day button!

We have tried to follow the interactive conventions of today's strategy games to make sure that the human-computer interaction principles are intuitive and easily accessible for users that are familiar with strategy games today.

8 Relevant Facts and Assumptions

8a Facts

Content

All data in this program must be factually correct backed up by at least three scholarly sources. There shall be no questions or problems in the product where the history is subjective to different cultures. All facts in this game must be objective and as to not offend any culture or religion.

Motivation

The game must be unbiased when it comes to history so as to when the game is created it must be facts that are accepted by mainstream society.

8b Assumptions

Content

A list of the assumptions that the developers are making. These assumptions might be about the intended operational environment, but can be about anything that has an effect on the product. As part of managing expectations, assumptions also contain statements about what the product will *not* do.

Motivation

To make people declare the assumptions that they are making. Also, to make everyone on the project aware of assumptions that have already been made.

Examples

Assumptions about which history subjects to cover.

Assumptions about the color scheme of the game.

The software components that will be available to the developers.

The availability and capability of bought-in components.

Dependencies on computer systems or people external to this project

The requirements that will specifically *not* be carried out by the product.

Considerations

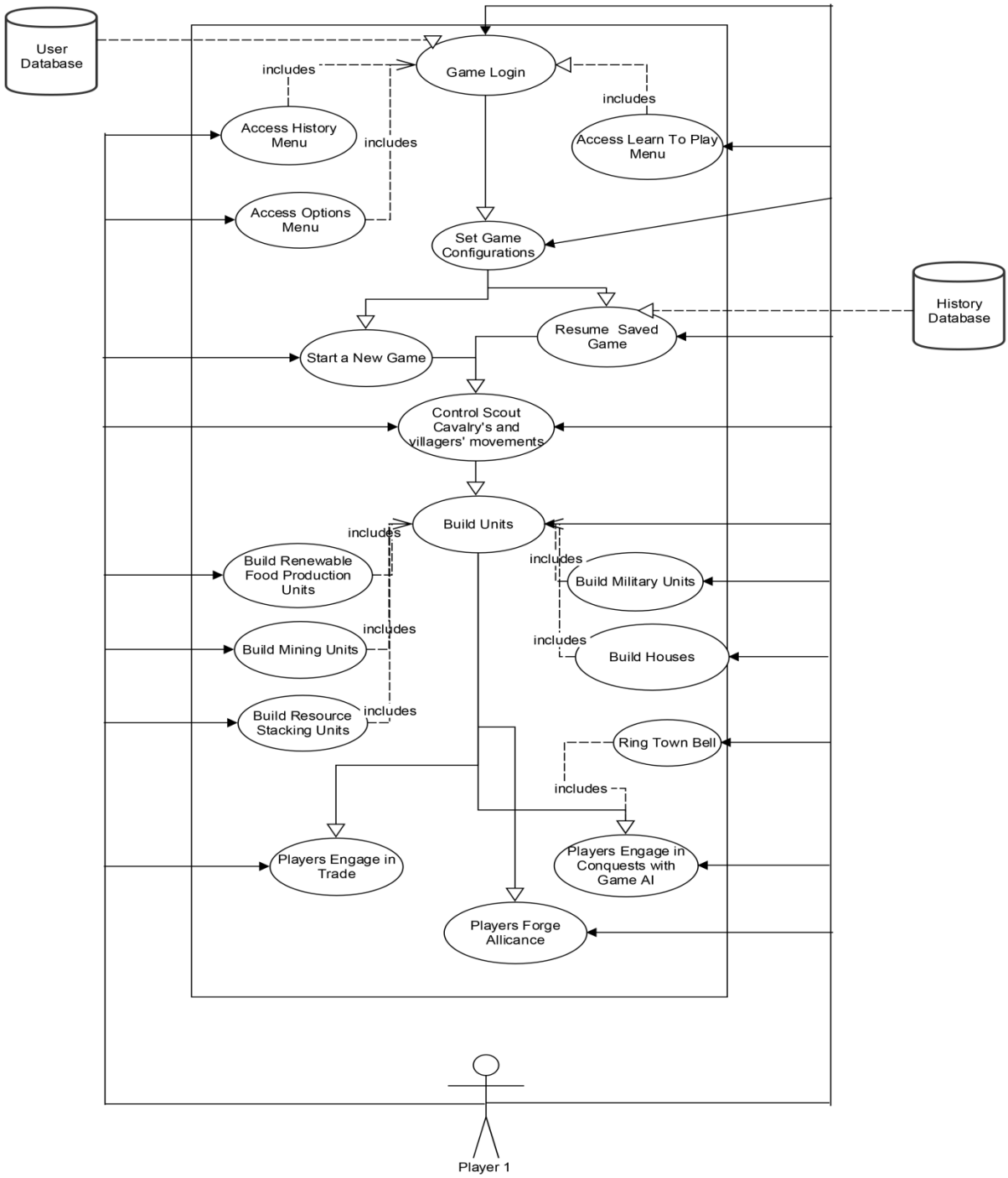
The client is not expecting to use the product in any new innovative way.

II Requirements

9 Product Use Cases

9a Use Case Diagrams

Figure 7.0: ***Use Case Diagram for a single user gameplay (Against AI)***



9b Product Use Case List

The following is the list of use cases, further developed in the next section:

1. Game Login
2. Accessing History Menu
3. Accessing Learn to Play Menu
4. Accessing Options Menu
5. Set Game Configurations
6. Start a Single Player Game
7. Start a Multiple Player Game
8. Start a New Game
9. Resume a Saved Game
10. Start Game
11. Control Scout Cavalry
12. Build Units
13. Build Houses
14. Build Renewable Food Production Units
15. Build Resource Stacking Units
16. Pause Game
17. Build Military Units
18. Build Mining Units
19. Players Engage in Conquests
20. Ring Town Bell
21. Player Wins the Conquest
22. Build Trading Marketplaces and Engage in Trade
23. Players Forge Alliance

24. End Game

9c Individual Product Use Cases

Use case ID: USC-01 Name: Game Login

Pre-conditions: The user should have "Rise of Civilizations" game installed on his system.

Post-conditions: The user is able to successfully login into the game and view game options.

Initiated by: Game Player (user).

Triggering Event: The user launches the game on his system.

Additional Actors: User Database.

Sequence of Events:

1. The player launches the "Rise of Civilizations" game on his system.
2. The system loads the game on the screen and gives the player the option to enter his/her username or create a new username.
3. The player enters a username.
4. a) The system searches for the existence of the username in database and displays existing game instances associated with that username (if any).
   b) The system gives the option to both types of users (new and returning), the option to start a new game.
   c) The system displays the various options for the user to customize his/her gameplay or to explore the various aspects of gameplay such as the "Options", "History" and "Learn to Play" menu options.

Alternatives: NA

Exceptions: The system will not accept a username if it already exists in the system.

Use case ID: USC-02 Name: Accessing History Menu

Pre-conditions: The user has successfully logged into the game.

Post-conditions: The user will be able to successfully view the "History" feature of the game.

Initiated by: Game Player (user).

Triggering Event: The player selects the "History" menu option, includes ***Game Login*** use case.

Additional Actors: NA

Sequence of Events:

1. The player selects the "History" menu option from the game screen.
2. The system displays the historical information about the eras and civilizations in which the game is intended to be played.

Alternatives: NA

Exceptions: The system will not enable the user to view the "History" feature of the game as a guest i.e. without logging into the game.

Use case ID: USC-03 Name: Accessing Learn to Play Menu

Pre-conditions: The user has successfully logged into the game.

Post-conditions: The user will be able to successfully view the “Learn to Play” feature of the game.

Initiated by: Game Player (user).

Triggering Event: The player selects the “Learn to Play” menu option, includes ***Game Login*** use case.

Additional Actors: NA

Sequence of Events:

1. The player selects the “Learn to Play” menu option from the game screen.
2. The system displays the user with the information about the different aspects of game such as "Marching and Fighting", "Feeding the Army", "Training the troops" and “Fighting Battles”.

Alternatives: NA

Exceptions: The system will not enable the user to view the “Learn to Play” feature of the game as a guest i.e. without logging into the game.

Use case ID: USC-04 Name: Accessing Options Menu.

Pre-conditions: The user has successfully logged into the game.

Post-conditions: The user will be able to successfully view the "Options" feature of the game.

Initiated by: Game Player (user).

Triggering Event: The player selects the "Options" menu option, includes ***Game Login*** use case.

Additional Actors: NA

Sequence of Events:

1. The player selects the "Options" menu option from the game screen.
2. The system displays the user with the option to set the game play configurations such as setting the keyboard keys, game volume or the mouse sensitivity.

Alternatives: NA

Exceptions: The system will not enable the user to view the "Options" menu as a guest i.e. without logging into the game.

Use case ID: USC-05 Name: Set Game Configurations

Pre-conditions: The user has successfully logged into the game.

Post-conditions: The user will be able to successfully set the game configurations required for gameplay.

Initiated by: Game Player (user).

Triggering Event: The player selects the "Start Game" option after filling out his username.

Additional Actors: NA

Sequence of Events:

1. The player selects the "Start Game" option on after filling out his username.
2. The system gives the user the option to customize the game with options such as population, difficulty level(Easy, Moderate, Hard), map style, starting age, civilization(Britons, Mongols, Goths, Persians), resources(Gold, Food, Stone),record game, building materials and player skill level.
3. The user sets the required game configurations and clicks on "Continue".
4. The system records the game configuration and loads the game for the player.

Alternatives: The user may continue his existing saved game in which case the game will load with the player's previously selected configurations which the user can edit.

Exceptions: The game will not allow the user to login into the game if the player has more than one instance of same game open on the same system with the same username.

Use case ID: USC-06 Name: Start a Single Player Game

Pre-conditions: The user has successfully logged into the game.

Post-conditions: The user will be able to successfully start a single player game.

Initiated by: Game Player (user).

Triggering Event: The player selects the "Start Game" option after filling out his username.

Additional Actors: NA

Sequence of Events:

1. The user selects the "Start Game" option after filling out his username and decides to play a new game.

2. The system loads the game configuration screen wherein the user can select to play in "Single Player" mode against the game AI.

Alternatives: The user may continue his existing saved game in which case the game will load with the player's previously selected configurations which the user can edit .

Exceptions: The system will not allow the user to convert a previously started single player game into a multiplayer game.

Use case ID: USC-07 Name: Start a Multiple Player Game

Pre-conditions: The user has successfully logged into the game.

Post-conditions: The user will be able to successfully start a single player game.

Initiated by: Game Player (user).

Triggering Event: The player selects the "Start Game" option after filling out his username.

Additional Actors: NA

Sequence of Events:

1. The user selects the "Start Game" option after filling out his username and decides to play a new game.

2. The system loads the game configuration screen wherein the user can select to play in "Multiplayer" mode and select his opponents from the list of available opponents.

Alternatives: The user may continue his existing saved game in which case the game will load with the player's previously selected configurations which the user can edit.

Exceptions: The system will not allow the user to convert a previously started single player game into a multiplayer game.

Use case ID: USC-08 Name: Start a New Game.

Pre-conditions: The user has successfully logged into the game.

Post-conditions: The user will be able to start a new game and successfully set the game configurations.

Initiated by: Game Player (user).

Triggering Event: The player selects to start a new game after filling out his/her username.

Additional Actors: NA

Sequence of Events:

1. The user selects the "Start Game" option after filling out his username and decides to play a new game.

2. The system loads the game configuration screen wherein the user can edit the game parameters.

Alternatives: The user may who may have a saved instance of a previously played game may still be able to start a new game.

Exceptions: The system will not allow the user to start a new game instance in case he/she already has 20 saved game instances on the same system.

Use case ID: USC-09 Name: Resume a Saved Game.

Pre-conditions: The user has successfully logged into the game.

Post-conditions: The user will be able to resume the saved instance of his previous gameplay and successfully able to edit the game configurations.

Initiated by: Game Player (user).

Triggering Event: The player selects to resume an existing game after filling out his/her username.

Additional Actors: History Database.

Sequence of Events:

1. The user selects the "Start Game" option after filling out his username and decides to resume a saved instance of his previous game.

2. The system loads the game configuration screen wherein the user can edit the game parameters and resume his previous gameplay.

Alternatives: NA

Exceptions: The system will not be able to resume a game that was started on a different network/LAN in case of an offline game play.

Use case ID: USC-10 Name: Start Game

Pre-conditions: The user has successfully logged into the game after setting all the necessary game configurations.

Post-conditions: The user has successfully started his game and can start building civilizations.

Initiated by: Game Player (user).

Triggering Event: The user clicks “Start Game” option after setting all the necessary game configurations.

Additional Actors: NA

Sequence of Events:

1. The user clicks “Start Game” option after setting all the necessary game configurations.

    2. a) The system should display the map, player's current score, a timer and the name and number of each of the resources (wood, food, gold, and stone) available with the player for constructing building and technologies.

       b) The system should display the player with the available villagers (who build buildings and resources) and scout cavalry. Also, the system should allow the player to select specific villager/scout cavalry by clicking on them. The system must also allow the player to guide their movements.

       c) The system should also display the user with a town center for building villagers and researching technologies.

Alternatives: The user may continue his existing saved game in which case the game will load with the player’s previously built units such as villages, army units, castles etc.

Exceptions: NA

Use case ID: USC-11 Name: Control Scout Cavalry

Pre-conditions: The user has successfully logged into the game after setting all the necessary game configurations and started a new game with the default game screen loaded.

Post-conditions: The user can control the movement of scout cavalry and view the health of villagers.

Initiated by: Game Player (user).

Triggering Event: The user clicks on any of the available villagers or the scout cavalry.

Additional Actors: NA

Sequence of Events:

1. The player clicks on a villager or a scout cavalry.

    2. The system will display the health and the weapons owned by the villager or the cavalry unit.

3. The user moves the scout cavalry to discover unexplored areas or resources (stones, animals, trees) or technology for building village/civilization.

    4. The system loads the various locations and the units available there and displays it on screen.

Alternatives: NA

Exceptions: The user will have to adjust the mouse sensitivity in case he is not able to synch with the movements of the characters on screen.

Use case ID: USC-12 Name: Build Units

Pre-conditions: The user has successfully started a new game with the home screen of the game successfully loaded and the user is able to control the movements of the scout cavalry and the villagers.

Post-conditions: The user can successfully create build units in the game.

Initiated by: Game Player (user).

Triggering Event: The user drags the desires unit's icon from the menu on to the screen.

Additional Actors: NA

Sequence of Events:

1. The player drags the desired unit's icon from the menu on to the screen and selects villagers and resources (wood, stones) to build the unit.

    2. a) The system engages the selected villagers to build the units and uses the resources from the available resources in order to build them.
    b) The system increases the civilization's population and resources as more and more units get added to it.

Alternatives: NA

Exceptions: The user will not be able to build any units in case he/she does not have any resources available at that point of time.

Use case ID: USC-13 Name: Build Houses

Pre-conditions: The user has successfully started a new game with the home screen of the game successfully loaded.

Post-conditions: The user can successfully create housing units in the game.

Initiated by: Game Player (user).

Triggering Event: The user drags the housing unit icon from the menu on to the screen, includes ***Build Units*** use case.

Additional Actors: NA

Sequence of Events:

1. The player drags the housing unit icon from the menu on to the screen and selects villagers and resources (wood, stones) to build houses.

    2. a) The system engages the selected villagers to build the housing units and uses the resources from the available resources in order to build them.
    b) The system increases the civilization's population (adds more villagers) as more and more housing units get added to it.

Alternatives: NA

Exceptions: The system will not allow the user to build any units in case he/she does not have any resources available at that point of time.

Use case ID: USC-14 Name: Build Renewable Food Production Units

Pre-conditions: The user has successfully started a new game with the home screen of the game successfully loaded.

Post-conditions: The user can successfully build renewable food production units such as farms and mills in the game.

Initiated by: Game Player (user).

Triggering Event: The user drags the icon for farm or mill unit from the menu on to the screen, includes ***Build Units*** use case.

Additional Actors: NA

Sequence of Events:

1. The player drags the farm or mill unit icon from the menu on to the screen and selects villagers and resources(wood, stones) to build houses

    2. a) The system engages the selected villagers to build the farm or mill units and uses the resources from the available resources in order to build them. The farms and mills act as renewable sources of food.
    b) The system increases the food resources available with the player as more and more food production units are added to the civilization.

Alternatives: NA

Exceptions: The system will not allow the user to build any units in case he/she does not have any resources available at that point of time.

