CS 319 - Object-Oriented Software Engineering Analysis Report

World War 3

Group 1-F

Nurefşan Müsevitoğlu

Hüseyin Taşkesen

Halil İbrahim Çavdar

Doğacan Kaynak

Contents

1 Introduction

World War 3 is a strategy game which is inspired from Plants vs Zombies. There are lots of similar strategy games look alike World War 3 in gaming websites, app store, gaming portal etc. In this game the main purpose is to defend the world against robot invasion and survive for as long as possible with your carefully designed tactics and rational thinking. It makes even fun for player to survive with many types of human. Inspiration from both Plants vs Zombies and controversy topic about artificial intelligence turned into a strategy game. This inspiration created similarities as well as differences. These are some of differences between World War 3 and Plants vs Zombies:

1. Different bonus system for user to have more advantageous position in battle.
2. Game will have different level structure, not like Plants vs Zombies.
3. Different map structure than Plants vs Zombies.

However, both of World War 3 and Plants vs Zombies will have a little similarity on their gameplay. It will be a desktop application which can be played with mouse. If you don't know anything about Plants vs Zombies, you can look its trailer from the link:

https://www.ea.com/games/plants-vs-zombies/plants-vs-zombies-2

2 Overview

Basically, the map of our game is a simple rectangle which has 4 rows and 8 columns in it. Human's main objective is to defend the world against robots, but robots will not give up easily. There are different types of humans which has different abilities to lead the player to develop different strategies. Humans differ from each other with respect to their damage, health or weapon. The game is going to be a survival game. The number of robots will increase with respect to time. Robots will come from different rows and they will have a tendency to attack the road with less humans. As the paths robots going to take is not known, it is not easy for the user to make the best move at the right time to the right square.

2.1 Gameplay

Our game will be played with mouse clicks. On the left-hand side of the rectangle there is human's base. On the other side, the robots are born and attack spontaneously to the human's base to conquer the world. Player should choose the human type and place it to the square that satisfies his/her strategy. All kinds of human objects have different behaviors. To name and describe those human objects:

Shooter: This is a classic gun-man. Shoots one bullet at a time and costs less. It is the counterpart of Peashooter in Plants vs Zombies.

Double-Shooter: The classic gun-man that shoots two bullets at a time. Cost is also doubled. It is the counterpart of Repeater in Plants vs Zombies.

Freezer: This also has a gun. However, bullets coming out of this gun will decrease the speed of the robots. It is the counterpart of Snow pea in Plants vs Zombies.

Obstacle: This is kind of a shield or tank. It has much more health than any other human but it has also no damage. It is the counterpart of Wall-nut in Plants vs Zombies.

Miner: This is the resource human. This will supply diamond/money to the user to buy other humans. It is the counterpart of Sunflower in Plants vs Zombies.

LandMine: This is a one time exploding mine. It takes some time to charge. Then, when a robot steps on it, it explodes and kills all robots in a small area. It is the counterpart of Potato Mine in Plants vs Zombies.

Swordsman: This is our addition to the game. It will fight with robots head-to-head. But, it will not change its location and will wait for the robot to come close.

2.2 Enemies

Since the game is single-player, enemies will attack spontaneously to confuse player’s strategy. They will not make any fancy things once they entered the screen and you know their types. If they are fast, they will go fast all the way unless being hit by a Freezer human. If they are tanky they will be harder to drop. But, they will never make unexpected things. The difficulty of this game only becomes the number that they are, as the number of robots are increasing dramatically today.

The game algorithm will check all the rows to count the number of humans to attack smartly in the game.

2.3 Resources

Main resource of the game is money/diamond. These two names can be used interchangeably as the thing we use is money. But, as we have miners we think it is better to name it gold or diamond. Our choice had been diamond.

Money can be gathered from miners. Player should place miners to right places and protect them. In case of death of miners, player will not be able to earn money and it will result in losing the game against robots.

2.4 Strategies

Player should make his decisions wisely in order to be effective in the game. There are many strategies player can obtain in order to achieve his goal. Passive approach against robots is not recommended however, robots have strategy too. They will try to attack to the weakest part of the human’s base. Protecting the miners and showing the best defense strategy is also crucial.

3 Functional Requirements

3.1 Start Game

Starting the game will present different type of humans to user for the game strategy. World War 3 will be played using only mouse. Players will use mouse to interact with game and control their strategy through game UI. World War 3 will be a survival game and user's score will be computed with an algorithm that depends on the time that user has been playing the game. Player should visualize his strategy before starting the game; unless the robot's victory is inevitable. Placing the right human type to the right square must be logical to survive.

On the top of our map, there will be a rectangle block which includes buttons to help you during your game. There will be 5 different buttons:

Pause: Pauses the game.

Fast Forward: Makes everything work as x2; speeds of robots, reload time of guns etc.

Handbook: Represent the human types with their attributes and short stories.

Sound button: The user will be able to change game sound.

Quit: The user will exit the game.

3.2 Pause & Resume Game

Players can pause the game session anytime they need a break. In that case they will see the pause menu where they can resume playing or do some changes about the game.

There will be a pause button on the top of our map.

3.3 Change Settings

In World War 3, there are a couple of things user can adjust according to their preferences. Settings include the level of the volume and Hints page.

3.4 Credits

Credits will display the people who had spent effort in developing this game.

4 Non-functional Requirements

4.1 Performance

Performance of the game will be high, since our planned methods never use an algorithm which goes exponential. With good choices and aesthetic design of algorithms, we expect our game to run fast.

4.2 Graphics

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

4.3 Readability of Documentation

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

5 System Models

5.1 Use case model

5.1.1 Main Menu Use Case Model

Main menu use case of World War 3 is shown in detail below.

