CS 319 - Object-Oriented Software Engineering

Analysis Report

Last Man

Group 2-8

Abdullah Enes Akdoğan

Burcu Çanakcı

Yasemin Doğancı

Özgür Taşoluk

1 CONTENTS

2 Introduction 4
3 Requirements Analysis 4
3.1 Overview 4
3.1.1 Gameplay 4
3.1.2 Maps and Walls 5
3.1.3 Heroes 5
3.1.4 Default Weapon & Special Weapons 5
3.1.5 Packs 6
3.2 Functional Requirements 7
3.2.1 Create New Game 7
3.2.2 Select Number of Characters & Bot Level 7
3.2.3 Select Hero 7
3.2.4 Select Map 7
3.2.5 Select Maximum Game Time (Timer) 8
3.2.6 Start Game 8
3.2.7 End Game 8
3.2.8 See Results Screen 8
3.2.9 See Help 8
3.2.10 See Credits 8
3.2.11 Change Settings 8
3.3 Nonfunctional Requirements 9
3.3.1 Usability 9
3.3.2 Reliability 9
3.3.3 Performance 9
3.3.4 Supportability 9
3.4 Constraints 10
3.4.1 Implementation 10
3.4.2 Packaging 10
3.4.3 Legal 10
3.5 Main Scenario 11
3.6 Use Case Models 12

3.6.1 Use Case Model 12
3.6.2 Use Case Definitions 13
3.7 User Interface 16
3.7.1 Main Menu Screen 16
3.7.2 Game Creation Screen 16
3.7.3 Gameplay Screen 17
3.7.4 Results Screen 17
3.7.5 Settings Screen 18
4 Analysis 18
4.1 Object Model 18
4.1.1 Class Diagram 19
4.1.2 Domain Lexicon 20
4.2 Dynamic Models 22
4.2.1 State Chart Diagrams 22
4.2.2 Sequence Diagrams 24
5 Conclusion 29

2 INTRODUCTION

Last Man is a basic strategy video game we decided to develop. There are lots of strategy type video games in market and each of them has its different features. Last Man is a game which is very similar to the basic Bomberman game, and the main purpose in the game is to place bombs and use different weapons in order to kill enemies and destroy the walls. The game we were influenced by, Bomberman, can be seen in the following link:
http://tr.y8.com/games/bomber_man

We plan to develop our game with different features. In Last Man, there will be different heroes to select, special weapons, and different packs that appear during gameplay. There will be different game maps to choose from, and although Last Man does not have a level-based system, different maps will give the user the experience of encountering different levels.

The game will be a desktop application and will be controlled by both mouse and keyboard.

3 REQUIREMENTS ANALYSIS

3.1 OVERVIEW

Last Man is a strategy game like the Bomberman. Like the game it's based on, it is fun and very easy to play at the same time. The game provides the user a map to play in and this map consists of different walls (obstacles) and paths, and the game also has different enemies to beat. Player tries to destroy the obstacles and kill his/her enemies by placing bombs and using weapons in the desired locations. Every explosion (meaning the bomb/weapon is used in some location on the map), can either damage enemies, or damage the walls. Enemies are destroyed depending on their health points and the walls crack or are destroyed depending on their type. If the player destroys all the enemies in the map, the player wins the game. The goal is to kill enemies before the player dies. In our project, we named these computer controlled enemies **Bots**, and the user-controlled player **the Player**. Both Bots and the Player are **characters** in the game.

3.1.1 Gameplay

The player can use the mouse and keyboard to play. Actions like using weapons are fulfilled by using specific buttons at the keyboard, and the movement of the player is provided by the arrow keys. Mouse

will be useful while choosing the game specifications and while the player is in the main menu of the game.

3.1.2 Maps and Walls

Last Man will have different maps and these maps differ in the type and number of the walls they have. Walls are designed to make the game more fun and challenging. There are 3 type of walls and they differ in their resistance.

Normal walls: Can be destroyed by one hit by Weapon 1 (default weapon of all heroes – the bomb)
Strong walls: Can be destroyed by one hit by Weapon 2 (special weapon of heroes if specified) or by two hits with the default weapon.
Unbreakable walls: The default and unbreakable walls of the map.