Use case ID: USC-15 Name: Build Resource Stacking Units

Pre-conditions: The user has successfully started a new game with the home screen of the game successfully loaded.

Post-conditions: The user can successfully build resource stacking units such as lumber camps, town center(s) and docking stations in the game.

Initiated by: Game Player (user).

Triggering Event: The user drags the icon for lumber camp, town center or docking station from the menu on to the screen, includes ***Build Units*** use case.

Additional Actors: NA

Sequence of Events:

1. The player drags the for lumber camp, town center or docking station icon from the menu on to the screen and selects villagers and resources(wood, stones) to build a lumber camp or town center or a docking station.

    2. a) The system engages the selected villagers to build the selected resource stacking unit and uses the resources from the available resources in order to build them. Lumber camps act as a unit for stacking the wood whereas the docking station acts as a unit for stacking fishes which act as food resources. The town center can act as a central stacking point for any kind of resource.
    b) The system increases the food or wood resources available with the player as more and more resources are stacked into more and more resource stacking units.

Alternatives: NA

Exceptions: The system will not allow the user to build any units in case he/she does not have any resources available at that point of time.

Use case ID: USC-16 Name: Pause Game

Pre-conditions: The user has successfully started a new game with the home screen of the game successfully loaded.

Post-conditions: The player is able to pause the game at any point of time during the gameplay and resume the game with no loss of data.

Triggering Event: The player presses the pause key at any point of gameplay.

Additional Actors: NA

Sequence of Events:

1. At any point of time, the player wishes to pause the game and presses the pause key on the keyboard.
2. The system will pause the game with the player's progress in the game till that point saved.
3. The player resumes his/her game by pressing the pause key again.
4. The system resumes the game for the player and loads the game exactly from the point where it was paused.

Alternatives: NA

Exceptions: The game will not be able to continue from the point the user left in case the user pauses the game and there is a power outage or a system failure.

Use case ID: USC-17 Name: Build Military Units

Pre-conditions: The user has successfully started a new game with the home screen of the game successfully loaded.

Post-conditions: The user can successfully build military units such as barracks, archery ranges and also associate armed men, archers with them.

Initiated by: Game Player (user).

Triggering Event: The user drags the icon for barracks or archery ranges from the menu on to the screen, includes ***Build Units*** use case.

Additional Actors: NA

Sequence of Events:

1. The player drags the icon for barracks or archery range from the menu on to the screen and selects villagers and resources (wood, stones) to build them.

    2. a) The system engages the selected villagers to build the selected military unit and uses the resources from the available resources in order to build them.
    b) The system adds more and more armed men and war technologies as more and more military units are built, which in turn increases the player's military strength.

Alternatives: NA

Exceptions: The system will not allow the user to build any units in case he/she does not have any resources available at that point of time.

Use case ID: USC-18 Name: Build Mining Units

Pre-conditions: The user has successfully started a new game with the home screen of the game successfully loaded.

Post-conditions: The user can successfully build mining units such as gold or stone mining units.

Initiated by: Game Player (user).

Triggering Event: The user drags the icon for gold or stone mining units from the menu on to the screen, includes ***Build Units*** use case.

Additional Actors: NA

Sequence of Events:

1. The player drags the icon for gold or stone mining units from the menu on to the screen and selects villagers and resources (wood, stones) to build them.

    2. a) The system engages the selected villagers to build the gold or stone mining units and uses the resources from the available resources in order to build them.
    b) The system adds more and more gold and stone resources as more and more mining units are added to the civilization.

Alternatives: NA

Exceptions: The system will not allow the user to build any units in case he/she does not have any resources available at that point of time.

Use case ID: USC-19 Name: Players Engage in Conquests

Pre-conditions: The user has successfully started a new game and currently has some holdings with respect to military resources.

Post-conditions: The user is able to successfully engage in conquests or trade with other players. Initiated by: Game Player (user).

Triggering Event: The user deploys his/her military resources to attack some other player.

Additional Actors: NA

Sequence of Events:

1. The player decides to increase his resources and land by attacking his/her opponents for which he/she deploys his/her available military units by controlling their movement and the type of war equipment/technologies used.

    2. The system deploys the player's military units against the opponent's and they engage in a conquest.

Alternatives: NA

Exceptions: The system will not allow the player to engage in a conquest in case he/she does not possess any military units.

Use case ID: USC-20 Name: Ring Town Bell

Pre-conditions: The user engages in a conquest with some other player.

Post-conditions: The player is able to use the "Ring the Town Bell" feature in order to alert his/her villagers to seek shelter in the event of an attack.

Triggering Event: The player clicks on “Ring the Town Bell” button on the menu, includes ***Players engage in conquests*** use case.

Additional Actors: NA

Sequence of Events:

1. The player rings the town bell so as to alert the villagers in his civilization of an upcoming attack

    2. The system will hide the villagers in the housing units and arm them with bow and arrows so that they can protect themselves during the conquest.

Alternatives: NA

Exceptions: The system will not allow the user to use the “Ring the Town Bell” feature in case there is no attack on the player’s civilization.

Use case ID: USC-21 Name: Player Wins the Conquest

Pre-conditions: The user engages in a conquest with some other player.

Post-conditions: The player is able to see the defeated opponents holdings added to his holdings, marked as captured.

Triggering Event: The player wins the conquest, includes ***Players engage in conquests*** use case.

Additional Actors: NA

Sequence of Events:

1. The player engages his military units in conquest with another player/opponent and is able to win the conquest.

    2. a) The system marks the opponent's resources as captured and now as the part of player's resources.
    b) The system increases the player's land, food, military, gold, stone, wood and other resources increase as a result of this win.

Alternatives: NA

Exceptions: NA

Use case ID: USC-22 Name: Build Trading Marketplaces and Engage in Trade.

Pre-conditions: The user has successfully started a new game and currently has some holdings with respect to food, gold, stone, technologies or weapons.

Post-conditions: The player is able to trade with other players.

Triggering Event: The player drags the trading place icon from the menu on to the screen to build trading marketplaces in order to engage in trade with other players, includes ***Build Units*** use case.

Additional Actors: NA

Sequence of Events:

1. The player drags the icon for trade marketplace units from the menu on to the screen and selects villagers and resources (wood, stones) to build them.

2. a)The system engages the selected villagers to build the trading marketplaces and uses the resources from the available resources in order to build them
   b) The system enables the user to now engage his trading market place with some other player's in order to engage in trade with him/her.
   c) The system will allow the user to use the chat feature to facilitate communication between players in order to initiate trade between them.

3. The player selects on of the players to engage in trade with him/her.

4. The system will implement scorecard for each trade to help evaluate the trade for each player.

Alternatives: The player can engage in conquest in order to capture opponent's resources.

Exceptions: The system will not allow the player to engage in trade in case he/she does not possess a trading marketplace unit or he/she does not possess any resources (to be used in trade).

Use case ID: USC-23 Name: Players Forge Alliance.

Pre-conditions: The user has successfully started the game.

Post-conditions: The user is able to forge an alliance with other players.

Initiated by: Game Player (user).

Triggering Event: The user clicks on "Forge Alliance" option on the menu.

Additional Actors: NA

Sequence of Events:

1. The player decides to forge an alliance with other players to increase their strength and resources and selects the "Forge Alliance" option on the menu.

2. The system displays the list of other players logged into the game along with their current holdings in the game.

3. The player selects one or more players from the list of players (one player at a time) with whom he intends to forge an alliance.

4. The system pops up the chat window for the current player to communicate with the other players.

5. The current player and one or more other players consent to forge an alliance.

6. The system would display the two players as a team and would combine the two player's holdings and display them as combined holdings of the two players.

Alternatives: NA

Exceptions: The system will not allow the current player to forge an alliance with all the other players available at that particular time, as in that case no opponent will be left in the game for the current player to play against.

Use case ID: USC-24 Name: End Game

Pre-conditions: The player has played the game in all eras and has some current holdings in the game in terms of land and resources.

Post-conditions: The player wins the game and the game ends.

Initiated by: Game Player (user).

Triggering Event: The user acquires all the available resources and land available in the game.

Additional Actors: NA

Sequence of Events:

1. The player captures the last piece of land available on the game map.

    2. The system identifies that the current player has occupied all the available resources and land available in the game and declares the player to be the winner of the game and ends his current game.

3. The player wishes to play again and starts a new game.

    4. The system starts a new game for the current user.

Alternatives: The user, after winning one of the instances of the game, may wish to continue another instance of his saved game rather than starting a new game.

Exceptions: The game will not allow the user to have more than 20 saved games on a single system. So, the player may not be a able to start another new game in case he already has the number of saved games on the system equal the maximum number allowed by the game.

10 Functional Requirements

Requirement #: 1

Description: The system must allow the player to play in *Single player* (Play against Game AI) or *Multi Player* modes.

Rationale: To be able to allow the user to start a new game or run a previously saved game.

Fit Criterion: A user is able to click on either the "Single player" mode or "Multi player" mode and either start a new game or run a previously saved game.

Requirement #: 2

Description: The system must also allow the player to change gameplay settings.

Rationale: To be able to allow the user to set the game volume, keyboard control keys and mouse sensitivity.

Fit Criterion: A user is able to click on the "Options" button to set the game settings as per their preference.

Requirement #: 3

Description: The system must allow the player to gain knowledge about a particular era or civilization.

Rationale: To be able to allow the user to make better informed decisions in order to gain an edge over their competitors.

Fit Criterion: A user is able to click on the "History" button to view and learn about the various eras and civilizations.

Requirement #: 4

Description: The system must allow the player to gain information different aspects of the game.

Rationale: To be able to allow the user to learn about the various activities involved in the game so that they are well prepared to take on the challenge from the competitors.

Fit Criterion: A user is able to click on the "Learn to Play" button to learn about the various activities involved in the game such as "Marching and Fighting", "Feeding the Army", "Training the Troops" and "Fighting Battles".

Requirement #: 5

Description: The system must allow the player to customize the game.

Rationale: To be able to allow the user to set the gameplay parameters such as the difficulty level, the map style, the population size, the location, etc.

Fit Criterion: A user is able to click on the "Standard game" button to set the gameplay parameters such as population size, difficulty level (Easy, Moderate and Hard), map style, starting age, civilization (Britons, Mongols, Goths and Persians), resources (Gold, Food and Stone), record game, building materials as per their skills.

Requirement #: 6

Description: The system must allow the player to pause the game at any moment and to resume immediately thereafter.

Rationale: To be able to allow the user to pause the game if they are feeling tired or in case of an emergency.

Fit Criterion: A user is able to press the "P" button in order to pause the game and to press "P" again to resume the game.

Requirement #: 7

Description: The system must allow the player to begin the game.

Rationale: To be able to allow the user to start the game and engage in conquests.

Fit Criterion: A user is able to press the "Start Game" button to begin the game.

Requirement #: 8

Description: The system must display a map for the player, along with the player's current score, timer and the name and number of all resources available.

Rationale: To be able to allow the user to do better planning.

Fit Criterion: A user is able to view their current score, keep track of the time and the resources available to them, and also, to track the location of their opponents.

Requirement #: 9

Description: The system must display the health of the villagers and the weapons owned by the scout cavalry.

Rationale: To be able to allow the user to access the health of their population.

Fit Criterion: A user is able to view the health of the villagers and the scout cavalry by clicking on them.

Requirement #: 10

Description: The system must display the player with a town center.

Rationale: To be able to allow the user to store food and research technologies.

Fit Criterion: A user is able to build the town center by clicking on the "Town center" icon.

Requirement #: 11

Description: The system must present a map to the player.

Rationale: To be able to allow the user to discover unexplored areas by moving the scout cavalry to different places on the map.

Fit Criterion: A user is able to view and scroll through the map provided in the bottom right corner.

Requirement #: 12

Description: The system must allow the player to be able to select villagers for building units and performing tasks.

Rationale: To be able to allow users to build houses, buildings, lumber centers, farms, docks, gold, mills and stone mining units with resources available to them.

Fit Criterion: A user is able to select villagers by clicking on them.

Requirement #: 13

Description: The system must allow the player to select villagers to build mills and farms.

Rationale: To be able to allow users to build mills and farms as renewable sources of food or to gain health and stack food from farming or hunting in the town center.

Fit Criterion: A user is able to select villagers by clicking on them.

Requirement #: 14

Description: The system must allow the player to build army units.

Rationale: To be able to allow users to build army units such as barracks, archery ranges, castles and associate armed men or archers with them

Fit Criterion: A user is able to select villagers by clicking on them and guiding them to the location where construction needs to be done.

Requirement #: 15

Description: The system must keep the user's resources up-to-date.

Rationale: To be able to allow users to view the most up-to-date information about their resources.

Fit Criterion: A user is able to view the resources available to them.

Requirement #: 16

Description: The system must allow the player to engage in conquests with other players.

Rationale: To be able to allow users to engage in conquests with other players and deploy their army units in a defensive or rigorous attacking response.

Fit Criterion: A user is able to add and select soldiers and guide them towards the adversaries.

Requirement #: 17

Description: The system must allow the player to use the available technologies.

Rationale: To be able to allow users to make better informed decisions and try out different strategies.

Fit Criterion: To be able to allow users to see the units or type of army units (archery units) available with other player(s) in a conquest.

Requirement #: 18

Description: The system must allow the player to be able to use the "Ring the Town Bell" feature.

Rationale: To be able to allow users to save their people for being killed in the battle.

Fit Criterion: To be able to allow users to click on the “Ring the Town Bell” button in order to alert their villagers to seek shelter in the event of an attack.

Requirement #: 19

Description: The system must allow the player to add new villagers.

Rationale: This will allow the users to have back-up in case they lose their villagers in the battle.

Fit Criterion: To be able to allow users to add villagers from the menu.

Requirement #: 20

Description: The system must allow the player to build castles

Rationale: This will allow users to have better defense mechanism.

Fit Criterion: To be able to allow users to build castles by clicking on the build castle icon and guiding villagers to the construction site.

Requirement #: 21

Description: The system must allow player to be able to trade with other users throughout the game.

Rationale: This will allow users to forge alliance with other players.

Fit Criterion: To be able to allow users to use the chat feature to help facilitate the trade of goods.

Requirement #: 22

Description: The system must allow players to advance to the next level.

Rationale: This will allow users to enter new eras and civilizations and learn about them.

Fit Criterion: To be able to allow users to click on the escape button and select the level they wish to play provided they fulfill the level's requirements.

Requirement #: 23

Description: The system must end the game when the player has played in all the eras and has captured all the available resources.

Rationale: This will allow users to end the game.

Fit Criterion: To be able to allow users to capture all of their opponent's resources and emerge victorious.

---

11 Data Requirements

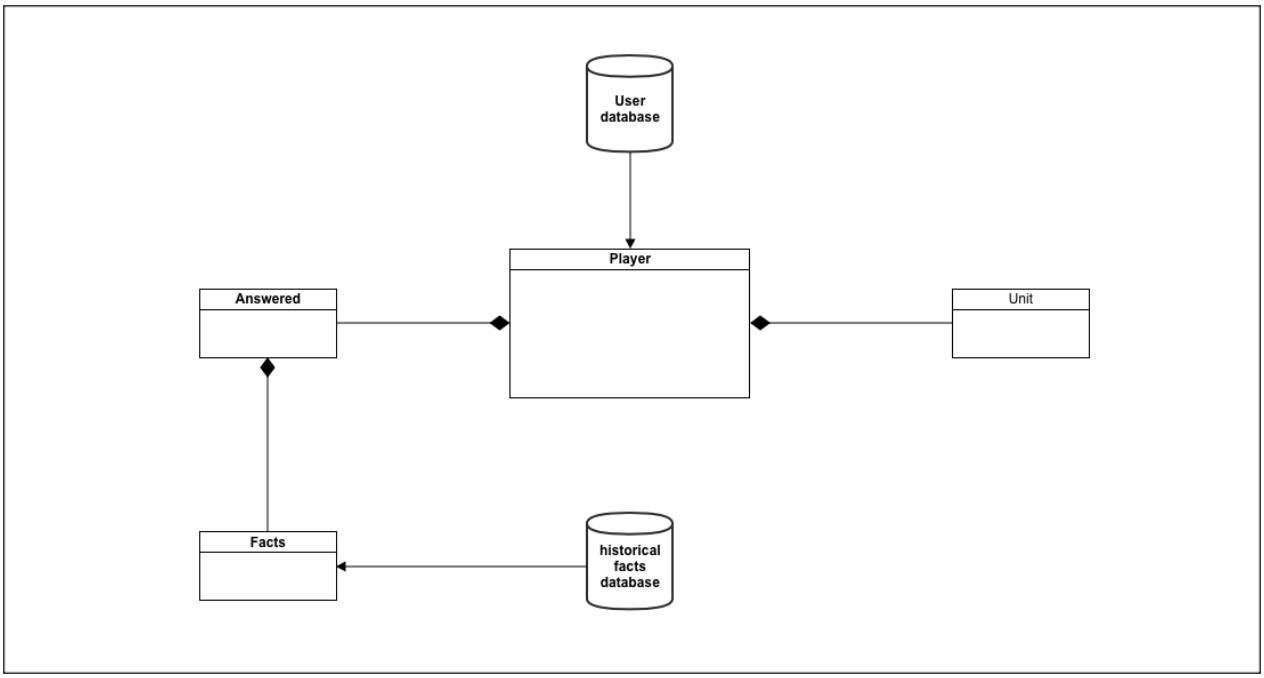


Figure: Diagram to illustrate data requirements for the game.