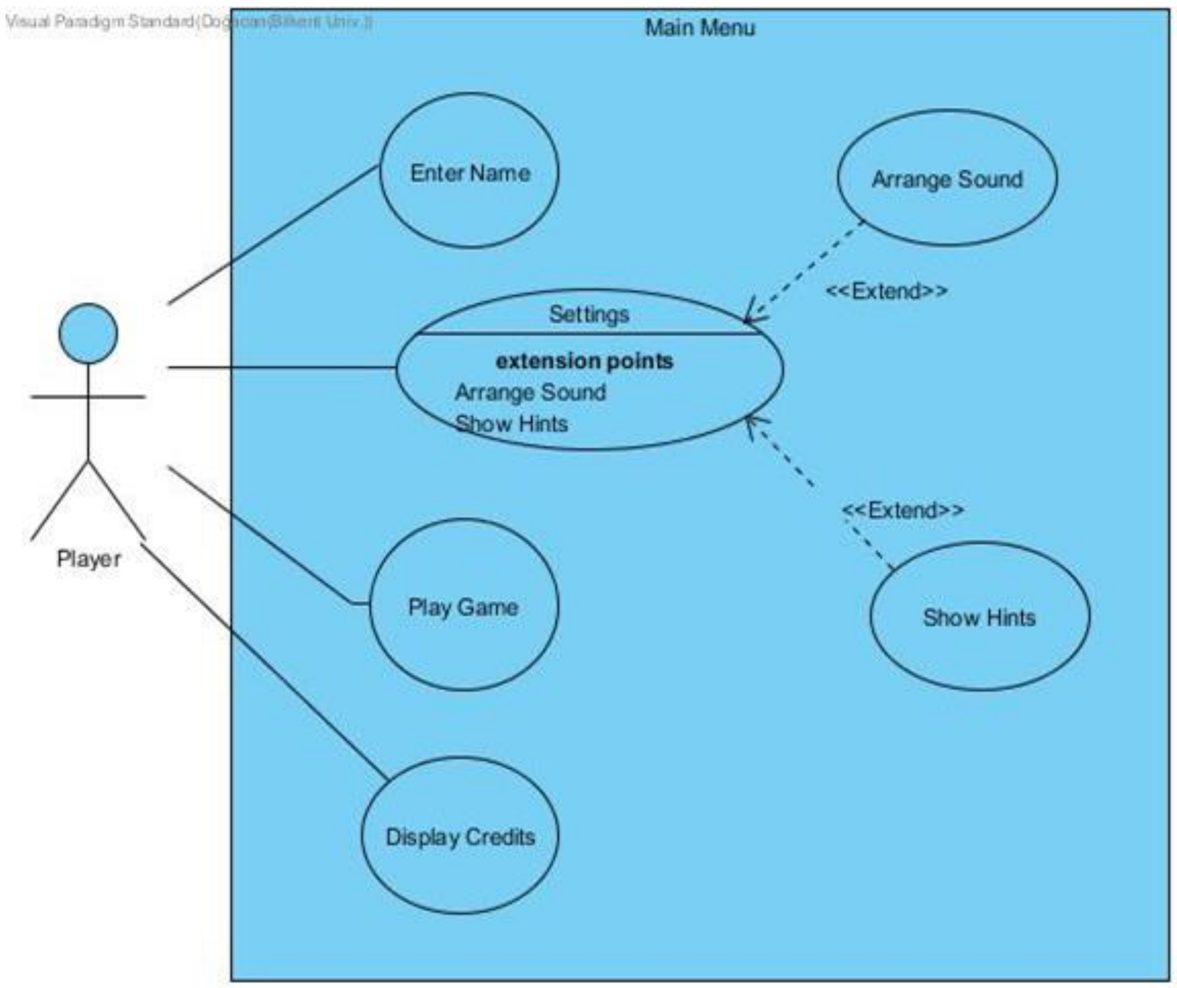


Figure 5.1.1 – Use Case Diagram

5.1.1.1 Main Menu Use Case Definitions

Use Case Name: Enter Name

Including Actors: Player

Pre-Conditions: System set a blank name for player

Entry Conditions: Player clicks on the bar via using main menu interface

Exit Conditions: Player clicks anywhere on the menu

Flow of Events:

1. Player clicks on the bar
2. After click, writes his/her name

3. Player clicks anywhere on the menu

Use Case Name: Settings

Including Actors: Player

Pre-Conditions: System is holding hints about game and music files which set to default.

Entry Conditions: Player clicks on “Setting” button.

Exit Conditions: Player clicks on back icon

Flow of Events:

1. Player clicks on “Setting” button.
2. Player arranges music according to himself/herself or look hints about game.
3. Exits by clicking back icon.

Use Case Name: Play Game

Including Actors: Player

Pre-Conditions: Music configuration of the game set to default unless it arranged from settings.

Entry Conditions: Player clicks on “Play Game” button via using main menü interface.

Exit Conditions: Player can select exit button via game menu toolbar. Player has to beat all robots.

Flow of Events:

1. Game starts by clicking “Play Game” button.
2. System starts from the first level.
3. Player clicks on exit button on the toolbar.

Alternative Flow of Events:

1. Game starts by clicking “Play Game” button.
2. System starts from the first level.
3. Player survives against robots using variety of soldiers.
4. Player wins/loses the game.
5. Game finishes and returns to main menu.

Use Case Name: Display Credits

Including Actors: Player

Pre-Conditions: The screen is set to show the people who worked on this game.

Entry Conditions: Player clicks on “Credits” button via using main menu interface.

Exit Conditions: Player clicks on back icon.

Flow of Events:

1. Player clicks on “Credits” button.
2. System displays the screen.
3. Player clicks on back icon to go back main menu.

5.1.2 InGameTopToolBar Use Case Model

Top tool bar use case of World War 3 is shown in detail below.

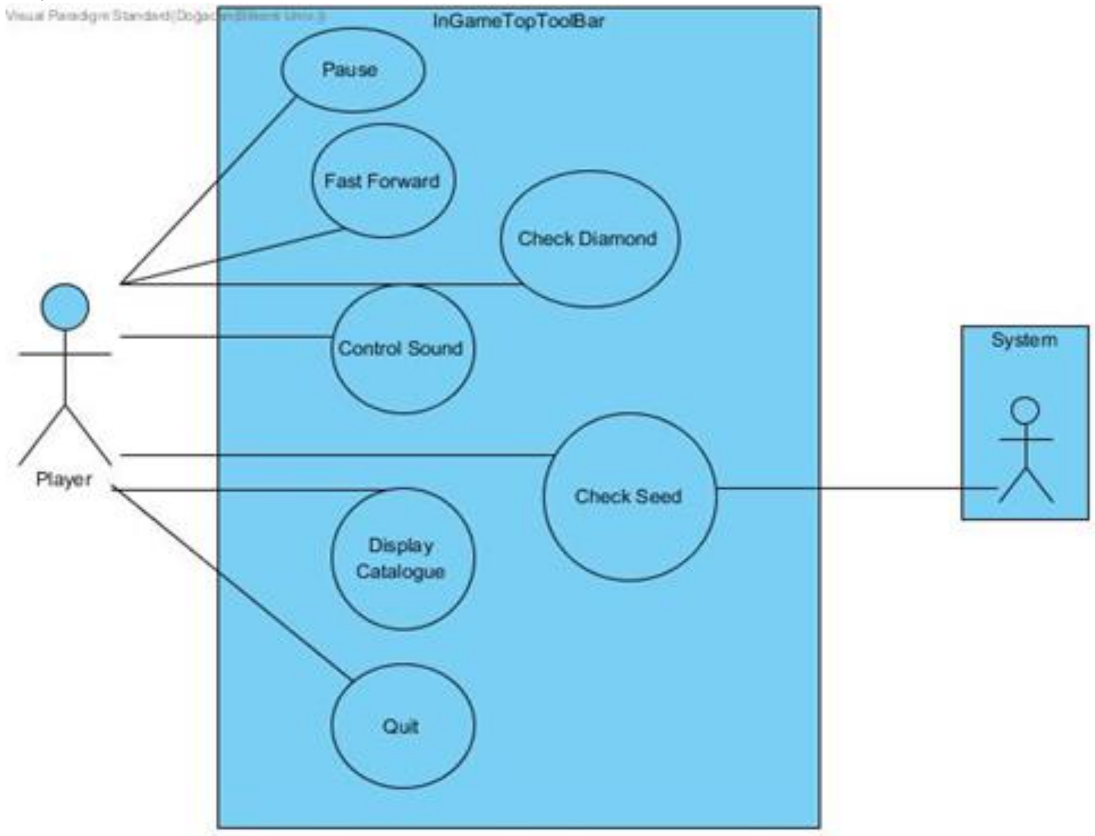


Figure 4.1.2 – Use Case Diagram

5.1.2.1 In Game Top Toolbar Use Case Definitions

Use Case Name: Pause

Including Actors: Player

Pre-Conditions: Game has to be running.

Entry Conditions: Player clicks on “Pause” button via using InGameTopToolbar interface.

Exit Conditions: Player clicks on “Pause” button via using InGameTopToolbar interface.

Flow of Events:

1. Player clicks on “Pause” button.
2. Game stops receiving inputs via keyboard or mouse until clicking to “Pause” button.

Use Case Name: Fast Forward

Including Actors: Player

Pre-Conditions: System has to be running.

Entry Conditions: Player clicks on “Fast Forward” button via using InGameTopToolbar interface.

Exit Conditions: Player clicks on “Fast Forward” button via using InGameTopToolbar interface.