3.1.3 Heroes

There are various types of heroes to choose from. The heroes differ from each other as their health points, speed and special weapons are diverse. Different hero selections will help player to enjoy the game even more and make the game more or less difficult. Below are some of heroes to choose from to play Last Man:

Table 1 List of Heroes

Name	Speed(1-10)	HP(1-1000)	Special Weapon
Kil	10	800	Lightning
Jas	8	900	Arrows
Ustah	8	700	Building Walls
Ayibogan	5	1000	Unbearable Attack
Drogon	7	760	Fireball
Nemo	8	650	Armageddon

3.1.4 Default Weapon & Special Weapons

Weapon types are the essential part of the game and they make the game very different from the basic Bomberman. The game style changes according to the hero selection of the player. **Damage** represents the damage of the weapon. **Delay** represents how often they can be used.

Table 2 List of Weapons

Name	Description	Damage	Delay
Default weapon – The bomb	All heroes have this default weapon.	100	Depends on assigned hero
Lightning	Hits specific location in the map	250	10s
Building Wall	Builds wall in a specific location. Wall type is normal.	-	2s
Arrow	Hero sends 3 arrow at intervals of 1 seconds to the row and column which hero standing. Arrows can pass through walls. Arrow goes until it reaches the boundaries of the map or until it hits another player.	60	15s
Armageddon	Hits different places in the map without any delay.	75	20s
Fireball	Destroys the walls in its path and damages all enemies in the path. Fireball's direction is specified as the hero's direction. (If the hero is facing North, fireball is sent in North direction.)	250	12s
Unbearable attack	Hero gains 10 second strength. Hero is able to damage any enemies in his path.	400	35s

3.1.5 Packs

Packs are the basic power-ups and power-downs which occur during gameplay. They make the game even more fun and enjoyable. The player will not be able to know what the package has, so it will be a surprise for him/her and eventually the game will become easier or harder. The packs will appear in specific locations on map and the number of packs and their types will be random during one game. The packs are as follows:

Shield Pack: For 15 seconds the hero cannot take any damage.
Increase Speed Pack: Hero's speed is increased for 15 seconds.
Decrease Speed Pack: Hero's speed is decreased for 15 seconds.

First Aid Pack: Hero's Health Points increase by 250.

Range Bonus Pack: Default weapon's range is increased.

Extra Weapon Pack: The player can use default weapon more than once. The number of weapons used before explosion is increasing in a cumulative manner.

3.2 FUNCTIONAL REQUIREMENTS

3.2.1 Create New Game

From the main menu of the game, the user will be able to choose from one of several options. One of them is the New Game option, which will proceed the user to a sequence of selections in order to start the game, when it's chosen.

3.2.2 Select Number of Characters & Bot Level

This is the first selection of the sequence. Here the user can select to play against from 1 to 3 Bots, so they can create a game that has 2 to 4 characters.

After that, the player can decide the skill level of the Bots. There will be 3 basic levels based on Bots' skill, defined as: Easy, Medium and Hard. These levels are based on Bots' ability to consume and/or use the packs. In Easy difficulty, Bots won't consume the packs; in Medium difficulty, they will consume the packs but will not be able to use them. This will cause a change in the game because in Medium difficulty, the Player will have a hard time when it comes to gaining packs. In Hard difficulty, Bots will be able to consume and use the packs, and therefore the game will be harder than usual.

3.2.3 Select Hero

Hero selection is another requirement of the game. The Player will be able to choose from various hero types. Heroes differ from each other as their initial health points, speed, damage potential and weapons change. Hero selection gives the Player the chance to gain experience in different playstyles.

3.2.4 Select Map

There are different types of maps the game can be played with. These maps differ in the amount of walls they have, in the properties of the walls they have and in the placement of these walls. The Player should be able to select one from these maps. The variety of maps makes the game more dynamic.

3.2.5 Select Maximum Game Time (Timer)

Just before starting the game, the user will set a timer value for the maximum length of the game. The input the user provides for this will be checked for validity before the game is started. When the specified time is reached, the player with the highest health points immediately wins.

3.2.6 Start Game

Start Game function will appear after every selection is made and when the Player chooses this option, their specifically created game will start. The Player begins to play and their hero will act according to their specifications.

3.2.7 End Game

Except from the game's ending conditions (death of Player, death of all bots or timer runs out), the Player should be able to end the game manually by clicking a button from the game view.