There will be two databases, a user database and the history facts database. The user database consists of the identification information of all the users. The historical facts database contains the information of all real historical facts that happened in Human history, and questions about them. Also, there will be at least four classes, the player class, the unit class, the facts class, and the answered class. The answered class will contain the information about all questions that user has answered correctly. Unit class contains information amount and type of units that are available to the user.

12 Performance Requirements

12a Speed and Latency Requirements

- The Rise of Civilization should execute graphics at least 120 fps (frames per second).
- The Rise of Civilization should require a minimum of 1.4 gigahertz (GHz) processor.
- The Rise of Civilization's load time should not be more than 5 seconds between levels.
- The time to reach the main menu upon the game launch should not exceed 15 seconds.

12b Precision or Accuracy Requirements

- The movement of the characters in the game and the attack at the enemy should be precise.
- The leaderboard should be updated daily to keep track of the leaders.
- The user should be able to return to their saved game as they left if before.
- The amount of user's resources should be accurate after every trade.
- The map should display the accurate view of the area.

12c Capacity Requirements

- The game should be able to handle up to five users logged in simultaneously at one time.
- The size of The Rise of Civilization should not exceed one gigabyte.
- The database should be able to hold up to date information on up to 100 of the top users in the game.

13 Dependability Requirements

13a Reliability Requirements

- The game should not fail or have any glitches more than once a day
- If the game does fail, it should take 15 seconds to reload to exact same game environment the user was in.
- The maintenance should take place once a week and the user must be notified of the time of maintenance two weeks before.
- The user should be able to access their profile and play the game during the maintenance period, even if any changes are restricted.

13b Availability Requirements

- The game should be available to the users 24 hours a day, 365 days a year.
- New versions of the game should be made available once a year.
- New updates should be made available every two months.
- The product must be up and functioning properly for 95 percent of the time

13c Robustness or Fault-Tolerance Requirements

- The game must notify all the users if the database connection is down which may restrict returning users from logging back in or restricting users from playing The Rise of Civilizations in multiuser mode.
- If the user's internet disconnects or the network goes down, the game should allow the user to continue playing their game until connection reestablishes, referred to as "offline mode", which will allow the users to normally save the game.

13d Safety-Critical Requirements

- The game should not cause any direct physical damage to the user
- The game should use well-formed colors for the design of the game, so not to disturb or cause any harm to the user.

- The game should not display bright colors.

Maintainability and Supportability Requirements

13e Maintenance Requirements

- The product should be maintained by the maintenance team to ensure that all the bugs discovered after the releases are fixed promptly

- Any patches for bugs should be released within 5 days of the day that the bug is released.

- The maintenance team should review and consider all of the suggestions for improvements that users suggest in the suggestion box.

- If the game is down for any reason, the maintenance team should have it up and running within 2 hours of the exact time that they are notified of the problem.

- There should be a member of the maintenance team on duty at all times, even after the first release.

13f Supportability Requirements

- The product should provide assistance through email for users at all times. The maintenance team will be in charge of addressing the emails initially.

- The user should receive a reply to their email within 10 hours.

- After the third release, the users should be able to receive support through the telephone. This will be the responsibility of the customer support team, which will be formed after the third release of the game. The third release signifies that the game has garnered good user traffic and is popular among the users.

- If the user forgets their identification, the game should email the user their identification information through email, provided that they correctly answered a security question.

13g Adaptability Requirements

- The website of the game (which is decided by the distributors), should operate normally on Mozilla Firefox, Google Chrome, Safari, and Internet Explorer.

- The product should be compatible with the latest versions of the Mac OS X, Windows XP, Windows & and Windows 8 as well as the latest Linux OS.

- All features of the previous versions of the product should be compatible with the latest version.

13h Scalability or Extensibility Requirements

- The product should be able to support nearly 5,000 users during the first 6 months of the first release.

- After one year, the game should be able to support 15,000 users.

13i Longevity Requirements

- The features of the game should continue to improve over time, removing any bugs when they are discovered and introducing newer, better features to enhance gameplay.

14 Security Requirements

14a Access Requirements

- Only the developers should have access to the source code.

- Users' personal information such as username, password and email addresses should not be accessed by anyone other than the users.

- The user database should only be accessed by the leader of the maintenance team.

- A new version should not be released without the consent of the team leader. This may be the owner, developer or a manager.

14b Integrity Requirements

- The data in the user's database should not be manipulated, or made available to public at any time.

- A concerted effort should be made to protect user's data and no data should be lost at any point.

- The maintenance team should also conduct daily checks every 24 hours to ensure that none of the data has been manipulated or modified.

- The product should protect itself from outside users, or hackers, attempting to manipulate the data.

- Once the users count exceeds 100,000, all of the user's data should be backed up in a separate database to ensure safety.

- The backups should take place every week to ensure that the backup database is up to date.

14c Privacy Requirements

- The privacy policy should be made available to the user at the outset of the game, on the main menu. This can simply be done under the "Policies" section.

- The product should notify the users through email if any changes are made to the privacy policy.

- Besides the information/statistics provided on the Leaderboard, no other information about other users should be made available to any users.

- The distributors of the game should also comply by the privacy laws in their respective state, company and country to ensure that the user data is safely protected and cannot be violated.

15 Usability and Humanity Requirements

15a Ease of Use Requirements

- The game should be easily accessible by clicking on the game icon on the Desktop of user's computer.

- The game should accurately display the users ranking and amount of virtual currency he holds.

- The game should accurately display consistent information for users every time they log in.

- The game should be appropriate for children aged 10 and up.

- Returning users should correctly provide their username and password.

- If the user forgets their password, the game should send the user the password to the email address that the user provided while registering.

- The game should be usable for users with little to no knowledge of history, warfare techniques, or trading experience and users that have never played the game before.

15b Personalization and Internationalization Requirements

- The game should only be available in English.

- The game should allow the users to select a username and a password of their choice while registering for the game.

- The game should allow the user to personalize their profile. This includes, but is not limited to, selecting a personalized background, different font; different colors of font and other dialog boxes.

15c Learning Requirements

- The game should be usable for person with no prior knowledge of history, warfare techniques or any prior experience with the game.

- The game should provide a tutorial for users that prefer to understand the basics of the game before attempting to play the game.

- The game requires that a user possesses the very basic computer skills.

15d Understandability and Politeness Requirements

- The "Help" section of the game should be detailed and concise providing a novice user with proper knowledge and understanding of the rules to proceed with the game.

- The game should not provide any unnecessary, low level background information that will confuse the user. This information can include, but is not limited to, the source code and the implementation details.

- The game should not introduce any terminology that is not part of Standard American English.

- The game should consist of "Go Back" button to make it simple to navigate between the different game windows.

- The game should consist of the word "please" when asking for username and password.

- The game should not use any terminology that is offensive to any individuals or groups across all cultures. If such case is determined to be true after the release of the game, the developers should immediately make changes within 24 hours of the discovery of the offense.

- Any images or logos used by the game should not be offensive to any individuals or groups across all cultures.

15e Accessibility Requirements

- The game should be accessible by all individuals with common disability which does not restrict them from operating a computer.

- The game should provide the following disclaimer at the beginning of the game to all users: "Any users with visual disability must restrain themselves from using the game for more than three hours."

- The game should avoid using red or green colors to make it easier for individuals that are red-green colorblind.

- The game should avoid using flashing, moving or rotating displays throughout the game.

15f User Documentation Requirements

- The product should include a user manual in electronic format.

- The product should send each user the user manual in a text file format to the email address that the user provided while registering.

- All graphs, charts and pictures in the user manual should include a detailed text description.

- The product should also provide each of the user with contact information that may be needed when potential bugs are discovered by the users. This may be done through the user manual or directly on the website that is hosting the game.

- The user manual should be updated for every new release to account for the new features or other changes that may have occurred.

15g Training Requirements

- No prior knowledge is required to excel at this game, but tutorial/training is provided to users who wish to learn about the aspects of the historical facts and other features of the game before attempting the game themselves.

- The tutorial that is provided should be interactive, include videos, and text descriptions that decreases the user's learning curve.

16 Look and Feel Requirements

16a Appearance Requirements

- The product should maintain a "historical" look and try to portray the real world feel of the time of the era user is playing in.

- The product must allow the user to personalize their profile, selecting from the various options for the backgrounds, fonts and font sizes.

- The product should be attractive to users of all ages.

- The products should not display any images deemed to be inappropriate.

- The product should not display any content deemed to be inappropriate. This includes, but is not limited to, profanity and any other inappropriate words or phrases.

16b Style Requirements

- The product should make the user feel as if they are participating a real historical environment.

- The game environment should appear trustworthy for first time users.

17 Operational and Environmental Requirements

17a Expected Physical Environment

- To access the game, a person should have access to a computer that has a keyboard, mouse and an access to an internet connection.

- The game should be downloadable from the games' webpage.

- The game should be usable in a brightly lit room or a dark room.

- The game should be accessible at any time of the day, and at any location.

17b Requirements for Interfacing with Adjacent Systems

- The game should be downloadable through all new and older versions of the following web browsers: Mozilla Firefox, Internet Explorer, Safari and Google Chrome.

- The game profile should be accessible through all smartphones that support internet connection and web browsers.

- Any internet providers should be a viable choice to access the game.

- The database that contains the historical facts data should have enough disk storage to contain the data of at least 1000 facts and 20000 questions.

- The user database should contain enough disk storage space to contain information of at least 100,000 users.

- The hosting website server, which is of distributor's choice, should consist of enough bandwidth to be able to support 5,000 users simultaneously.

17c Productization Requirements

- The game is available online through a website of the distributor's choice.

- The game should be accessible and usable for a person with little (basic) knowledge of computers.

- The game should be accessed through a web browser.

- The game should not only be used where an internet connection is available.

17d Release Requirements

- There should be at least one new release per year to enhance the current features of the game, fix any current bugs and to introduce new features.

- The maintenance releases should be offered every six months to keep information in the databases updated.

- Maintenance team must fix all bugs within 10 days of the date that the bug is identified.

- All previous features of the game should still operate normally on any of the new releases.

- All of the releases should follow proper testing with various test cases to verify that new features function normally.

18 Cultural and Political Requirements

18a Cultural Requirements

- The Rise of Civilization should not purposefully intend to offend individuals of any particular culture

- The only language used throughout the game should be English, so not to be biased towards any particular culture

18b Political Requirements

- The game should adhere by all laws laid out by the Constitution of the United States.

- The product should be made available to download on a public website that should be made available through only Mozilla Firefox, Internet Explorer, Safari and Google Chrome web browsers.

- Only individuals over the age of ten should be allowed to play the Game

19 Legal Requirements

19a Compliance Requirements

- The product should not distribute user information to third-party sources without the consent of the user, thereby complying with the Data Protection Act.
- The game should avoid all copyright infringements.
- The game is available free online and should not be redistributed by independent individuals or companies seeking personal gains.
- The product should comply with stock market industry's rules and regulations

19b Standards Requirements

- The product should be developed following the agile development process.

IV Design

20 System Design

21 Current Software Architecture

There is no current software architecture. However, below is the proposed software architecture.

22 Proposed Software Architecture

22a Overview

The Rise of Civilizations follows the Model-View-Controller (MVC) architecture which can be explained as follows:

- **Model** is where the application's data objects are stored. The model doesn't know anything about views and controllers. In the game, the model layer will encapsulate the logic for gameplay. The model layer will be responsible for performing all the computations such as determining the outcome of an attack, adding/subtracting user's resources. The Model layer will also have the logic to interact with the database.

- **View** is what's presented to the users and how users interact with the app. In the game, view layer will be responsible for building the user interface. Everything that will be visible on screen will be part of the view layer. The view layer will also take care of the interaction of the user with UI such as entering his/her username, controlling the movement of villagers/scout cavalry/ army/ resources in order to perform activities such as building units attacking opponents.

- **Controller** is the decision maker and the glue between the model and view. The controller updates the view when the model changes. It also adds event listeners to the view and updates the model when the user manipulates the view. The controller will be responsible for delegating the calls between the model and the controller. The controller will load the necessary view based on the computations performed by the model. The controller will be responsible for calling the necessary model classed based on the conditions in the game such as calling the database classes in case of storage/retrieval of user information or the calling of the appropriate model classes in order to update the user resources in case the user acquires/loses any.

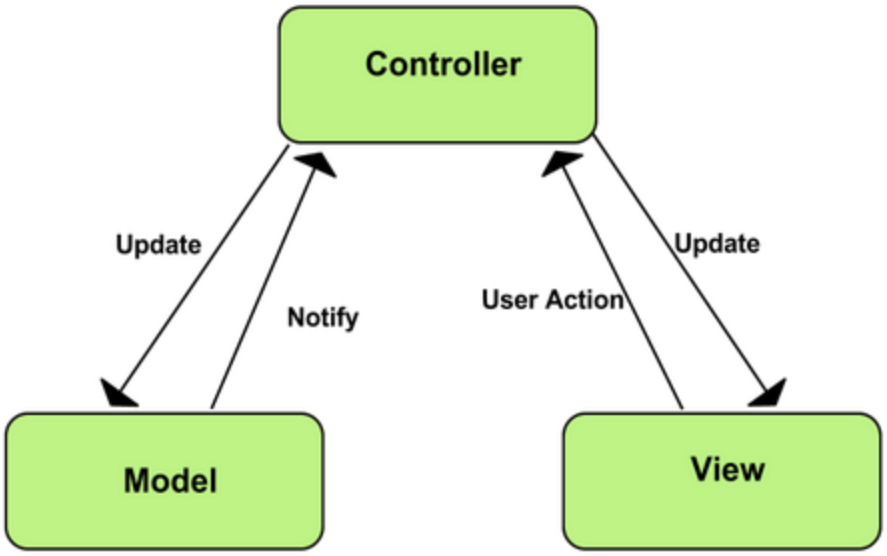


Figure 8.0: Figure depicting the software architecture implemented as Model-View-Controller architecture.

22b Class Diagrams

The class diagram below shows the preliminary relationships between the different classes. An object model is shown later with more detail, which includes data fields, data methods and properties of each class.