Flow of Events:

1. Player clicks on “Fast Forward” button.
2. Game starts to run faster until clicking to “Fast Forward” button.

Use Case Name: Check Diamond

Including Actors: Player

Pre-Conditions: System contains diamond variables of player.

Entry Conditions: Player clicks on “Diamond” button via using InGameTopToolbar interface.

Exit Conditions: Player stops pressing on “Diamond” button.

Flow of Events:

1. Player clicks on “Diamond” button.
2. Player releases the button.
3. Pop-up diamond interface closes.

Use Case Name: Control Sound

Including Actors: Player

Pre-Conditions: System set music files to default.

Entry Conditions: Player clicks on “Sound” button via using InGameTopToolBar interface.

Exit Conditions: Player clicks on “Sound” button via using InGameTopToolBar interface.

Flow of Events:

1. Player clicks on “Sound” button.
2. Sound of the game stops until player clicks “Sound” button.

<u>Use Case Name:</u> <u>Check Seed</u>

Including Actors: Player, System

Pre-Conditions: System contains limited number of soldiers and robots.

Entry Conditions: Player places soldier by clicking on the map.

Exit Conditions: System places soldier on the location or player can’t place the soldier.

Flow of Events:

1. Player clicks on the map where he/she can place the soldiers.
2. System creates soldier or not according to slots emptiness.

<u>Use Case Name:</u> <u>Display Catalogue</u>

Including Actors: Player

Pre-Conditions: System contains information about soldiers attributes.

Entry Conditions: Player clicks on “Catalogue” button via using InGameTopToolBar interface.

Exit Conditions: Player stops clicking on “Catalogue” button via using InGameTopToolBar interface.

Flow of Events:

1. Player clicks on “Catalogue” button.
2. Player releases the button.
3. Pop-up catalogue closes.

<u>Use Case Name:</u> <u>Quit</u>

Including Actors: Player

Pre-Conditions: Game has to be running.

Entry Conditions: Player clicks on “Quit” button via using InGameTopToolBar interface.

Flow of Events:

1. Player clicks on “Quit” button and window pops up.
2. Player clicks “Yes” button, then returns to main menu.

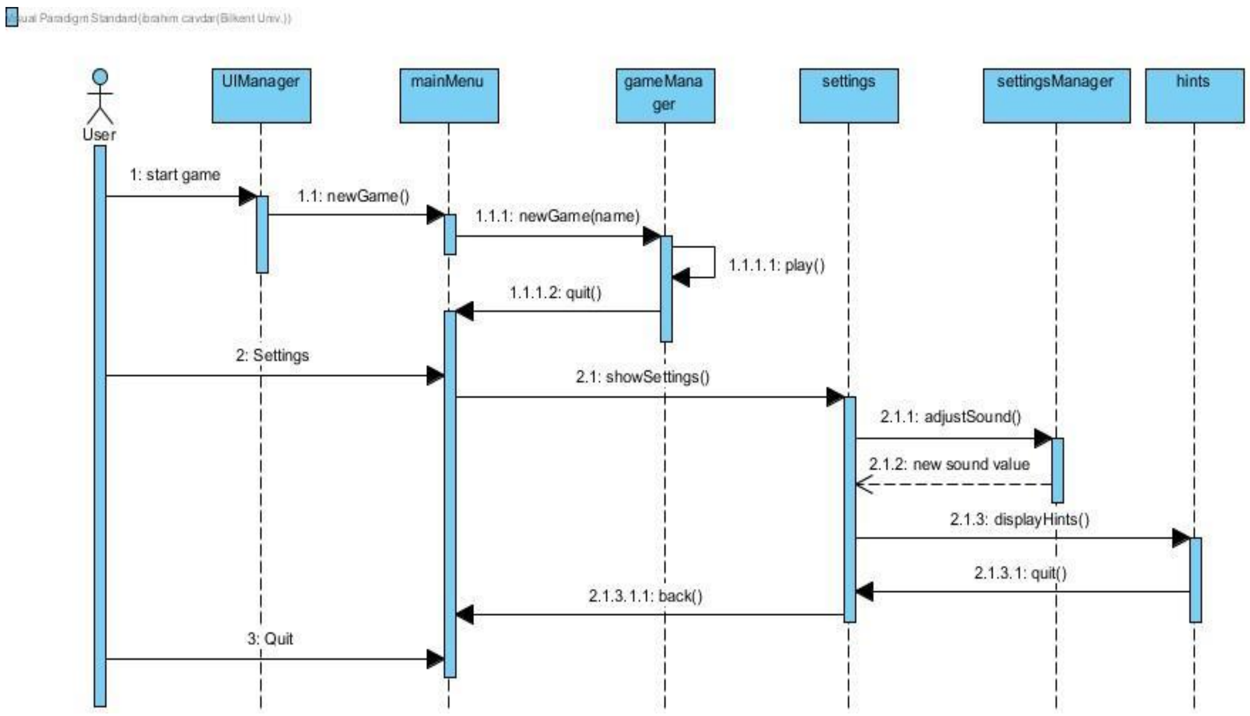
5.2 Dynamic model

5.2.1 Sequence Diagrams

5.2.1.1 *Interacting with menu*

Scenario: User wants to open the game after installing all files and make the application ready for use. He/she starts the application which results in working of main method. One UIManager object and related objects from other classes are created at the beginning. Then, the main menu is displayed to the user. The user wants to immediately play the game and presses the Play Game button. MainMenu object then takes this message and sends it to gameManager with the addition of user name, which the user enters into the box before pressing Play Game. Then gameManager object loads the necessary data and plays the game until quit button is pressed by the user or when the game ends.

He/she then proceeds to have a look at some hints or tutorials after getting demolished by the robots. Then he/she presses the settings button, which is the logical choice, and mainMenu object makes the connection possible to go to Settings page. He then realizes there is a slider to change the volume of the music and then he/she plays with it for some time, which leads to the connection between settings and settingsManager objects. Then, he/she decides to press the Hints button and displays the hints screen. After having a look at that page, he/she goes back to main menu and quits the game.