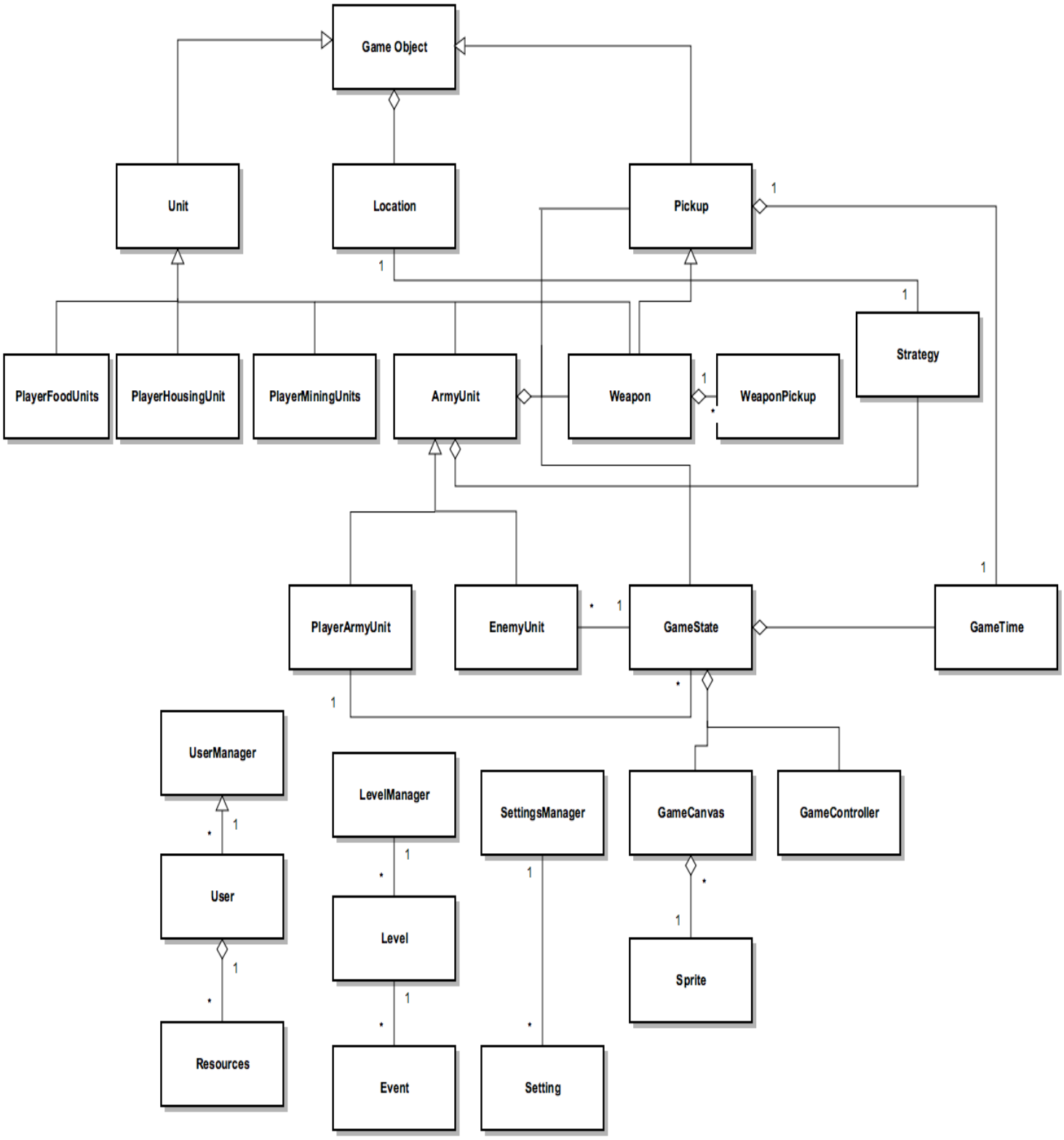
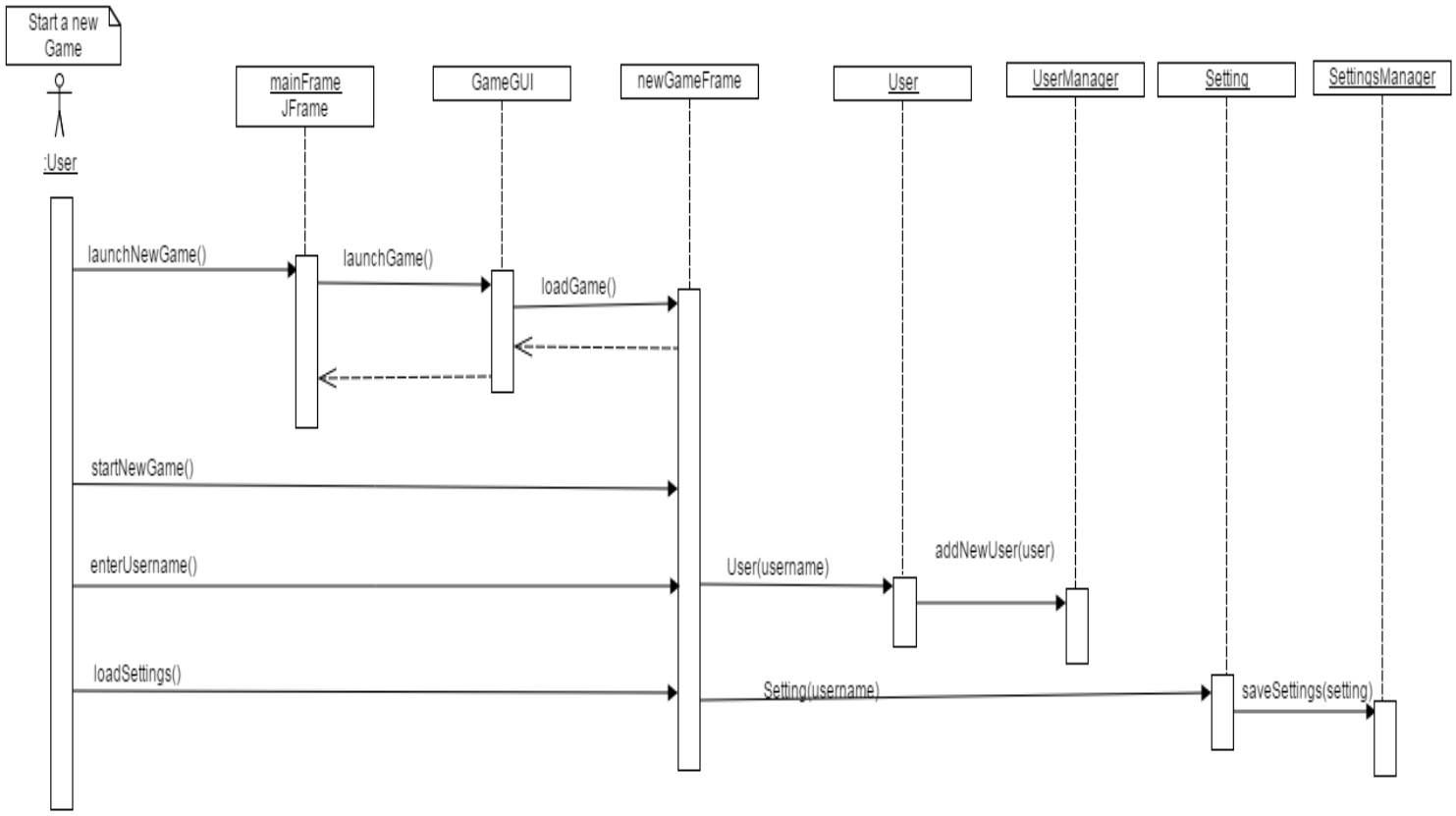


Figure 9.0: Class diagram for the game

22c Dynamic Model

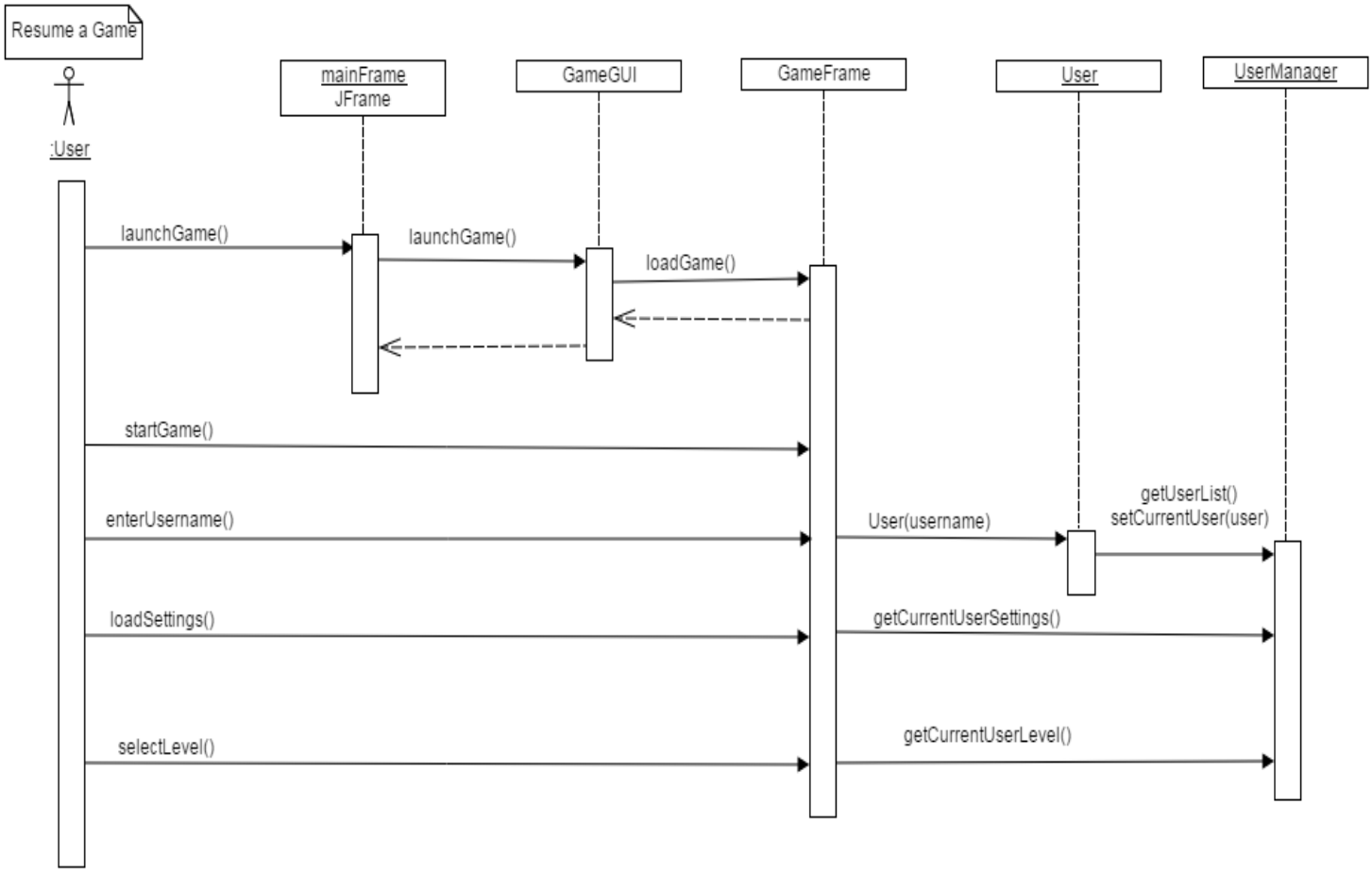
The sequence diagrams for the uses cases are diagramed below. Each sequence diagram corresponds to a use case discussed earlier.

Figure 10.0: Sequence diagram to allow the user to start a new game



9

Figure 11.0: Sequence Diagram to allow the user to resume a game



9

Figure 12.0: Figure to allow the user to build units

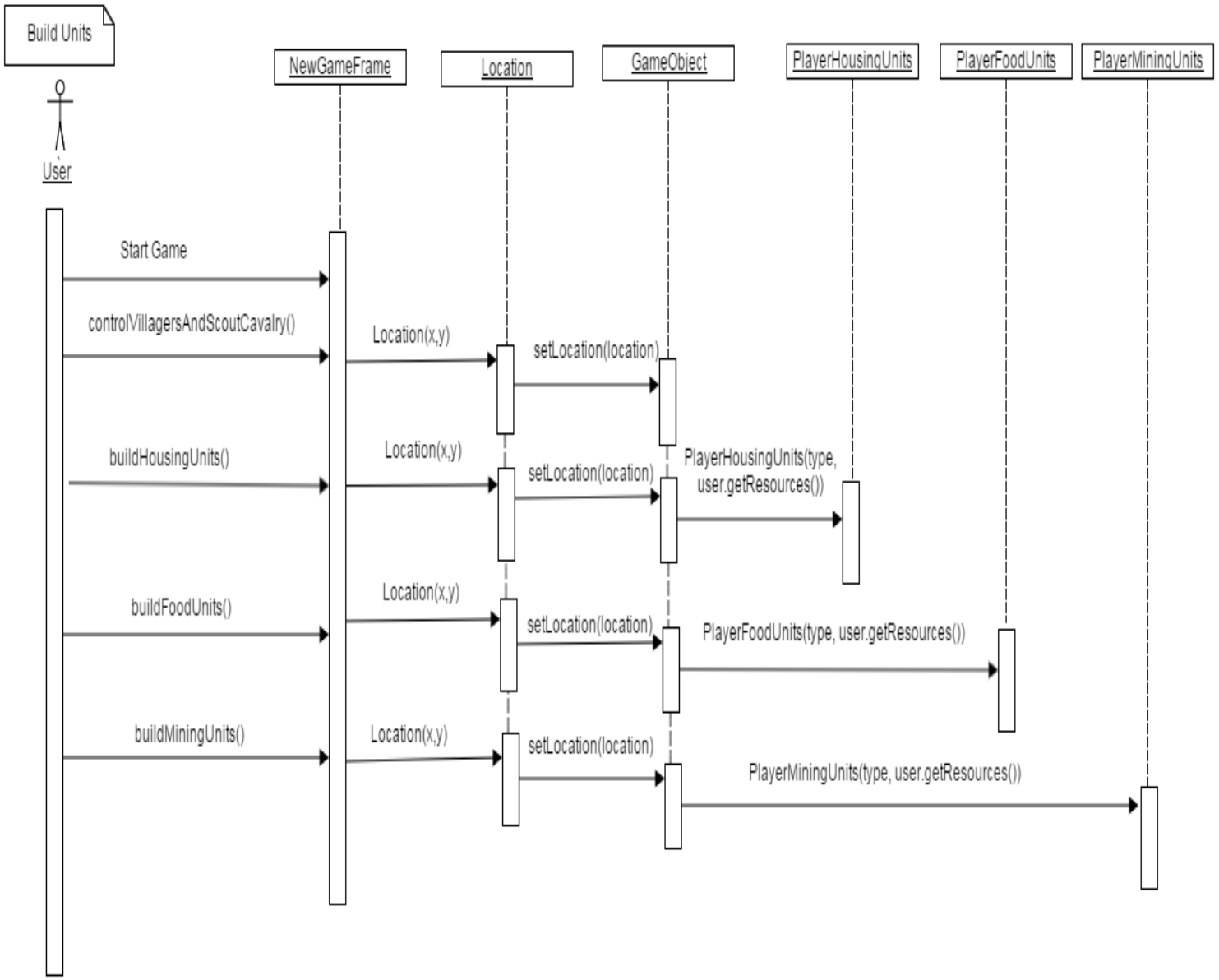
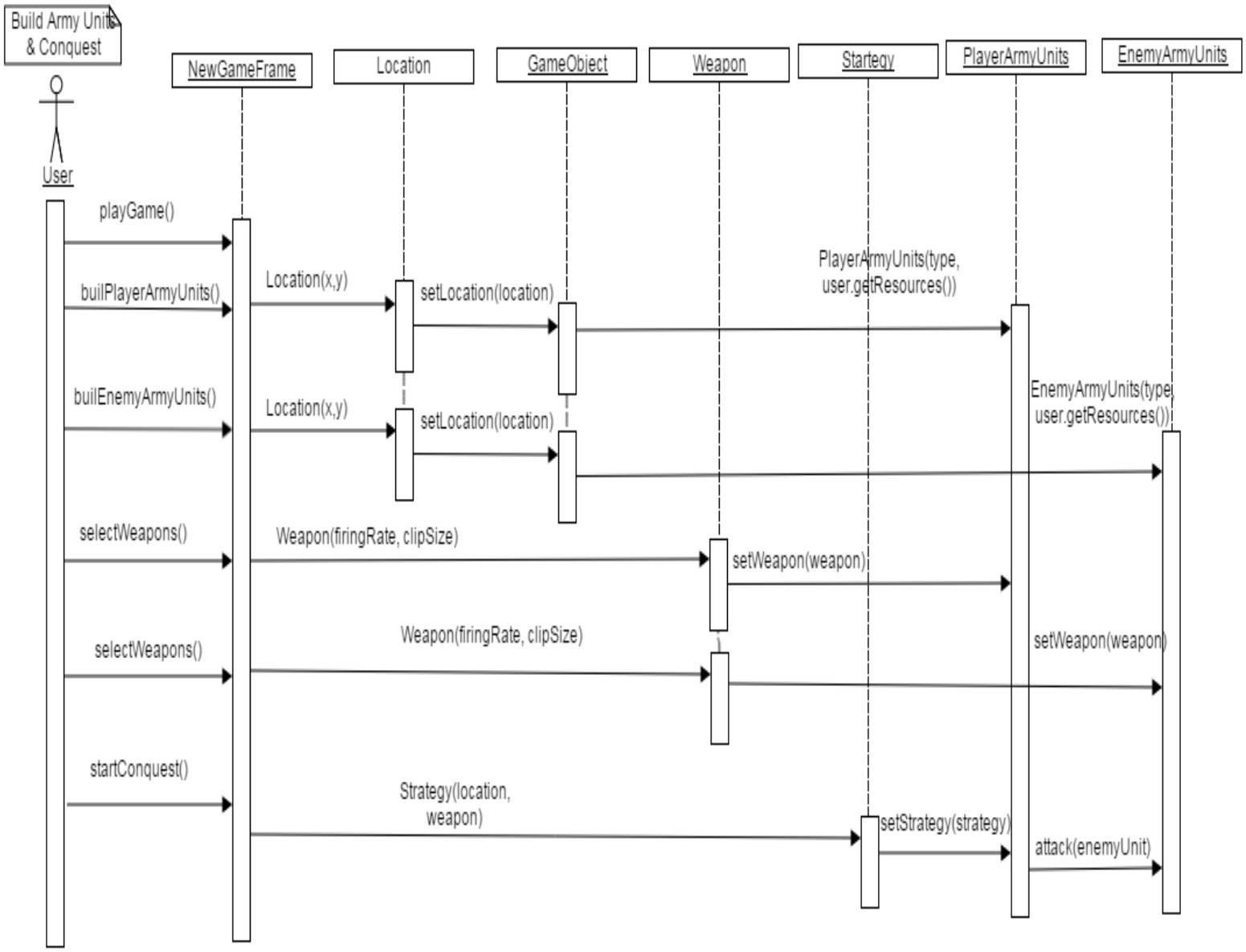


Figure 13.0: Sequence Diagram to allow the user to engage in conquest with opponent.