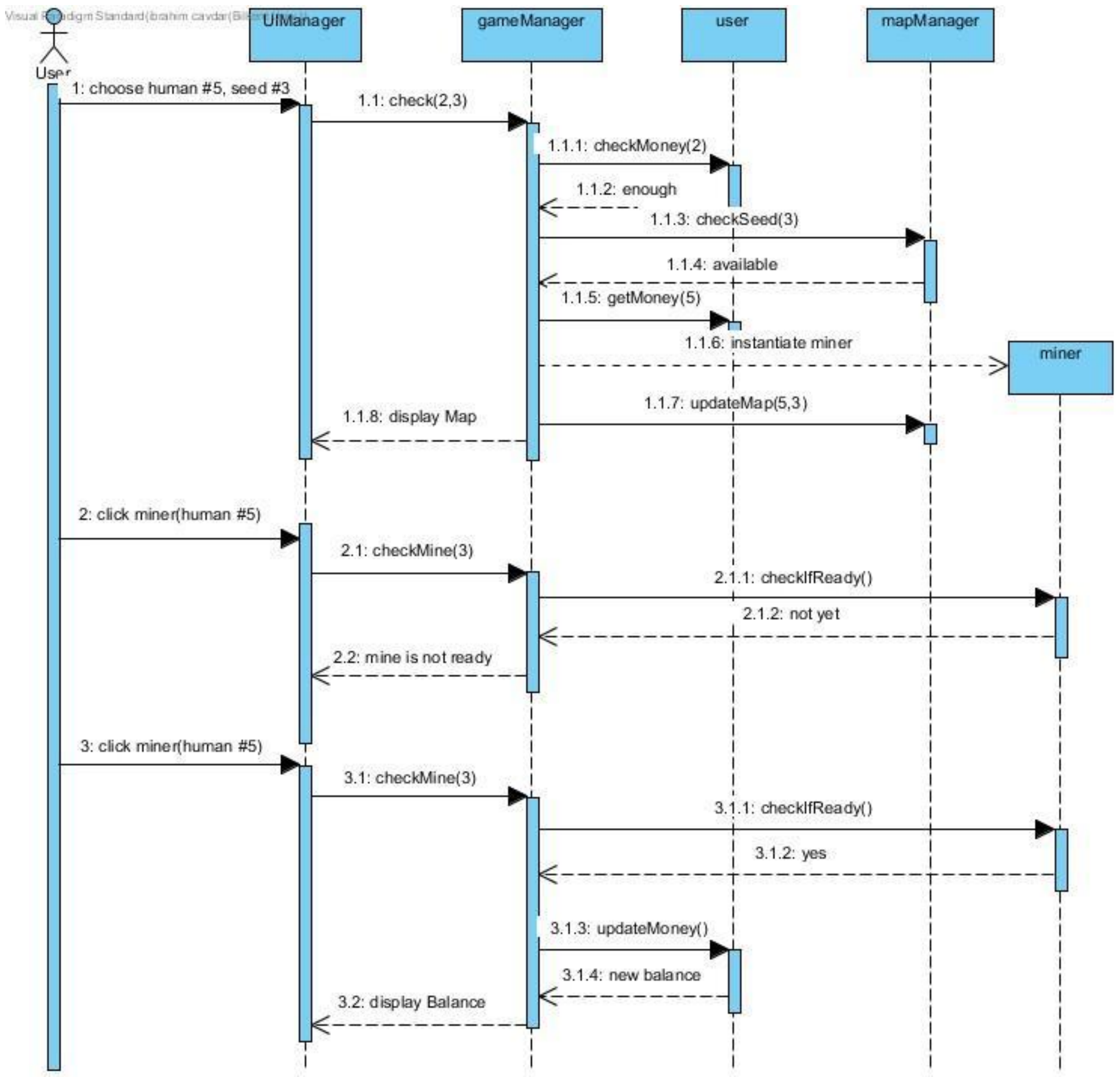


5.2.1.2 *Miner mechanism and gaining money*

Scenario: Our user wants to start a new game after reading hints page. He/she proceeds by pressing start game button as described in the previous part. After going into the game, he/she selects one kind of human and place for it. In this case these are human #5(which is the miner) and seed #3 which is the 3rd seed of the game's squares. The process after the selection is as follows:

1. UIManager object sends the information of humanCode and seedCode to gameManager,
2. gameManager then communicates with user object and decides if there is enough money to place a miner, which will return true as the user will have enough money to place a miner at the beginning,
3. after receiving the message from the user object, gameManager object communicates with mapManager object to decide if the seed is empty or there is a human on it already, which again will return true as it is the first human to be placed in the game,
4. after receiving the message from the mapManager object, gameManager communicates again with the user to get the money and update the balance of user accordingly, as the operation is valid after checking both seed and balance,
5. gameManager then will instantiate a Miner object and update the map accordingly, then returns the new map to UIManager.

After placing the miner, miner will mine some diamonds after a time interval. When the diamonds are ready to collect, the color of the miner image will change and the user will collect it by clicking on the icon. The process of checking if the diamonds are ready when the miner is clicked depends on the communication between gameManager and miner as shown in the below diagram.

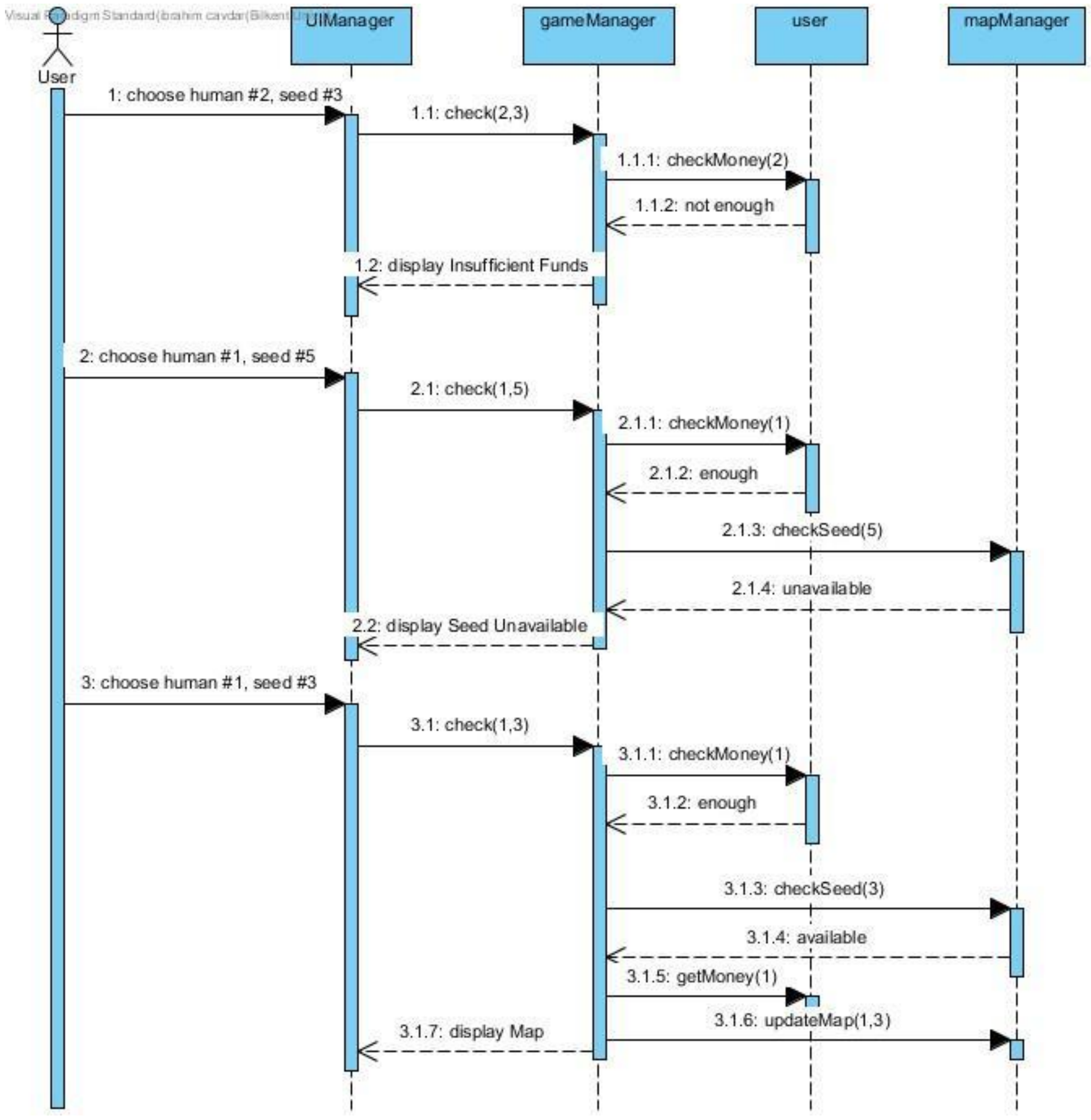


5.2.1.3 Placing a human

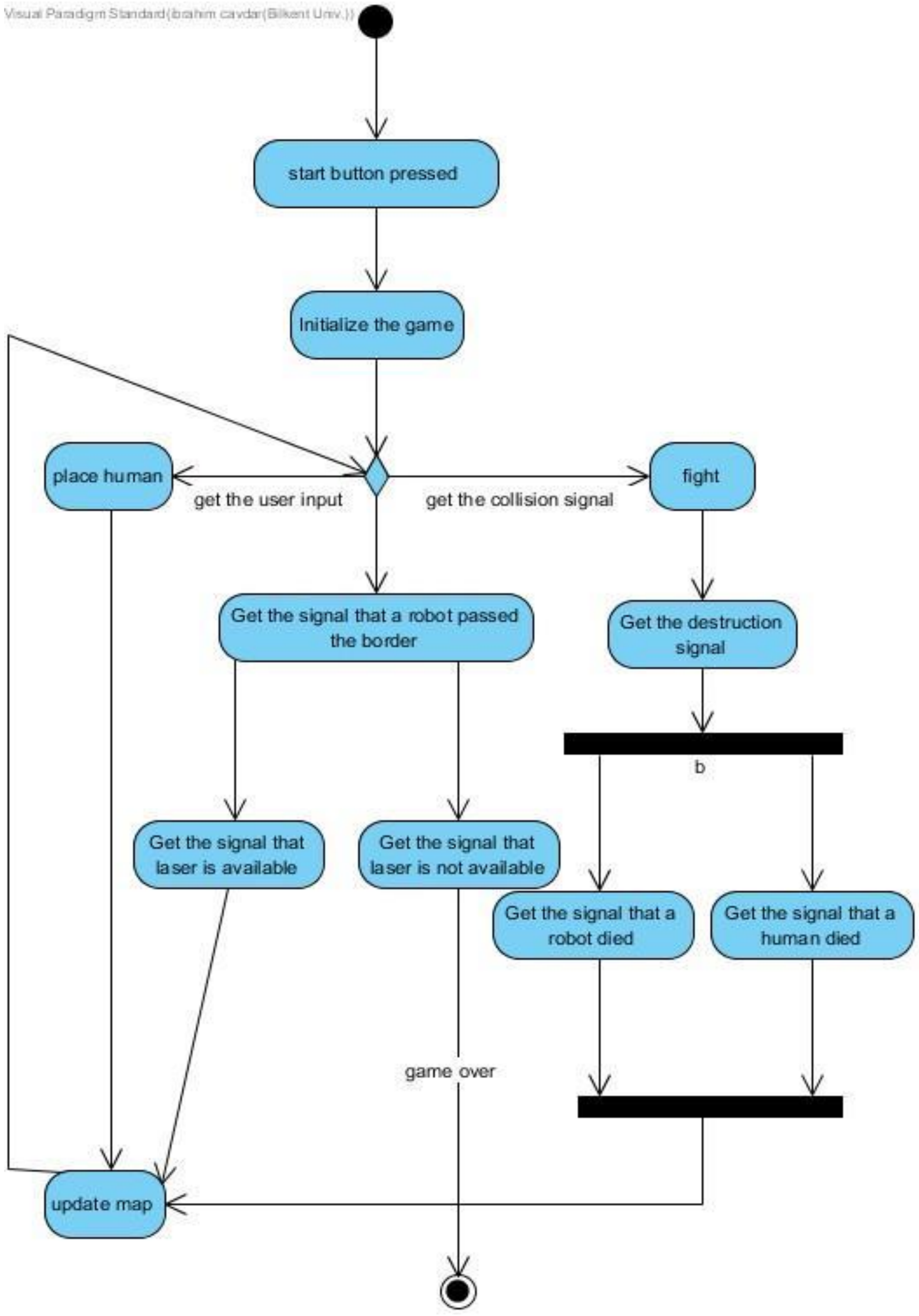
Scenario: In the middle of a game, where the user has 75 diamonds and one human at seed #5, he/she tries to place a human of code #2 at seed #3. The cost of #2 human is 100 diamonds so the system will respond with insufficient funds.

Then, he/she proceeds to place a human of code #1 after realizing his/her balance is 75 diamonds. His/her money is enough to place a #1 code human as it is 50 diamonds. However, this time the user accidentally clicks on seed #5 which already has a human on it. Therefore, the mapManager will respond with seed unavailable message.

Finally, the user pays attention and chooses human #1 and seed #3 and the process described above in the section 5.2.1.2 occurs without an error and the new human is placed to the ground.



5.2.2 Activity Diagram


Visual Paradigm Standard(ibrahim cavdar(Bilkent Univ.))
start button pressed
Initialize the game
place human
get the user input
get the collision signal
fight
Get the signal that a robot passed the border
Get the destruction signal
b
Get the signal that laser is available
Get the signal that laser is not available
Get the signal that a robot died
Get the signal that a human died
game over
update map

Explanation of Activity Diagram:

This activity diagram explains the flow of the game only. Only the game logic is taken into consideration.

In the beginning, Play Game button is pressed from the main menu which results in the initialization of the game. Then, the system ends up in a decision node which have 3 possible outcomes.

Case 1:

Placing a human is queried from the user. This will update the map and go back into the same decision node.

Case 2:

A collision signal is taken from the system. Because of the collision of ranges, one human and one robot will start fighting. At the end of it, one of them will win and the other will die. Then the map will be updated and the system goes back into the decision node.

Case 3:

A robot passed the borders signal comes from the system.

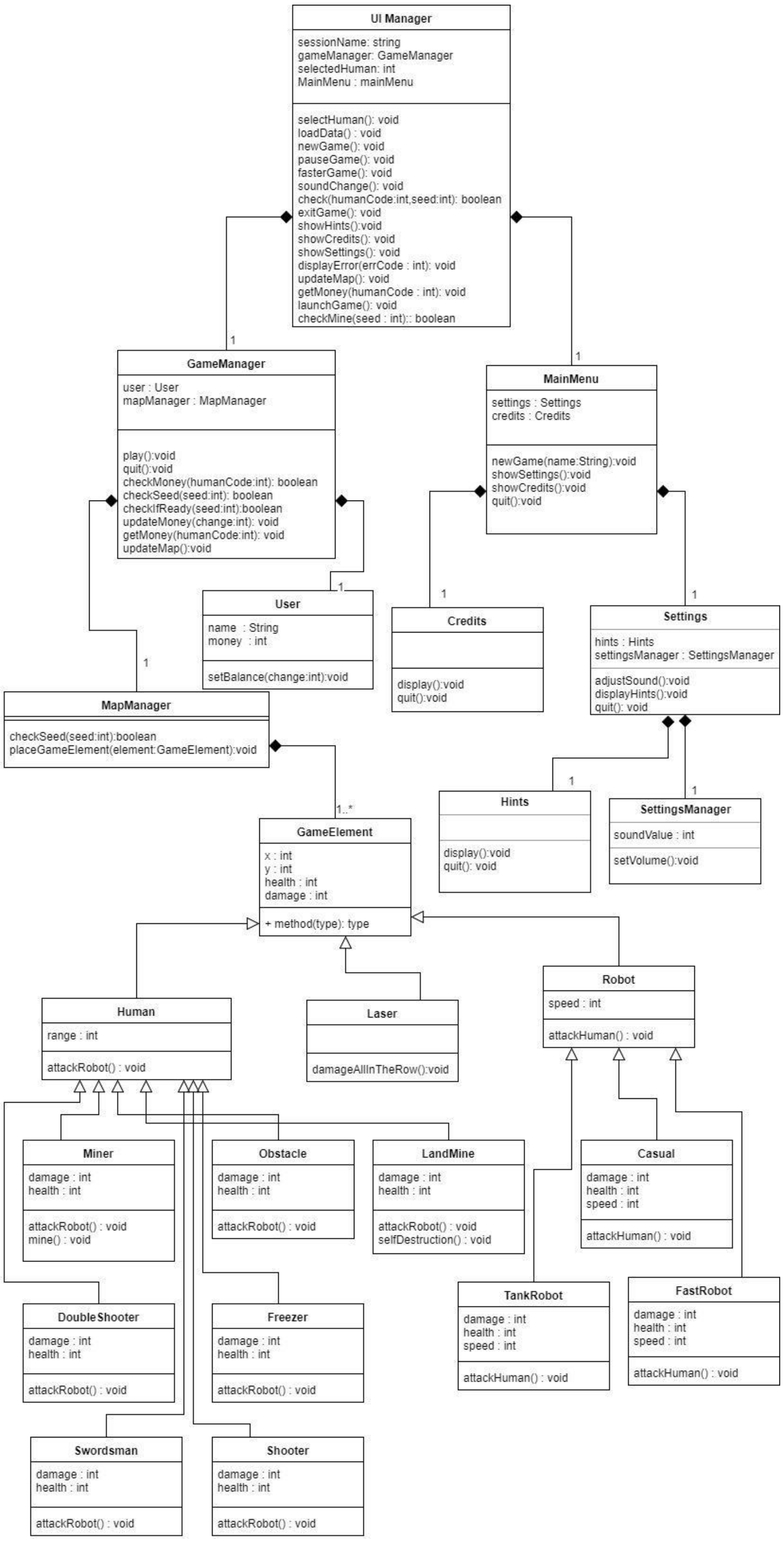
Case 3.a:

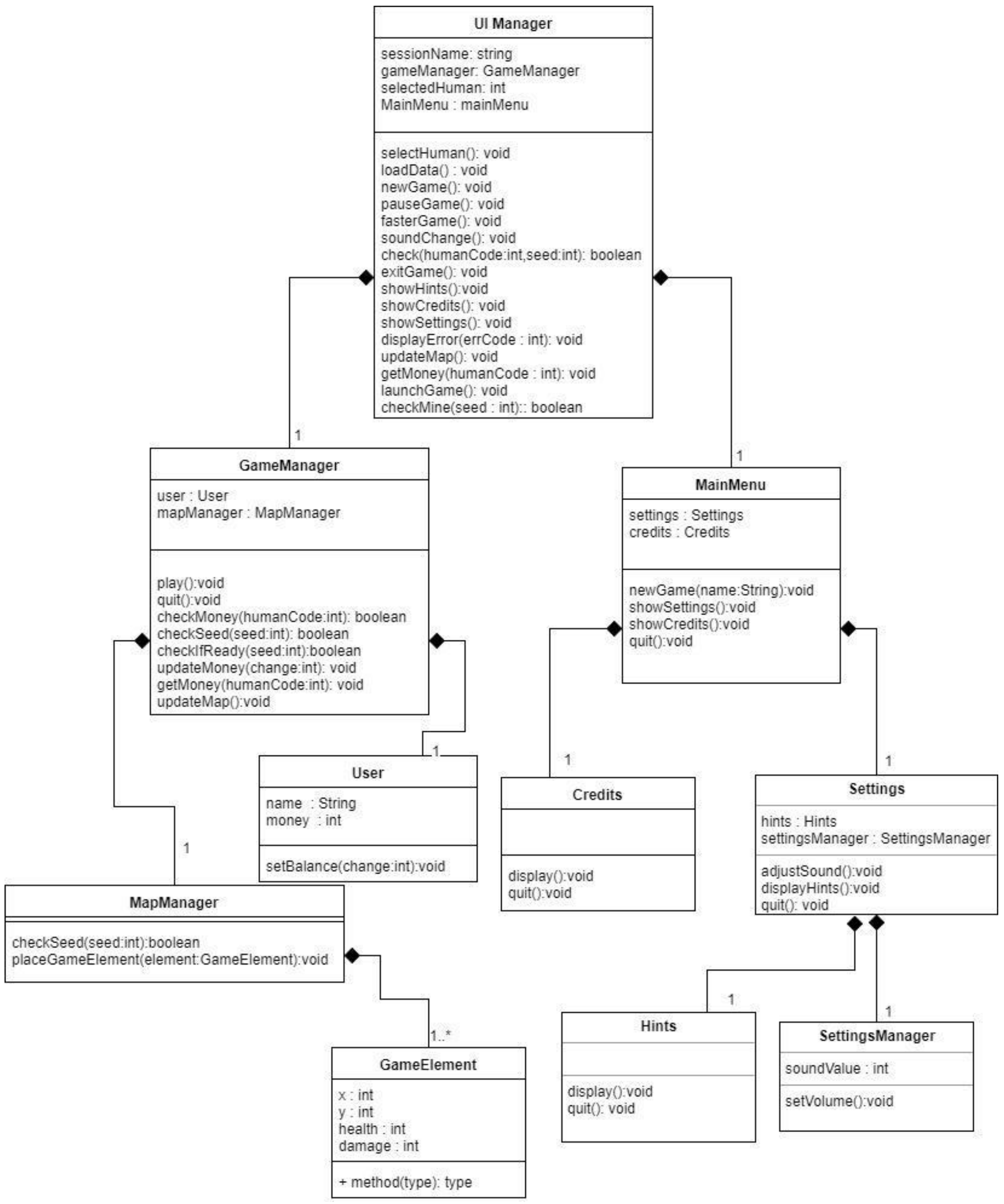
If it is the first time border is being passed in this row, the laser will clear the row and map will be updated. System goes back into decision node.

Case 3.b:

If it is the second time the border is being passed in a particular row, there will be no laser ready so the robots will destroy the world and the game will finish its execution. System will go into the terminal point of execution.

5.3 Object and Class Model





5.3.1.1 UIManager

This class makes the connection possible between classes. It consists of one object of two classes called GameManager and MainMenu.

5.3.1.2 GameManager

This class is the class that handles all of the game algorithm and connections to necessary classes such as MapManager, to load the map and interact with the map, and User , to control the money.

5.3.1.3 MainMenu
This class represents the main menu of the program. It communicates with UIManager to be able to change the displaying screen when some button is pressed. Holds two objects of Settings and Credits classes to navigate through them.

5.3.1.4 MapManager
This class, as its name suggests, manages all the control about the map, makes updates and communicates with GameManager about whether the action the user want to take is legal.