22d Subsystem Decomposition

The various subsystems of the game describe the features of the actual game play. This includes the features that are at the disposal of the users during the game. Each of the subsystem can be assigned to small group of programmers or an individual programmer. The game play subsystem is summarized with a component diagram below:

i. Game Configuration component

The component given below shows the *Game Configuration* component in terms of other subcomponents of the system. The component consists of the *UserManager, LevelManager* and *SettingsManager* sub components. The implementation of this component will constitute the login, user retrieval, level selection and the options selection functionalities of the game.

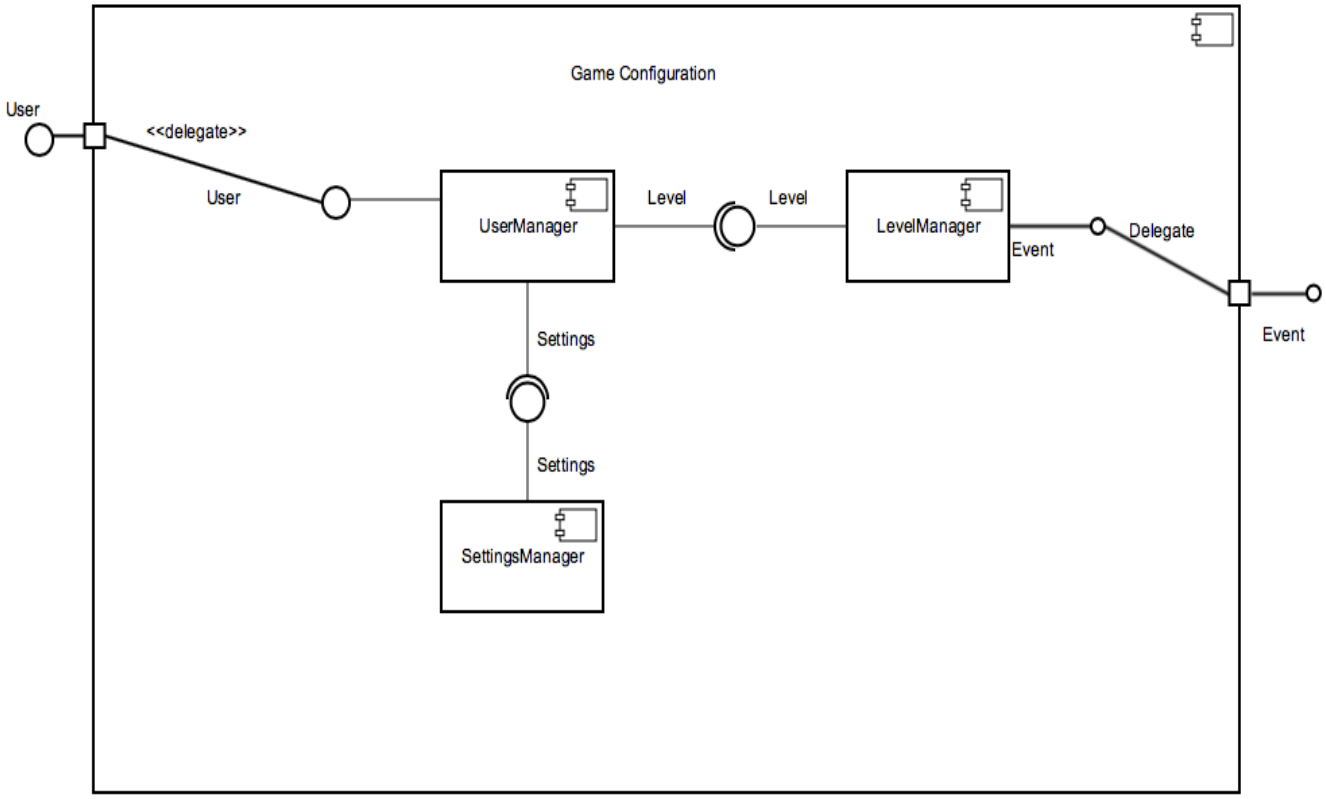


Figure 14.0: The Game Configuration component

ii. ***Build Units Component***

The component given below shows the *Build Units* component in terms of other subcomponents of the system. The component consists of the *PlayerUnits, GameResources, Location* sub components. The implementation of this component will constitute the Building the player units at specific (x. y) coordinates and using only the resources selected by the user.

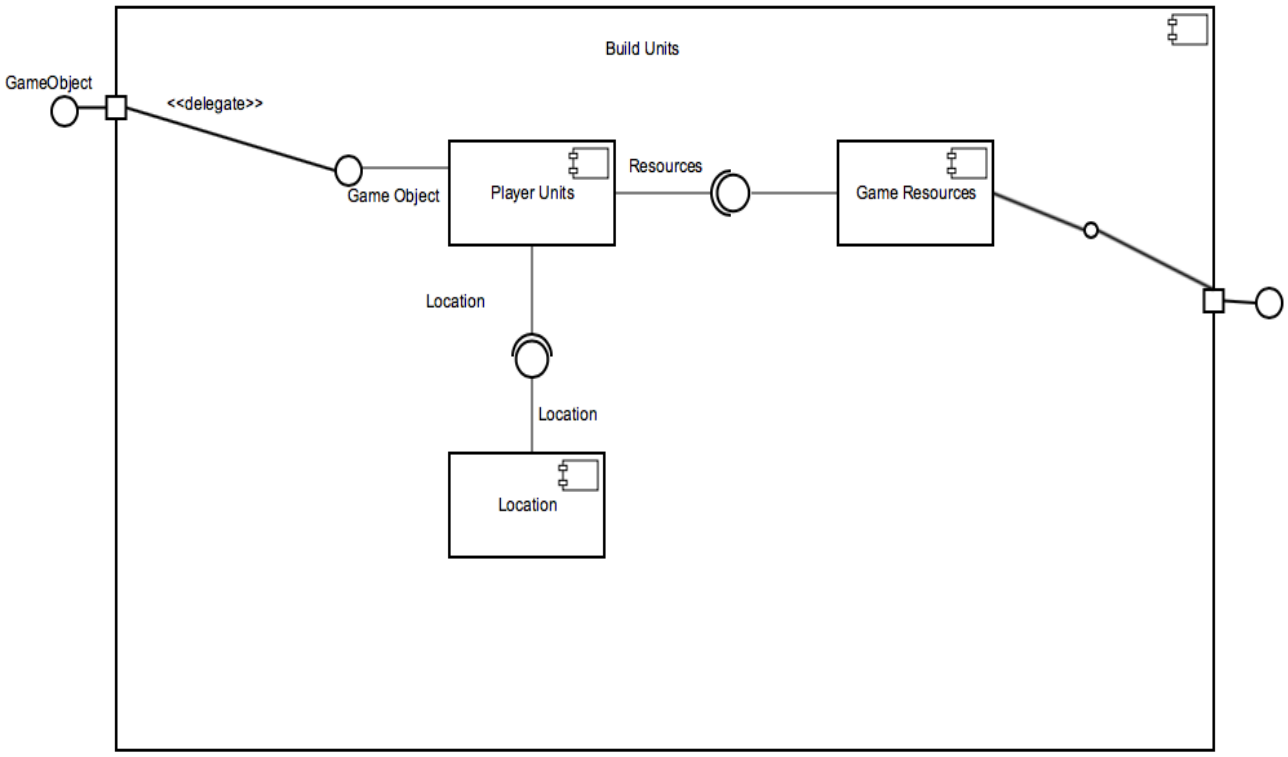


Figure 15.0: The Build Units Component

iii. ***Attack Opponent Component***

The component given below shows the *Attack Opponent* component in terms of other subcomponents of the system. The component consists of the *PlayerArmyUnit, EnemyArmyUnit* and *Strategy* sub components. The implementation of this component will constitute functionality wherein the user will able to engage in conquest with opponents deploying his army unit and the technology (strategy) for fighting the conquest.

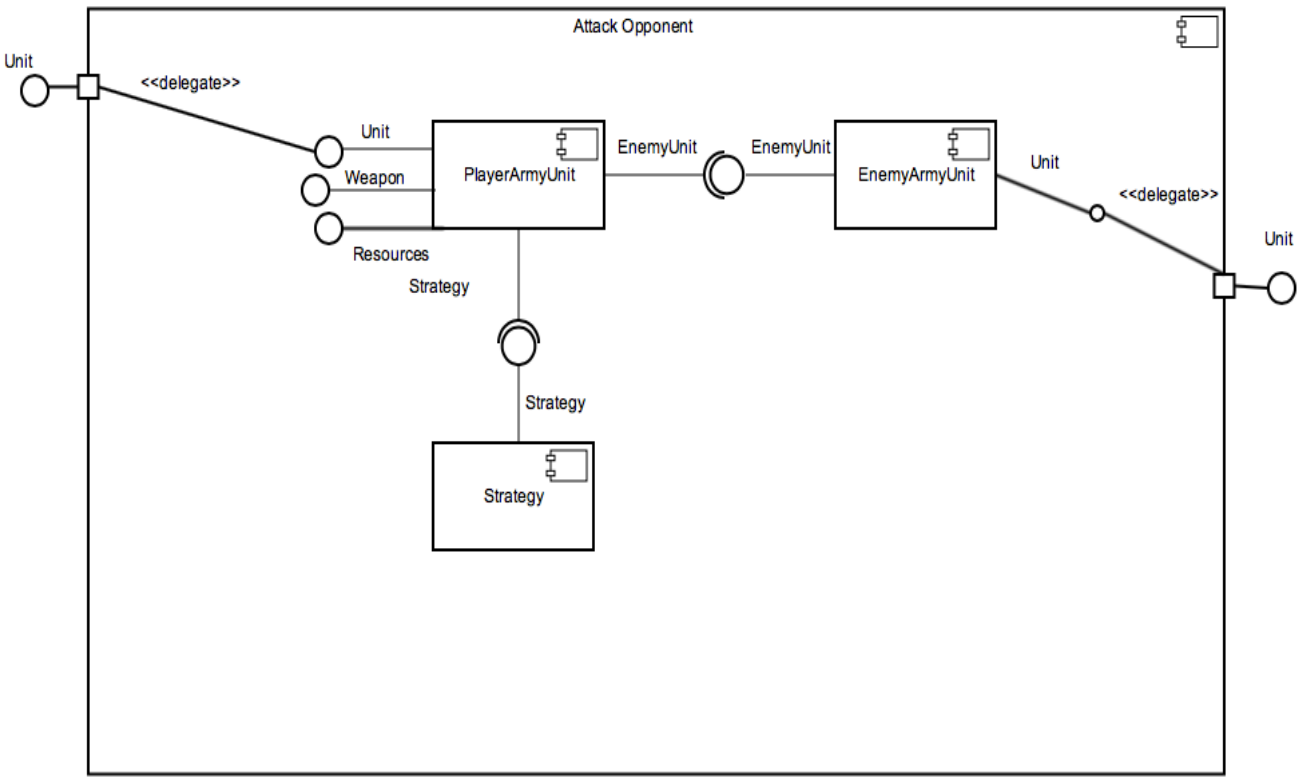


Figure 16.0: Attack Opponent component

iv. Game Load Component

The component given below shows the *Game Load* component in terms of other subcomponents of the system. The component consists of the *GameController*, *GameCanvas* and *Sprite* sub components. The implementation of this component will constitute the functioning of single screen of display on screen which constitutes calling the game canvas from the game controller along with sprite animations displayed on screen.

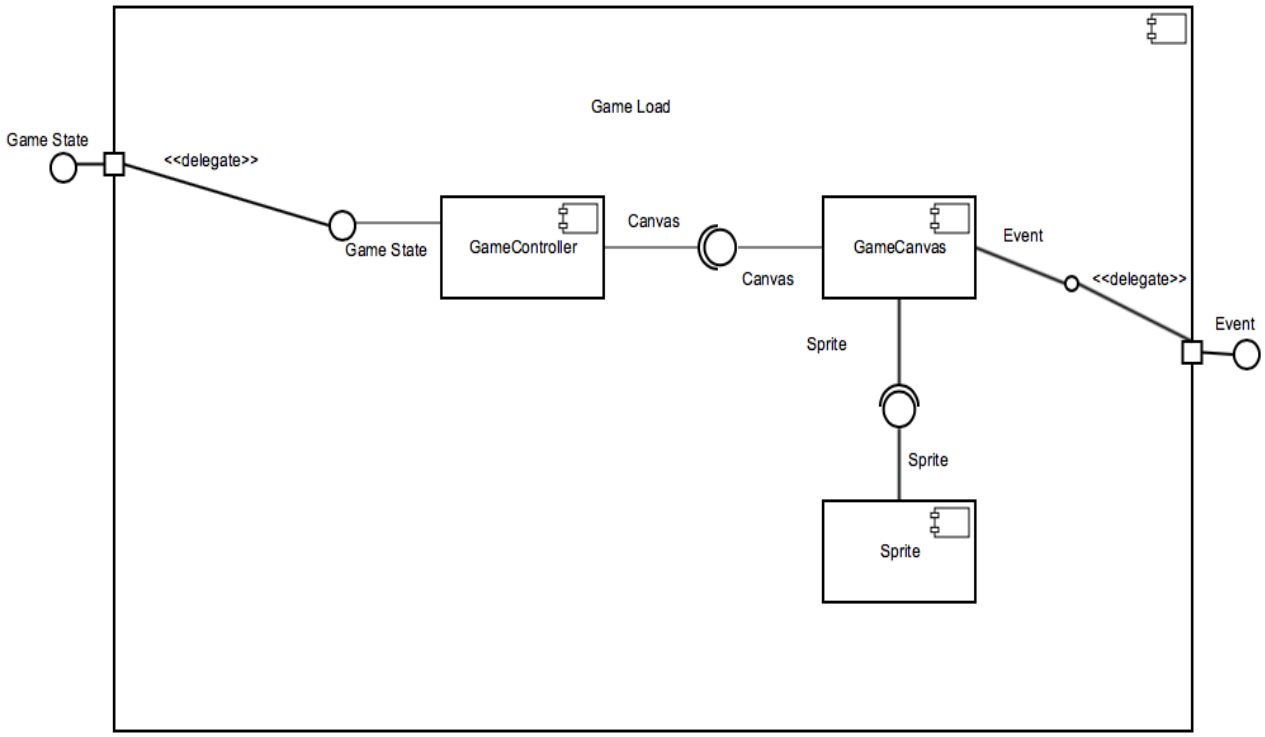


Figure 17.0: Game Load Component

22e Hardware/ Software Mapping

The *Rise of Civilizations* game will be implemented to be played as an executable over the network (LAN). The *Deployment Diagram* for the game, hence, consists of the following platforms: *File Server*, *User PC* and *Database Server*. The *Filer Server* can be any file server from where the game will read some of the initial game configurations required for its initialization. The *User PC*, as the name suggests, is the computer on which the user launches his/her game. The *Database Server* will be used to store the user's progress in the game such as the score, last level played, the resources owned by him/her and other game specific information that needs to be saved to a persistent storage. The diagram also shows the mapping between the hardware and software components of the system depicting which software component(s) will run on which platform/ hardware component.

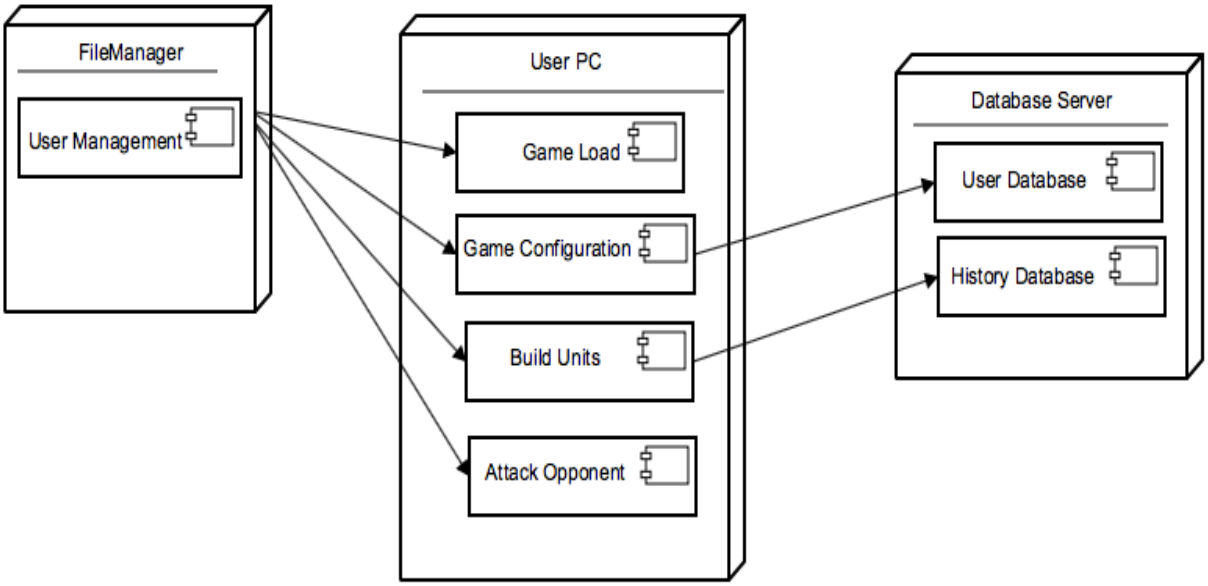


Figure 18.0: Deployment Diagram for the game.

22f Data Dictionary

Start Game:

The user has successfully logged into the game after setting all the necessary game configurations.

Build Unit:

The user can successfully create build units in the game.

Build Houses:

The user can successfully create housing units in the game.

Build Resource Stacking Units:

The user can successfully build resource stacking units such as lumber camps, town center(s) and docking stations in the game.

Build Military Units:

The user can successfully build military units such as barracks, archery ranges and also associate armed men, archers with them.

Build Mining Units:

The user can successfully build mining units such as gold or stone mining units.

Conquests:

The user deploys his/her military resources to attack some other player.

Forge Alliance:

The user is able to forge an alliance with other players.

Balance:

Shows the current amount of virtual currency on user's account.

Trade:

Exchange of goods for virtual currency.

Buy Goods:

Users can buy any number of goods which they are interested in and the virtual currency will deducted from the virtual account.

Profile:

Contains the information about the user's virtual account, their past trades, access to leaderboard.

Sell Goods:

In portfolio users can sell his/her owned company shares and the money is added to the virtual bank account.

Virtual Account:

User can add extra money to their account by purchasing virtual currency via online credit card transaction from bank account screen.

22g Persistent data management

The persistent data includes the login information of the users that is entered into the database once the user signs up for the game. This persistent data is stored into the database. The database that will be used is a relational database. The database will be kept at a remote location and will only be accessible to game administrators.

Another persistent data that exists for this game is the Historical Facts database. This is also stored in a relational database to keep track of all of the events that have happened in history. This is also kept at a remote location; however, this database is also backed up at that location in case the other database is down at any time. If that is the case, then the backed up database can be used.

22h Access Control and Security

Access Control Matrix:

Objects **Actors**	History Database	Player database	Settings	Resources	Leaderboard
Player	viewOnly()	viewOnly()	view() and update()	view() and update()	viewOnly()
Opponent	viewOnly()	viewOnly()	view() and update()	view() and update()	viewOnly()
Game Manager	viewOnly()	viewOnly()	view() and update()	view()	view()
Database Administrator	view() and update()	view() and update()	view() and update()	view()	view()

22i Global Software Control

The control flow that will be selected for The Rise of Civilizations will be the event-driven control flow paradigm. Once the user requests some operation by clicking on a button, an event will become available and it will be dispatched to an appropriate object, based on the information associated with the event. This can include clicking on the "Build" button. Once the user clicks on the "Build Unit" button, an event will be fired and an appropriate object will be dispatched which will result in a creation of a new Unit on the game screen. This will be the case for almost all of the features of the game. Once the user clicks or presses something, an appropriate event will be fired, resulting in a new window appearing or a new action taking place.

22j Boundary Conditions

The boundary conditions of the system specifies how the system is started, initialized, and shut down and we need to define how we deal with major failures such as data corruption and network outages.

Startup and Shutdown: Each game instance is initialized and started as soon as the player connects to the game server on the social networking website using the game icon on local machine. The player can terminate that game instance by choosing the Exit option from the menu on the game window or closing the session forcibly by clicking the cross button on the top right corner of the window.

Exception Handling: The Rise of Civilizations game is somewhat network-dependent. Hence, we have to take care of network outages. So, in case of any such failure, the system should be able to save the state of the game for a particular player after every move. Later on when the network resumes, the player should be able to restore to its previous state of game from where he left. There are exceptions handling mechanisms on java to handle such unexpectedly closed sockets. We need to handle these exceptions carefully, since our game is network dependent for most of its data resources. Subsystem responsible for database management should be able to detect whether or not it was properly shut down and should be able to perform consistency checks and repair corrupted data, as necessary.

23 Subsystem Services

- *UserManager:* The *UserManager* should keep track of all the users that may have logged into the game from the current installation of the game. The *UserManager* will be responsible for enabling a new user to successfully log into the game and to enable a returning user to continue his/her game from a previous checkpoint.

- *LevelManager:* The *LevelManager* should keep track of all the levels available in the game. It should also contain a mapping for all the returning users with their respective levels of gameplay. The *LevelManager* should also know the next level to load at any point of user's gameplay.

- *SettingsManager:* The *SettingsManager* should keep track of all the user settings. The user settings may include the map style, screen resolution, sound level, key combinations. The *SettingsManager* should keep track of the settings corresponding to different users of the system so that the game can successfully load the game settings for all the returning users of the game.

- *GameController:* The *GameController* class has to keep track of the flow of the game. The *GameController* class should know which user action should call which computation and that computation should display what on screen. It should also act as a dispatcher between the model and the view components of the game.

- *GameState:* The *GameState* should keep track of all the data present in the game at any instant such as the player units, the player's army strength, the technologies owned by him. The *GameState* is also responsible for having the logic for the game encapsulated within such that for any game scenario, it is able map every user action/move to gameplay progress.

- *GameResources:* The *GameResources* should keep track of all the resources present in the game at any instant. The resources consist of stone, gold, wood, housing units, mining units, food stacking units, army units, in other words, any such entity in the game that determines the user's worth in the game.

- *GameUnits:* The *GameUnits* should keep track of all types of units such as food stacking units, mining units, docking units, army units, their location in the game and the amount of resources that unit is worth.

- *Strategy:* The *Strategy* come into play when the players engage in conquests. The *Startegy* keeps track of the technologies and the arms/weapons owned by the army units of each of the players engaged in a conquest in the game.

24 User Interface

Below is the preliminary design for the user interface.

Figure 19.0: The image shows the home screen for the game, contains the "Learn to play", "History", "Game Configuration" menus.

Figure 20.0: The image shows the first screen that the user encounters after starting a new game, shows the available resources in bottom left corner, map in the bottom right corner and the player name in the top-right corner.

Figure 21.0: The image shows the various units such as "Housing Units", "Renewable Food Units", and "Mining Units" that may be built by the user during his/her gameplay.

Figure 22.0: The image shows the scenario wherein the players engage in conquest. The menu at the bottom shows the technologies the player can deploy during the conquest along with the armed men available for conquest.

Image source: https://www.youtube.com/watch?v=mMey4_itA&list=PLEQWWx1EYwpqmXRFnWeBLh77n4tzujrs2&index=1

25 Oject Design

25a Object Design trade-offs

- *Efficiency vs Reusability*: Reusability is not the main concern for designing our system because we do not plan to integrate any of our classes to a different game or any other similar systems. Therefore, the classes are designed specifically for the tasks of our game so the code is not made more complex than necessary. This design approach fortified our most important design goal, which is efficiency, since there will not be any fancy "if statements" or type checking to enforce the reusability constraint.

- *Functionality vs Usability*: It is very important to have wide range of customers. Therefore, the game will have plain usage. The system should not be too complex to play. It means that the functionality of the system will be basic. Since the purpose of the system is entertaining the users, we focused on the usability of the system rather than making it functional more than necessary. The game has a simple interface and familiar instructions to play instead of complex menus and various features. Thus, the users can spend time enjoying the game rather than struggling to learn it.

- *Space vs Speed*: Space is notably inexpensive and as such is more expendable than speed. It is more important that the site load fast than that we take up less disk space. With that said, images and other date should be stored in formats that allow for the quickest possible processing regardless of size.

25b Interface Documentation Guidelines

The following *Interface Documentation Guidelines* should be followed for the *Rise of Civilizations* game:

- Each Java source file contains a single public class or interface. When private classes and interfaces are associated with a public class, you can put them in the same source file as the public class. The public class should be the first class or interface in the file.
- All source files should begin with a c-style comment that lists the class name, version information, date, and copyright notice.
- The first non-comment line of most Java source files is a package statement. After that, import statements can follow.
- The declaration of class as well instance variables should follow the following order:

First the public class variables, then the protected, then package level (no access modifier), and then the private.

- Methods should be grouped by functionality rather than by scope or accessibility. For example, a private class method can be in between two public instance methods. The goal is to make reading and understanding the code easier.
- Avoid lines longer than 80 characters, since they're not handled well by many terminals and tools. When an expression will not fit on a single line, break it according to these general principles:
    - Break after a comma.
    - Break before an operator.
    - Prefer higher-level breaks to lower-level breaks.
    - Align the new line with the beginning of the expression at the same level on the previous line.
    - If the above rules lead to confusing code or to code that's squished up against the right margin, just indent 8 spaces instead.

- Block comments are used to provide descriptions of files, methods, data structures and algorithms. Block comments may be used at the beginning of each file and before each method. They can also be used in other places, such as within methods. Block comments inside a function or method should be indented to the same level as the code they describe.
- Short comments can appear on a single line indented to the level of the code that follows. If a comment can't be written in a single line, it should follow the block comment format. A single-line comment should be preceded by a blank line.
- Very short comments can appear on the same line as the code they describe, but should be shifted far enough to separate them from the statements. If more than one short comment appears in a chunk of code, they should all be indented to the same tab setting.
- The // comment delimiter can comment out a complete line or only a partial line. It shouldn't be used on consecutive multiple lines for text comments; however, it can be used in consecutive multiple lines for commenting out sections of code
- Doc comments describe Java classes, interfaces, constructors, methods, and fields. Each doc comment is set inside the comment delimiters /**...*/, with one comment per class, interface, or member
- One declaration per line is recommended since it encourages commenting
- Try to initialize local variables where they're declared. The only reason not to initialize a variable where it's declared is if the initial value depends on some computation occurring first.
- Put declarations only at the beginning of blocks. (A block is any code surrounded by curly braces "{" and "}".) Don't wait to declare variables

until their first use; it can confuse the unwary programmer and hamper code portability within the scope.

- When coding Java classes and interfaces, the following formatting rules should be followed:
  - No space between a method name and the parenthesis "(" starting its parameter list.
  - Open brace "{" appears at the end of the same line as the declaration statement.
  - Closing brace "}" starts a line by itself indented to match its corresponding opening statement, except when it is a null statement the "}" should appear immediately after the "{".
- Each line should contain at most one statement.
- Compound statements are statements that contain lists of statements enclosed in braces "{ statements }".
- The enclosed statements should be indented one more level than the compound statement.
- The opening brace should be at the end of the line that begins the compound statement; the closing brace should begin a line and be indented to the beginning of the compound statement.
- Braces are used around all statements, even single statements, when they are part of a control structure, such as an if-else or for statement. This makes it easier to add statements without accidentally introducing bugs due to forgetting to add braces.
- A return statement with a value should not use parentheses unless they make the return value more obvious in some way
- The if-else class of statements should have the following form:

```
if (condition) {
   statements;
}

if (condition) {
   statements;
} else {
   statements;
}

if (condition) {
   statements;
} else if (condition) {
   statements;
} else {
   statements;
}
```

- A for statement should have the following form:

for (*initialization*; *condition*; *update*) {
   statements;
}

- An empty for statement (one in which all the work is done in the initialization, condition, and update clauses) should have the following form:
  for (*initialization*; *condition*; *update*);
- A while statement should have the following form:

  while (*condition*) {
     statements;
  }

- An empty while statement should have the following form:
  while (*condition*);
- A do-while statement should have the following form:
  do {
     statements;
  } while (*condition*);
- A switch statement should have the following form:
  switch (*condition*) {
  case ABC:
     statements;
     /* falls through */
  case DEF:
     statements;
     break;
  case XYZ:
     statements;
     break;
  default:
     statements;
     break;
  }
- Every switch statement should include a default case. The break in the default case is redundant, but it prevents a fall-through error if later another case is added.
- A try-catch statement should have the following format:

  try {
     statements;
  } catch (ExceptionClass e) {
      statements;
  }

- A try-catch statement may also be followed by finally, which executes regardless of whether or not the try block has completed successfully.

```
try {
   statements;
} catch (ExceptionClass e) {
   statements;
} finally {
   statements;
}
```

- Blank lines improve readability by setting off sections of code that are logically related
- Two blank lines should always be used in the following circumstances:
  - Between class and interface definitions.
- One blank line should always be used in the following circumstances:
  - Between methods.
  - Between the local variables in a method and its first statement.
  - Before a block.
  - Between logical sections inside a method to improve readability.
- Blank spaces should be used in the following circumstances:
  - A keyword followed by a parenthesis should be separated by a space
  - A blank should appear after commas in argument lists.
  - All binary operators except . should be separated from their operands by spaces. Blank spaces should never separate unary operators such as unary minus, increment ("++"), and decrement ("--") from their operands.
  - The expressions in a for statement should be separated by blank spaces.
  - Casts should be followed by a blank space
- Don't make any instance or class variable public without good reason. Often, instance variables don't need to be explicitly set or gotten-often that happens as a side effect of method calls.
- Avoid using an object to access a class (static) variable or method. Use a class name instead.
- Numerical constants (literals) should not be coded directly, except for -1, 0, and 1, which can appear in a for loop as counter values.
- Avoid assigning several variables to the same value in a single statement. It is hard to read.
- It is generally a good idea to use parentheses liberally in expressions involving mixed operators to avoid operator precedence problems. Even if the operator precedence seems clear to you, it might not be to others-you shouldn't assume that other programmers know precedence as well as you do

- If an expression containing a binary operator appears before the ? in the ternary ?: operator, it should be parenthesized
- Use XXX in a comment to flag something that is bogus but works. Use FIXME to flag something that is bogus and broken.