5.3.1.5 User
This class holds the required values of a player. Those are the money and the name of the player.

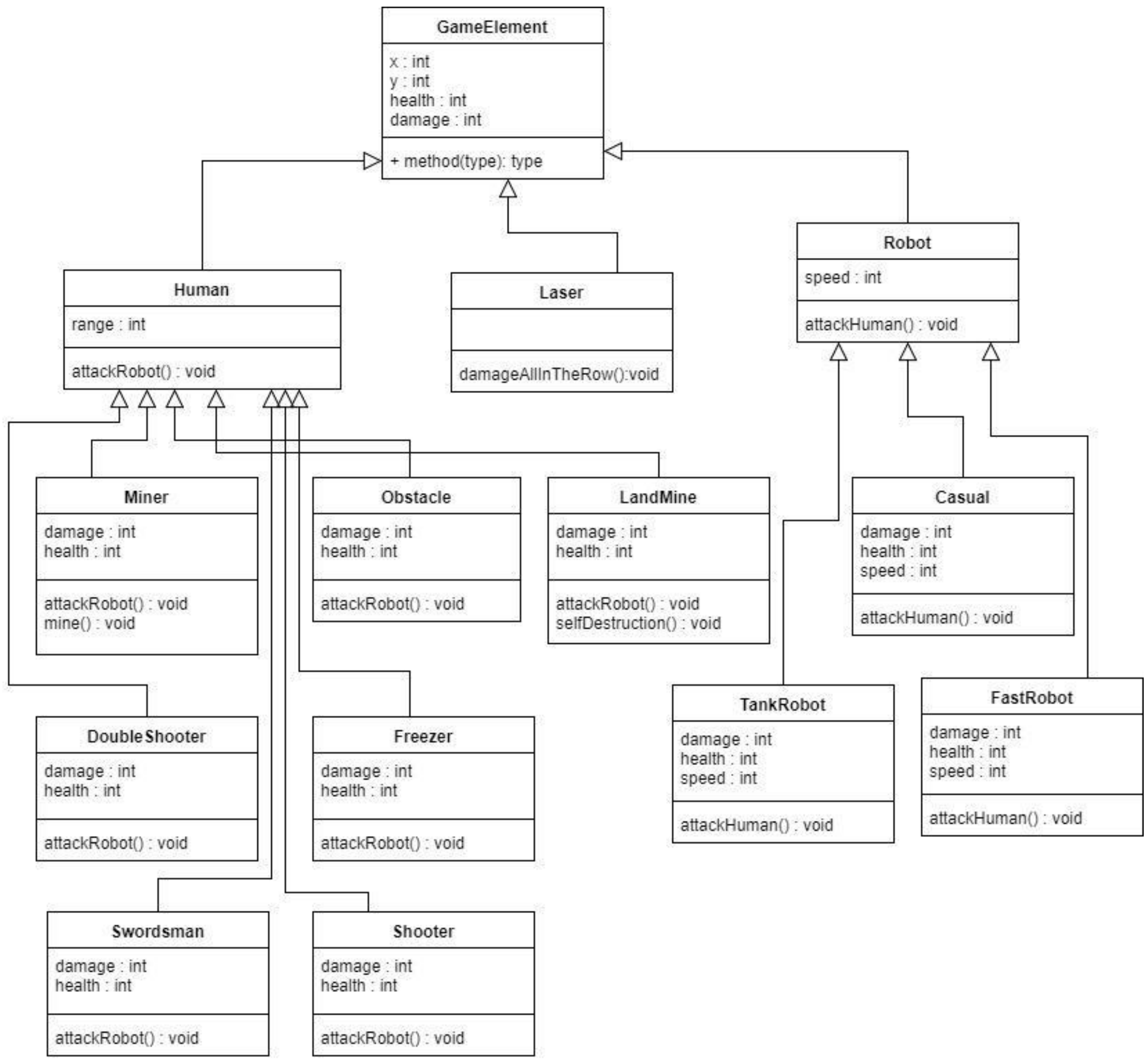
5.3.1.6 Credits
This class is there to display the Credits page. Has only display() function to display that screen and a quit() function which navigates back to main menu.

5.3.1.7 Settings
This class is there to display and be able to change the settings of the game. It holds two objects of Hints and SettingsManager classes.

5.3.1.8 Hints
This class is there to display Hints page. Similarly to the credit page it has no interaction with the user other than going into and going out from there.

5.3.1.9 SettingsManager
By this class we can adjust settings according to user's choice. It communicates with Settings class which is the class to display Settings.

5.3.1.10 GameElement
This class is an abstract class. It is the parent of all game related objects like humans, robots. As all of those elements have a location, this class contains x and y coordinates. Moreover, as all of them have health and damage, (some of them have 0 damage but it will not be a problem to change the damage variable to 0) this class contains them as class variables.



5.3.1.11 Human
This class is one level below from GameElement class. It is also an abstract class. It is the parent of all Human objects in the game. It has the class variable range which is the range that human can attack.

5.3.1.12 Laser
This class is one level below from GameElement class. It is also a normal class (not abstract). It is there to represent the laser at the base that kills all robots in one row when one robot reaches to base.

5.3.1.13 Robot
This class is one level below from GameElement class. It is also an abstract class. It is the parent of all Robot objects in the game. It has the class variable speed which is speed of all those robots.

5.3.1.14 Shooter
This is a human with range. It shoots one bullet at every small time interval.

5.3.1.15 DoubleShooter
This is another version of shooter which shoots two bullets at the same time.

5.3.1.16 Freezer
This is also another human with a gun. But the bullet of this gun will freeze the target. In other words declines the speed of the Robot class object that it hits.

5.3.1.17 Obstacle
This object has much more health than other human objects and it has no damage. So, the parameters of the class will be different than other Human objects.

5.3.1.18 Miner
This is the class that represents Miners in the game. They have 0 damage and they have an extra mine() function.

5.3.1.19 LandMine
This class represents the land mines that will explode when one robot steps onto it. To make it destruct itself, we have the extra function selfDestruction().

5.3.1.20 Swordsman
This is the class of Humans that fights with robots head-to-head. It has lower range than other damaging human classes.

5.3.1.21 Casual
This represents the normal robots that we face at every stage. These are the most common ones.

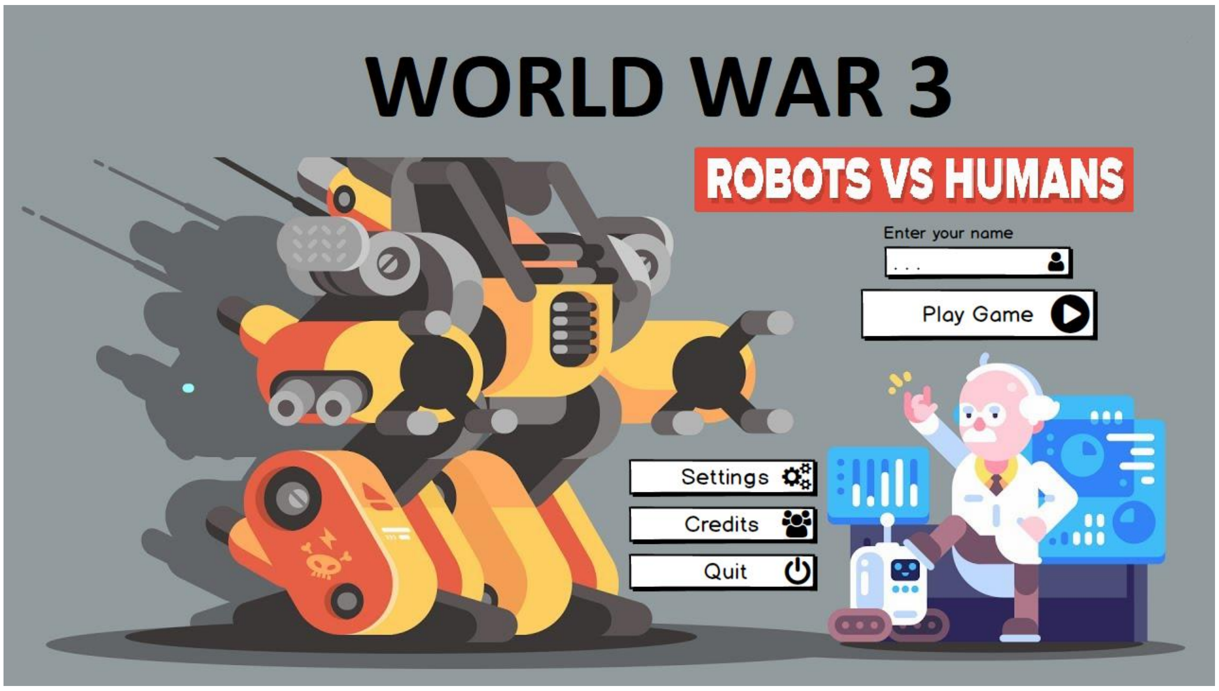
5.3.1.22 TankRobot
This robot will be slower than the casual one but will have much more health. So, we will adjust speed and health variables.