- ***Naming Conventions***

Identifier Type	Rules for Naming	Examples
Packages	The prefix of a unique package name is always written in all-lowercase ASCII letters and should be one of the top-level domain names, currently com, edu, gov, mil, net, org, or one of the English two-letter codes identifying countries as specified in ISO Standard 3166, 1981. Subsequent components of the package name vary according to an organization's own internal naming conventions. Such conventions might specify that certain directory name components be division, department, project, machine, or login names.	com.sun.eng com.apple.quicktime.v2 edu.cmu.cs.bovik.cheese
Classes	Class names should be nouns, in mixed case with the first letter of each internal word capitalized. Try to keep your class names simple and descriptive. Use whole words-avoid acronyms and abbreviations (unless the abbreviation is much more widely used than the long form, such as URL or HTML).	class Raster; class ImageSprite;
Interfaces	Interface names should be capitalized like class names.	interface RasterDelegate; interface Storing;
Methods	Methods should be verbs, in mixed case with the first letter lowercase, with the first letter of	run(); runFast();

	each internal word capitalized.	getBackground();
Variables	Except for variables, all instance, class, and class constants are in mixed case with a lowercase first letter. Internal words start with capital letters. Variable names should not start with underscore _ or dollar sign $ characters, even though both are allowed. Variable names should be short yet meaningful. The choice of a variable name should be mnemonic- that is, designed to indicate to the casual observer the intent of its use. One-character variable names should be avoided except for temporary "throwaway" variables. Common names for temporary variables are i, j, k, m, and n for integers; c, d, and e for characters.	int i; char c; float myWidth;

Constants	The names of variables declared class constants and of ANSI constants should be all uppercase with words separated by underscores ("_"). (ANSI constants should be avoided, for ease of debugging.)	static final int MIN_WIDTH = 4;
		static final int MAX_WIDTH = 999;
		static final int GET_THE_CPU = 1;

25c Class Interfaces

- *Class GameObject:* This class represents the UI object, represents everything that is visible on screen. This is the superclass of almost all the classes. This class contains the (x, y) coordinates of the UI object as well as the current user information.

- *Class Unit:* This class is the represents the units that can be built by the user during gameplay. This will have the information of the resources currently available to the user as resources will be utilized during the creation of units. The classes such as *PlayerHousingUnits, PlayerMiningUnits* and *PlayerFoodUnits* extend from this class.

- *Class ArmyUnit:* This class represents the army units built by the players in order to increase their worth in the game. Both *PlayerArmyUnits* and *EnemyArmyUnits* extend from this class. The class has the strategy

information associated with it which represents the technologies the players want to deploy in the conquest. The player will also be able to set the *Weapon* information associates with it to its army unit which will obviously be utilized in case of a conquest.

- *Class Resource:* This class represents all the resources such as wood, stone, villagers associates with a player during the gameplay.

- *Class User:* This class represents the player of the game and has his basic information such as *username, currentLevel* and the *resources* held by him in the game.

- *Class UserManager:* This is a factory class and contains tha information about all the users that may have logged into the current installation of the game along with their information such as their current level of play, the game configurations corresponding to a particular player.

- *Class Setting/ SettingsManager:* The *Setting* class represents the settings of the user corresponding to a gameplay. The class had information about the user settings such as *sound, music, upKey, downKey* and the other keys and the actions associated with them. The *SettingsManager* class will store the kist of all the settings for the user.

- *Class Level/LevelManager:* The *Level* class represents the level information such as the *levelId, levelName* associated with it. The *LevelManager* class contains a list of all the levels in the game.

- *Class GameState:* This game is the *model* class for the game and contains the information about all the players, opponents and elapsed time. This class contains all the data and dictates what is to be displayed and presented to the user.

- *Class GameController:* As the name suggests, this is the class that acts as a glue between the view(*GameObject*) and model(*GameState)* classes and dictates when the model class is to be called and based on the computations of the model class, which view is to be rendered on screen.

- *Class GameCanvas:* This is the basic UI class that is essential to every gaming application. This represents a blank canvas on which the UI is rendred.

- *Class GameTime:* This is the timer class and represents the *timeElapsed* in the game.

- *Class Sprite:* This is the basic animation class. A sprite represents a computer graphic that may be moved on-screen and otherwise manipulated as a single entity.

25d Object Model

The classes described in the last section can be used to describe the object model of the game with the help of the following class diagram:

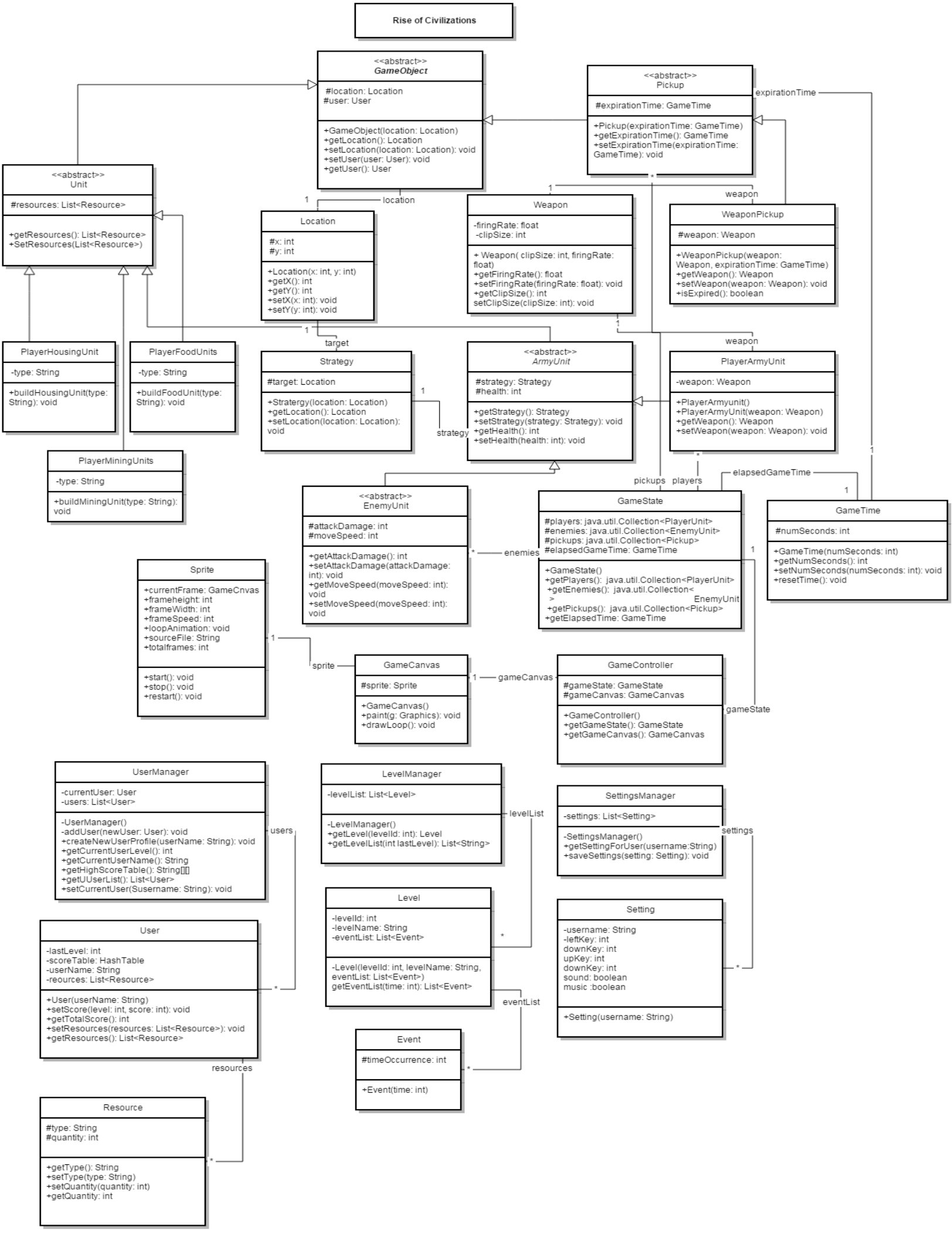

Rise of Civilizations
<<abstract>>
GameObject
#location: Location
#user: User
+GameObject(location: Location)
+getLocation(): Location
+setLocation(location: Location): void
+setUser(user: User): void
+getUser(): User
<<abstract>>
Pickup
#expirationTime: GameTime
+Pickup(expirationTime: GameTime)
+getExpirationTime(): GameTime
+setExpirationTime(expirationTime: GameTime): void
expirationTime
<<abstract>>
Unit
#resources: List<Resource>
+getResources(): List<Resource>
+SetResources(List<Resource>)
location
Location
#x: int
#y: int
+Location(x: int, y: int)
+getX(): int
+getY(): int
+setX(x: int): void
+setY(y: int): void
Weapon
-firingRate: float
-clipSize: int
+ Weapon(clipSize: int, firingRate: float)
+getFiringRate(): float
+setFiringRate(firingRate: float): void
+getClipSize(): int
setClipSize(clipSize: int): void
weapon
WeaponPickup
#weapon: Weapon
+WeaponPickup(weapon: Weapon, expirationTime: GameTime)
+getWeapon(): Weapon
+setWeapon(weapon: Weapon): void
+isExpired(): boolean
PlayerHousingUnit
-type: String
+buildHousingUnit(type: String): void
PlayerFoodUnits
-type: String
+buildFoodUnit(type: String): void
PlayerMiningUnits
-type: String
+buildMiningUnit(type: String): void
target
Strategy
#target: Location
+Stratergy(location: Location)
+getLocation(): Location
+setLocation(location: Location): void
strategy
<<abstract>>
ArmyUnit
#strategy: Strategy
#health: int
+getStrategy(): Strategy
+setStrategy(strategy: Strategy): void
+getHealth(): int
+setHealth(health: int): void
weapon
PlayerArmyUnit
-weapon: Weapon
+PlayerArmyunit()
+PlayerArmyUnit(weapon: Weapon)
+getWeapon(): Weapon
+setWeapon(weapon: Weapon): void
<<abstract>>
EnemyUnit
#attackDamage: int
#moveSpeed: int
+getAttackDamage(): int
+setAttackDamage(attackDamage: int): void
+getMoveSpeed(moveSpeed: int): void
+setMoveSpeed(moveSpeed: int): void
enemies
pickups
players
GameState
#players: java.util.Collection<PlayerUnit>
#enemies: java.util.Collection<EnemyUnit>
#pickups: java.util.Collection<Pickup>
#elapsedGameTime: GameTime
+GameState()
+getPlayers(): java.util.Collection<PlayerUnit>
+getEnemies(): java.util.Collection<EnemyUnit>
+getPickups(): java.util.Collection<Pickup>
+getElapsedTime: GameTime
elapsedGameTime
GameTime
#numSeconds: int
+GameTime(numSeconds: int)
+getNumSeconds(): int
+setNumSeconds(numSeconds: int): void
+resetTime(): void
Sprite
+currentFrame: GameCnvas
+frameheight: int
+frameWidth: int
+frameSpeed: int
+loopAnimation: void
+sourceFile: String
+totalframes: int
+start(): void
+stop(): void
+restart(): void
sprite
GameCanvas
#sprite: Sprite
+GameCanvas()
+paint(g: Graphics): void
+drawLoop(): void
gameCanvas
GameController
#gameState: GameState
#gameCanvas: GameCanvas
+GameController()
+getGameState(): GameState
+getGameCanvas(): GameCanvas
gameState
UserManager
-currentUser: User
-users: List<User>
-UserManager()
-addUser(newUser: User): void
+createNewUserProfile(userName: String): void
+getCurrentUserLevel(): int
+getCurrentUserName(): String
+getHighScoreTable(): String[][]
+getUUserList(): List<User>
+setCurrentUser(Susername: String): void
users
LevelManager
-levelList: List<Level>
-LevelManager()
+getLevel(levelId: int): Level
+getLevelList(int lastLevel): List<String>
levelList
SettingsManager
-settings: List<Setting>
-SettingsManager()
+getSettingForUser(username:String)
+saveSettings(setting: Setting): void
settings
Level
-levelId: int
-levelName: String
-eventList: List<Event>
-Level(levelId: int, levelName: String, eventList: List<Event>)
getEventList(time: int): List<Event>
eventList
Setting
-username: String
-leftKey: int
downKey: int
upKey: int
downKey: int
sound: boolean
music :boolean
+Setting(username: String)
User
-lastLevel: int
-scoreTable: HashTable
-userName: String
-reources: List<Resource>
+User(userName: String)
+setScore(level: int, score: int): void
+getTotalScore(): int
+setResources(resources: List<Resource>): void
+getResources(): List<Resource>
resources
Event
#timeOccurrence: int
+Event(time: int)
Resource
#type: String
#quantity: int
+getType(): String
+setType(type: String)
+setQuantity(quantity: int)
+getQuantity: int

Figure 23.0: Class diagram for the game.

IV Test Plans

26 Features to be tested / not to be tested

Rise of Civilization shall be thoroughly tested to ensure that it operates with minimal glitches and maintains high standards. The following is a list of general features that will be fully tested before the final release. More detailed test plans can be found in the Test Cases section.

Game Login

Accessing History menu

Accessing Learn to play menu

Accessing Options Menu

Set game Configuration

Start a Single Player Game

Start a Multiplayer Game

Start a new Game

Resume a saved Game

Start Game

Control Scout Cavalry

Build houses

Build Renewable food production units

Build Resource Stacking Units

Build Military Units

Build Mining Units

Players engage in conquests

Player wins the conquest

Players engage in trade

Ring Town Bell

Pause Game

Build trading marketplaces and engage in trade.

Players Forge Alliance

End Game

History Database

User Database

Features not to be tested are those that reside outside of our control, namely to do with the Rise of Civilization Database. We cannot test the accuracy of the information contained on the file or the reliability of servers storing this information.

27 Pass/Fail Criteria

A test will be considered a pass if and only if the actual results of the test match the expected results specified in the associated test case. Any deviation from the expected results will be considered a failure of that test. A feature or method will pass when it has been subjected to all associated test cases and passed those at least 99% of the time when run on various inputs. Anything less will call for debugging of the associated code.

28 Approach

The testing approach for this project will be efficient. Testing will occur in tandem with Development during every phase of the project. All Developers are expected to perform white box path testing and unit testing, and generate associated reports, for any code they submit to the project. A fellow developer must then inspect submitted code.

Once all the code associated with a certain feature has gone through these first steps, integration testing may begin for that feature. Then the feature will be black box tested by the associated test cases below. As features are completed, they will then be run through integration testing to ensure they work together.

At various points the User Interface will be subjected to usability testing using the Wizard of Oz strategy. Where the ‘Wizard’ plays the part of the Civilization file. This is to ensure the interface is user friendly and easy to understand.

29 Suspension and Resumption

If at any point in the schedule above, a piece of code or feature fails to pass a test according to the pass criteria it shall be brought back to the developers to attempt to fix.

Developers will be given reports of what tests were not passed so that they may have an idea of what to look for. The code must go back through the steps again until it passes.

30 Testing materials (hardware / software requirements)

As this is a desktop game, it must be compatible with the most popular operating systems, Windows, Macintosh OS X, and Linux. The software should be tested on a machine running each of these operating systems. To properly test the interaction with the Civilization file, testing will need the .exe game file correctly installed.

31 Test cases

Test Case #	Test Name	Description	Pre-Condition	Step Name	Flow of Events	Post Condition
TC_1	**Select_SinglePlayer**	The user should be able to choose to play against the computer	The user should be on the home screen	Step 1	Click on the Single Player option on the home screen	A pop up will open with the following options: a)Standard Game b)Saved and Recorded Games
TC_2	**Select_MultiPlayer**	The user should be able to choose to play against players on LAN	The user should be on the home screen	Step 1	Click on the Multi Player option on the home screen	A new window should open which displays the list of players who are online
TC_3	**Set_GamePlaySettings**	The user should be able to fill out the player name and customize the gameplay settings such as setting the keyboard keys, setting the game volume level or to alter the mouse sensitivity by selecting the "Options" menu	The user should be on the home screen	Step 1	Click on the Options button	Settings page should open
				Step 2	Fill out player's name in the player name text box	Player's name should appear
				Step 3	Set the music volume by adjusting the vertical scroller	music volume should be set
				Step 4	Set the mouse sensitivity by adjusting the vertical scroller	mouse sensitivity should be set

				Step 5	Click on Apply button	Game play settings should be set
TC_4	**Select_LearnTo Play**	The user should be able to select the Learn to play option	The user should be on the home screen	Step 1	Click on the "Learn To Play" option on the home screen	A new window should open with the following options: a) Marching and Fighting b) Feeding the Army c)Training the troops d)Fighting Battles e)Forging Alliance
TC_5	**Select_History**	The user should be able to select the History option to learn about a particular era or civilization	The user should be on the home screen	Step 1	Click on the "History" option on the home screen	A new window will open with a list of all the civilizations
TC_6	**Research_Histor y**	The user should be able to research a particular civilization	The user should be on the home screen	Step 1	Click on the "History" option on the home screen	A new window will open with a list of all the civilizations
				Step 2	Click on any of the civilizations from the list	A new window will open with the historical facts about the civilization
TC_7	**Set_GamePara meters_SinglePl ayer**	The user should be able to set the game parameters such as population size, difficulty level, map style, civilization, location	The user should be on the home screen	Step 1	Click on the Single Player option on the home screen	A pop up will open with the following options: a)Standard Game b)Saved and Recorded Games
				Step 2	Click on Standard Game	Standard Game page should appear
				Step 3	Set the following parameters: a. Difficulty level(Easy, Moderate, Standard, Hard) b. Population size(100,200,300) c. Map style(Easy, Moderate, Standard, Hard)	The parameters should be set

					d. Location	
				Step 4	Click on Apply button	The parameter settings should be saved.
TC_8	**Start_Game**	The user should be able to start a new game	The user should be on the home screen	Step 1	Click on the Single Player option or multiplayer option on the home screen	A pop up will open with the following options: a)Standard Game b)Saved and Recorded Games
				Step 2	Click on Standard Game	Standard Game page should appear
				Step 3	Set the following parameters: a. Difficulty level(Easy, Moderate, Standard, Hard) b. Population size(100,200,300) c. Map style(Easy, Moderate, Standard, Hard) d. Location e. Civilization(Britons, Mongols, Goths, Persians)	The parameters should be set
				Step 4	Click on Apply button	The parameter settings should be saved.

				Step 5	Click on Start Game button	The round will load and display the map, the player's current score, a timer and the name of the player, number of each of the resources(wood, food, gold, stone), the total villagers and scout cavalry available with the player
TC_9	**Guide_movement**	The user should be able to guide the movement of villagers, scout cavalry and sheep	The user should have started the game	Step 1	Select any resource(villager, scout cavalry, sheep) and click on the spot where you want them to move to	The resource will move to the spot
TC_10	**Build_LumberCentre**	The user should be able to build lumber center, mills, farms ,docks, castle, town center and market	The user should have started the game	Step 1	Select the units to be built from the menu in the bottom left corner	The desired unit should get selected
				Step 2	Click on the spot where you want to position your unit	The villagers will start building the unit at the desired spot
TC_11	**Issue_Alert**	The system should issue an alert if an adversary enters the village/town	An adversary enters the town/village	Step 1	System issues an alert	The alert message “The enemy(enemy name) is in town”
TC_12	**Attack_Adversary**	The user should be able to attack the adversary	An adversary enters the town	Step 1	Click on the soldiers and guide them towards the adversaries	The soldiers should start moving towards the adversary
				Step 2	Select the weapons to use from the menu at the bottom left corner	The weapons should get selected
				Step 3	Click on the target using the left mouse button	The target should get selected
				Step 4	Click on the right mouse button to attack	The target should be attacked

TC_13	**Add_resources/units**	The user should be able to add new resources and units at any time	user must have some resources already	Step 1	Click on the resources to be added from the menu in the bottom left corner of the screen	The resources should get added
TC_14	**Press_RingTownBell**	The user should be able to press "Ring the Town Bell" option to signal the villagers to seek shelter during battle	An adversary enters the town	Step 1	Click on the soldiers and guide them towards the adversaries	The soldiers should start moving towards the adversary
				Step 2	Click on "Ring the Town Bell" button	The villagers should start moving inside their houses
TC_15	**Select_NextCivilization**	The user should be able to advance to the next civilization by selecting the next round provided they have sufficient resources	The user must possess sufficient resources to proceed to the next civilization	Step 1	Click on Escape button	A new window will open with a list of rounds and the requirements for each civilization
				Step 2	Select the civilization you want to choose	The desired civilization should get selected
TC_16	**Display_Leaderboard**	The user should be able to view the Leaderboard	The user should have started the game	Step 1	Click on Escape key	A new window should open with options to choose a new civilization and to view the leaderboard
				Step 2	Click on View Leaderboard button	A new window should open which displays the "Military Stats", "Economy Stats", "Society Stats", "Technology Stats" and "Total Score" of each player
TC_17	**Trade_resources**	The user should be able to trade goods with other players	User should have started the game	Step 1	Click on market	A new window should open with the list of all the players and the resources available with them and the options to buy or sell the resources

				Step 2	Click on the player name with whom you want to trade.	The player name should be selected
				Step 3	Click on Buy/Sell button	The amount of resources should get updated for each of the players involved in trade
TC_18	**Resume_savedGame**	The user should be able to resume a previously saved game	The user should be on the home screen	Step 1	Click on the Single Player option on the home screen	A pop up will open with the following options: a)Standard Game b)Saved and Recorded Games
				Step 2	Click on Saved and Recorded Games	A new window should open with the list of any previously saved games
				Step 3	Double Click on any of the games	The game should load
TC_19	**Build_Mills**	The user should be able to build mills	The user should have started the game	Step 1	Select the units to be built from the menu in the bottom left corner	The desired unit should get selected
				Step 2	Click on the spot where you want to position your unit	The villagers will start building the unit at the desired spot
TC_20	**Build_Farms**	The user should be able to build farms	The user should have started the game	Step 1	Select the units to be built from the menu in the bottom left corner	The desired unit should get selected
				Step 2	Click on the spot where you want to position your unit	The villagers will start building the unit at the desired spot
TC_21	**Build_Docks**	The user should be able to build docks	The user should have started the game	Step 1	Select the units to be built from the menu in the bottom left corner	The desired unit should get selected
				Step 2	Click on the spot where you want to position your unit	The villagers will start building the unit at the desired spot
TC_22	**Build_Castle**	The user should be able to build castle	The user should have started the game	Step 1	Select the units to be built from the menu in the bottom left corner	The desired unit should get selected

				Step 2	Click on the spot where you want to position your unit	The villagers will start building the unit at the desired spot
TC_23	**Build_TownCentre**	The user should be able to build town centre	The user should have started the game	Step 1	Select the units to be built from the menu in the bottom left corner	The desired unit should get selected
				Step 2	Click on the spot where you want to position your unit	The villagers will start building the unit at the desired spot
TC_24	**Build_Market**	The user should be able to build market	The user should have started the game	Step 1	Select the units to be built from the menu in the bottom left corner	The desired unit should get selected
				Step 2	Click on the spot where you want to position your unit	The villagers will start building the unit at the desired spot
TC_25	**Win_Conquest**	The user should be able to see the acquired resources from the adversary after winning the conquest	The user should have destroyed the adversary's units	Step 1	Click on the escape button	A new window should open with options to choose a new civilization and to view the leaderboard
				Step 2	Click on View Leaderboard button	A new window should open which displays that the "Military Stats", "Economy Stats", "Society Stats", "Technology Stats" and "Total Score" of the adversary displays zero and their resources get added to the winner's account.
TC_26	**Pause_Game**	The user should be able to pause an already started game	The user should have started the game	Step 1	Press "P" button on the keyboard	The game should pause and a pop up should open with the message "What would you like to do?" and two buttons a)Resume the Game b)Close the game

TC_27	**Auto_Save_SinglePlayer**	The user should be able to see an auto saved game after they have closed the game	The user should have started the game	Step 1	Press “P” button on the keyboard	The game should pause and a pop up should open with the message “What would you like to do?” and two buttons a)Resume Game b)Close game
				Step 2	Click on “Close game” option	The user navigates to Home screen
				Step 3	Click on Single Player option	A pop up will open with the following options: a)Standard Game b)Saved and Recorded Games
				Step 4	Click on “Saved and Recorded Games”	A new window should open which displays the presently closed game
TC_28	**Browse_Map**	The user should be able to browse through the map	The user should have started the game	Step 1	Hover mouse over the map	A hand mouse icon should display
				Step 2	Drag the hand icon across the map	Different areas on the map will be visible

32 Testing schedule

Test Level	Start Time	Duration	External Party	Project Team	Business
Unit Testing	3 weeks before each release.	1 week.		Primary	
Integration Testing	2 weeks before each release.	1 week.		Primary	
User Acceptance Testing	1 week before each release.	1 week.		Secondary	Primary
Security Testing	1 week before Global Availability (GA).	1 week.	Primary	Secondary	
Product Verification Testing	1 week before each product release.	1 week.		Secondary	Primary

Note: Where primary and secondary in the above table suggest who has Primary or Secondary responsibilities for the particular test level.

V Project Issues

33 Open Issues

Currently, we have decided to launch the game as an executable file. But, we believe that we could make the game available online. This will allow us to reach more people and will provide the users a more realistic gaming experience.

34 Off-the-Shelf Solutions

34a Ready-Made Products

Since ready-made products provide lower costs, minimal problems and unparalleled technical support, it would be wise to consider going that route for some software of hardware products during the development of this game. One of the products that can be taken from the market is the type of databases that will be used for this game. Two databases are needed, one for the user information and one for the companies' information. Microsoft SQL Server, MySQL and SQLite are among the few candidates for the databases that can be used off the market. It will be the decision of the development team to decide which product to eventually use.

34b Reusable Components

Instead of having to recreate hypothetical data and hypothetical historical facts, we will be using real historical facts and consult with historians. This will be a major help for the development team, as they can place more of their focus on the design of the game, instead of having to worry about creating and modifying data throughout. Getting data from historians or other existing source will also require less maintainability compare to self-produced hypothetical data, which will need to be updated after short period of time. However, since data that is based on real historical facts is from the past, it will not require as much updating.

Simulation models and simulation software should always be regarded as candidates for reuse. There are obstacles to simulation software and model reuse but these be surmounted. There is a range of levels at which simulation software may be reused, a range of costs to be borne and range of benefits that may be achieved. It is crucial to consider the issue of validity when considering model reuse and these needs to be a fundamental part of any reuse strategy. There may be circumstances in which reuse is economic, especially when a small, low-fidelity model will suffice.

34c Products that can be copied

As it turns out, there are quite a few other products on the market that are similar to the Rise of Civilizations that we would like to create. One such product is called "Age of Empires" and it can be found on steam.com. It is considered to be the largest strategy game in the United States. The purpose of the game is to provide individuals with a real life experience and feel of leading the Empire in the world. Their game is based on a fiction, where as our game uses real historical data from the past. We could get their specification as a starting point; it would have shorted the

development time for the rest of the game and also cut a lot of cost and effort as well. It could be tough to do so, however, as they may consider our product as a competitor to theirs. In such case, we will have to start from scratch.

35 New Problems

35a Effects on the Current Environment

The Rise of Civilizations game should not cause any harm to people in way shape or form. The users shall not be affected in any way by playing the game. This includes protecting user information and not distributing it to any third party source at any time. There shall not be any damage caused to any users or the environment by this game.

35b Effects on the Installed Systems

Since there is no system currently installed, there will not be any effect on any such system.

35c Potential User Problems

This game can be very addictive, therefore it is highly advised that the users shall take breaks every two hours between game play. It is important to understand that it is just a game at the end of the day and not real world simulation.

35d Follow-up Problems

If the user count exceeds 10,000, there shall be new arrangements to accommodate the new users. The current database will most likely will not be enough to take care of such a large number of audience. Therefore, it will be very important to switch to a new database.

36 Tasks

36a Project Planning

Software development life-cycle (SDLC), is used in the developing the game. It is a structure imposed on the development of a software product. There are several models for such life cycles, each describing approaches to a variety of tasks or activities that take place during the process. Some people consider a life-cycle model a more general term and a software development process a more specific term. For example, there are many specific software development processes that 'fit' the spiral life-cycle model. An international standard for software life-cycle processes aims to be the standard that defines all the tasks required for developing and maintaining software. Details of the life cycle and approach that will be used to deliver the product. The new product will be developed using SDLC. However, some circumstances are unique to a particular product and will necessitate changes to the life cycle. While these considerations are not product requirements, they are needed if the product is to be successfully developed.

36b Planning of the Development Phases

The development of the stock market simulator is planned in the following ways. First would Requirement Phase in which the Functional and Non Functional Requirements would be gathered followed by Analysis phase and design phase. In Implementation the coding would be done. It must be taken care the game should be able to deploy most of the Functional Requirements.

37 Migration to the New Product

There is no current product; therefore we will not be migrating from an existing product to the new product. This will be a new product, starting from scratch.

38 Risks

All projects involve risk—namely, the risk that something will go wrong. Risk is not necessarily a bad thing, as no progress is made without taking some risk. However, there is a difference between unmanaged risk—say, shooting dice at a craps table—and managed risk, where the probabilities are well understood and contingency plans are made. Risk is only a bad thing if the risks are ignored and they become problems. Risk management entails assessing which risks are most likely to apply to the project, deciding a course of action if they become problems, and monitoring projects to give early warnings of risks becoming problems. The following risks being the most serious:

• Management malpractice

• Inaccurate cost estimating

• Creeping user requirements

• Low quality

39 Costs

The cost estimate is the amount of resources estimate each type of deliverable will take to produce within the environment. Very early cost estimates can be created based on the work context. At that stage, the knowledge of the work will be general, and one should reflect this vagueness by making the cost estimate a range rather than a single figure.

The other cost requirements may include the amount of money or effort that we have to spend building them into a product. In this development at least one project meeting per week should be conducted. The online stock market stimulator has included more than four use cases. It has strict functional and nonfunctional requirements.

As knowledge of the requirements is increased using function point counting is suggested not because it is an inherently superior method, but because it is so widely

accepted. So much is known about function point counting that it is possible to make easy comparisons with other products and other installations' productivity.

It is important that the client be told at this stage what the product is likely to cost. This amount is described as the total cost to complete the product. The requirements-gathering process often throws up requirements that are beyond the sophistication of, or time allowed for, the current release of the product. The intention behind holding this requirement in waiting is to avoid stifling the creativity of the users and clients, by using a repository to retain future requirements. You are also managing expectations by making it clear that you take these requirements seriously, although they will not be part of the agreed-upon product.

Many people use the waiting room as a way of planning future versions of the product. Each requirement in the waiting room is tagged with its intended version number. As a requirement progresses closer to implementation, then time can be spend on it and add details such as the cost and benefit attached to that requirement.

40 Waiting Room

Other requirements that we would like to be implemented in the future releases include:

- Create game for mobile platforms, such as tablets/smartphones.
- Implement an option to buy virtual currency with the real money.
- Include articles about the company and the administrative members of the company.

41 Ideas for Solutions

When you gather requirements, you focus on finding out what the real requirements are and try to avoid coming up with solutions. However, when creative people start to think about a problem, they always generate ideas about potential solutions. This section of the template is a place to put those ideas so that you do not forget them and so that you can separate them from the real business requirements.

Content

Any idea for a solution that you think is worth keeping for future consideration. This can take the form of rough notes, sketches, pointers to other documents, pointers to people, pointers to existing products, and so on. The aim is to capture, with the least amount of effort, an idea that you can return to later.

Motivation

To make sure that good ideas are not lost. To help you separate requirements from solutions.

Considerations

While you are gathering requirements, you will inevitably have solution ideas; this section offers a way to capture them. Bear in mind that this section will not necessarily be included in every document that you publish.

42 Project Retrospective

This development project is being developed using the Waterfall Software Development Life Cycle. It was very helpful to divide the report into separate sections so that each section focused on one part of the report, such as requirements, design and testing report. The communication between groups is always challenging at the beginning, but we were able to coordinate times to accommodate everyone's schedule. It is essential that the team coding this project also find proper ways of communication that work for everyone in the group early in the development stage. Team members should also openly think about any problem and brainstorm for the solution.

VI Glossary

Player: The client on the user end of the game.

Computer: The artificial intelligence element of the game.

Simulation: Refers to gameplay situations carried out automatically

Flat: The basic land that takes up the majority of the map. Flat allows any unit or any building to be placed on it or move across it.

Water: Part of the map that only tanks can move pass through

Bridge: A way for units to cross the water terrain. Only marines can move across this terrain

Barracks: The building is used to create soldiers. Only four soldiers can be spawned around the barracks giving preference to the x-axis and then the y-axis

Factory: The building used to create machinery, only four machines can be spawned around the factory.

Soldiers: The basic unit, marines have no limit on how many can be created

Time Limit: The game is a turn based game. There will be 5 minute turns. After this amount of time has ended, the turn will be ended

Day: A day has taken place when all users have clicked the end turn button once

Player Money: This is how much a user has to build new units and buildings.

Current Player: This is the player who is currently playing the game. This is also shown in the team color.

Team Color: This is a reference so a user knows what their team color is.

Forest: Only marines can move across this terrain.

Mountains: Only marines can move across this terrain

VI References

- Bernd Bruegge, Allen H. Dutoit Object-Oriented Software Engineering Using UML, Patterns, and Java, 3rd Edition

- http://www.cs.uic.edu/~i440/
- https://en.wikipedia.org/wiki/Unified_Modeling_Language