5.3.1.23 FastRobot
This robot will be faster. We will adjust speed variable of the class accordingly.

5.4 User Interface Design

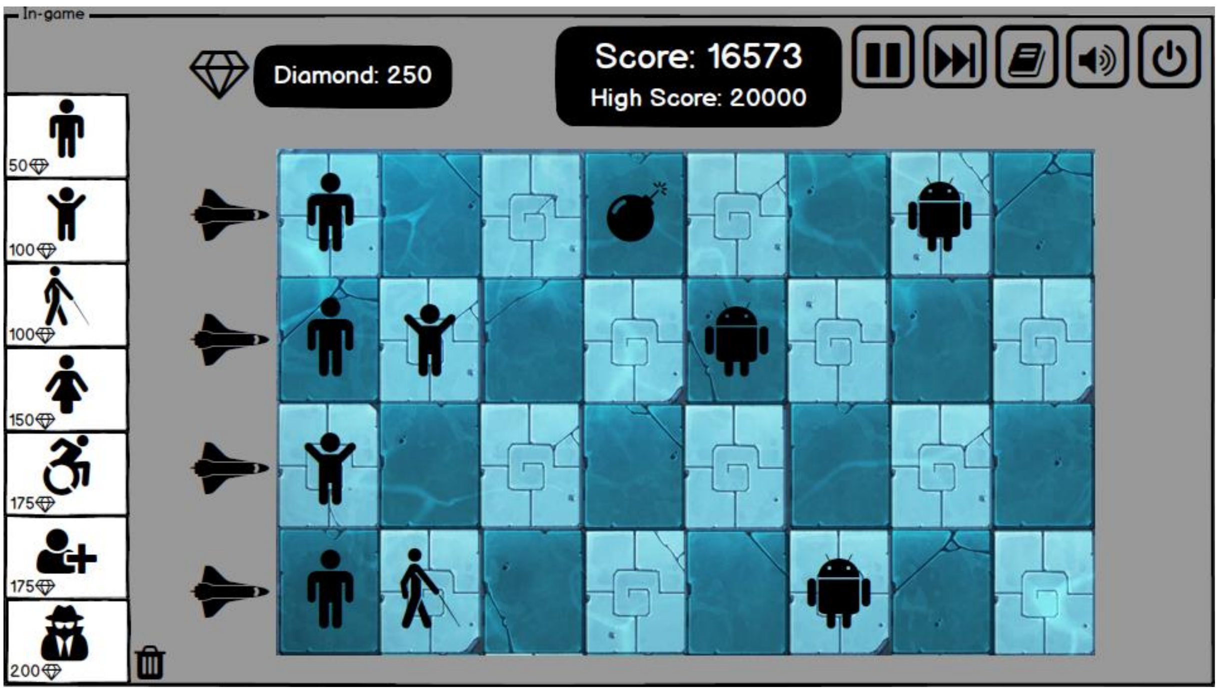
5.4.1 Main Menu

When the game starts, the first screen that will show up is main menu screen where the player is able to enter his/her name and start the game, rather she/he can select the other options such as settings, credits or quit.



5.4.2 In-Game Interface

The game play screen consist of a map, human character options and robot characters as enemies.



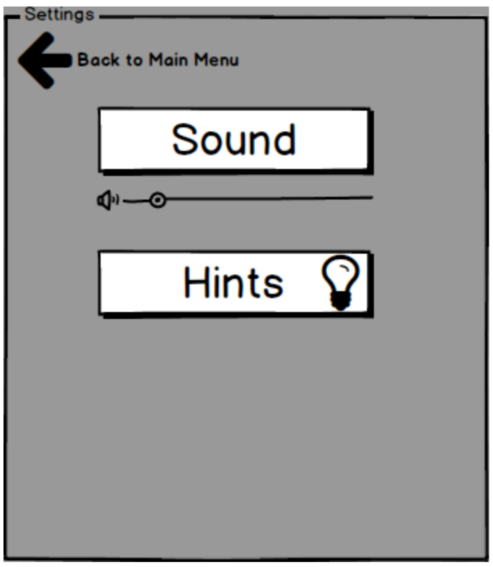
5.4.3 End of the Game

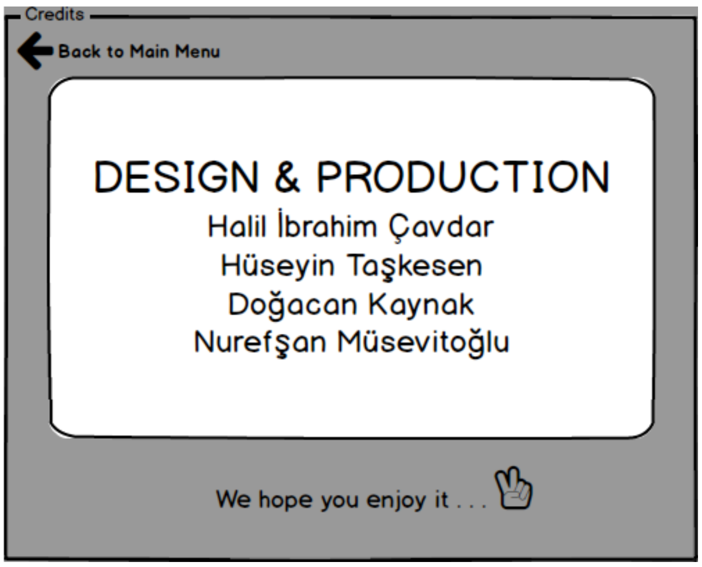
At the end of the game, score of the user will be displayed.

End of the Game
GAME OVER
You survived 17,580 much . . .
<3

5.4.4 Settings & Credits

In settings menu, player can arrange the sound and also display the hints including the information of how to play the game and specificaiton of human and robot characters in the game.





DESIGN & PRODUCTION
Halil İbrahim Çavdar
Hüseyin Taşkesen
Doğacan Kaynak
Nurefşan Müsevitoğlu

We hope you enjoy it . . .

6 Conclusion

In this analysis report, we specified our design and implementation methods to create a survival type strategy game, named World War 3. Our analysis report analyzes our game in two parts which are requirement specification and system model. The aim of this report is to show the main idea and the way we are thinking on working for our projects.

Requirement Specification is the part where we determine functional and non-functional units of the game. In functional units of the game, we represent the parts of the game that must work correctly in order to have a playable game. Nonfunctional requirements are mostly about the implementation process of our project.

Models used to define the system architecture are given below in order:

1. Use Case Model
2. Dynamic Models
    a. Sequence Diagrams
    b. Activity Diagram
3. Class and Object Model
4. User Interface Mockups

Use case reports have two use cases, first one is main menu use case which shows that in the main menu player is able to start the game, change settings or display credits. Second one is in game top toolbar where player can pause or fast forward the game, control sound, display the information catalog or quit the game.

Furthermore, dynamic models are very crucial to understand the flow of the game. Also they help us to create the heart and core of the game. In dynamic models, we included sequence diagrams and activity diagram. One of the sequence diagram is interacting with menu includes installing and starting the game. Second one is about miner mechanism and gaining money. It is very important because miner mechanism is related to all activities player does. Other sequence diagrams are about placing humans and. Finally, activity diagram is important to understand the general behavior of the game.

All in all, our analysis report is very detailed in terms of explaining many different cases in the game and has a big importance in creating a good project. This report creates a solid underground for the start of our project and will help us in the design report and during the implementation.

7 Glossary & References

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

2. http://holub.com/uml/
3. https://www.youtube.com/watch?v=Mj9OGupiC1s (main menu image)