3.2.8 See Results Screen

Once the game ends, the user should be able to see a results screen that consists of the characters in the game and their rankings according to their health points and death times.

3.2.9 See Help

From the main menu, the user should be able to click on the help button in order to see the picture-text tutorials that teach how to play the game. This help section will provide detailed information about the rules and controls of the game and specifications about maps, heroes and packs.

3.2.10 See Credits

Again from the main menu, the user will be able to click on the credits button for information about the developers.

3.2.11 Change Settings

The user should also be able to adjust sound and music settings by clicking on the settings button from the main menu. The music option can be turned off in which case just the sound effects will be active. The volume of these effects can also be adjusted from this settings option.

3.3 NONFUNCTIONAL REQUIREMENTS

3.3.1 Usability

- The game should be accessible, that is it should be easy to use by as many people as possible. For this, the user interface will be made simple and consistent.
- The amount of user documentation provided in the help section should also be sufficient for anyone with minimal computer experience to enjoy the game.

3.3.2 Reliability

- The inputs given to the game are almost always selected from a pre-determined menu, therefore unexpected cases are hard to occur.
- The game should always work smoothly independent of users' choices, however, if an event of failure occurs, the game can be manually shut down and restarted.
- This is not a level-based game, therefore no data is stored once the current game is over. Hence, data loss will not be a problem.
- It is also a simple game with basic graphics so it should not create stressful conditions for the computer.

3.3.3 Performance

- The game should not take more than one minute to load. The users should be able to iterate through game creation smoothly and once the game starts it should never stall.
- While creating the game, the user can take as much time as they want at each step, so there will be no time constraints for the user during game creation.
- The game supports only one player. Only one game can be played at a time.
- The game does not store permanent data, so there shouldn't be storage problems. Since this a simple game, the maximum latency shouldn't be more than one minute.

3.3.4 Supportability

- The game can be played on any desktop or laptop computer.
- The game is very basic and therefore should require minimal maintenance.
- An extension to the game can be making it a multiplayer game that people can play from different computers. This game builds on simple fundamental game ideas, so it is open to a lot of creative extensions.

3.4 CONSTRAINTS

3.4.1 Implementation

- The game will be programmed using the Java programming language and the Swing or OpenGL framework.
- The game will be developed on laptop computers. Since it is to be programmed with Java, it can be developed and run on any computer with any operating system.
- Software like Photoshop, Paint.NET or AutoDesk Sketchbook will be used while creating the user interface.
- Software like Audacity will be used to manage sound effects and music.

3.4.2 Packaging

The game requires no installation. It can be downloaded as a .jar file and run smoothly.

3.4.3 Legal

This will be a non-commercial open-source project. The code and the game will be accessible to everyone by GitHub. All software and techniques used to develop the game are open source or already licensed so there will be no royalties or licensing fees.

3.5 MAIN SCENARIO

Scenario name mainScenario

Participating actor instances alice: Player

Flow of events

1. Alice clicks on the game icon from the desktop and opens the game. After the game loads, she activates the “Create New Game” system by choosing the “New Game” option from the main menu.

2. Alice selects to play with 4 characters in total and sets bots’ skill level to hard.

3. She selects the hero named “Kil” and the map named “Mountain”.

4. She sets the timer to 5 minutes and starts the game.

5. Alice uses Kil’s speed skills to avoid getting hit by weapons and successfully manages to place weapons near the Bots. She manages to make all of the Bots’ health points hit 0 by the third minute and so she wins the game.

6. The game ends and the program directs Alice to the results screen. This screen shows Alice in the first place and then the rest of the Bots in order from the one that was killed last to the one that was killed first.

7. Alice chooses to go back to the main menu from the results screen.

8. Alice chooses the “Change Settings” option from the main menu. The program shows her the settings menu. She chooses to turn the music off and she lowers down the sound level. She chooses the “Save Settings” option and a confirmation message is shown. Alice goes back to the main menu.

9. Alice chooses the “See Help” option from the main menu. The program creates a pop-up window with the user documentation. Alice observes the documentation and closes the window. She is back at the main window of the program.

10. Alice closes the main window of the program, a confirmation pop-up box appears. She confirms and exits the game.

This is the main usage scenario of our program. Several smaller and more specific scenarios are provided with the sequence diagrams.

3.6 USE CASE MODELS

3.6.1 Use Case Model

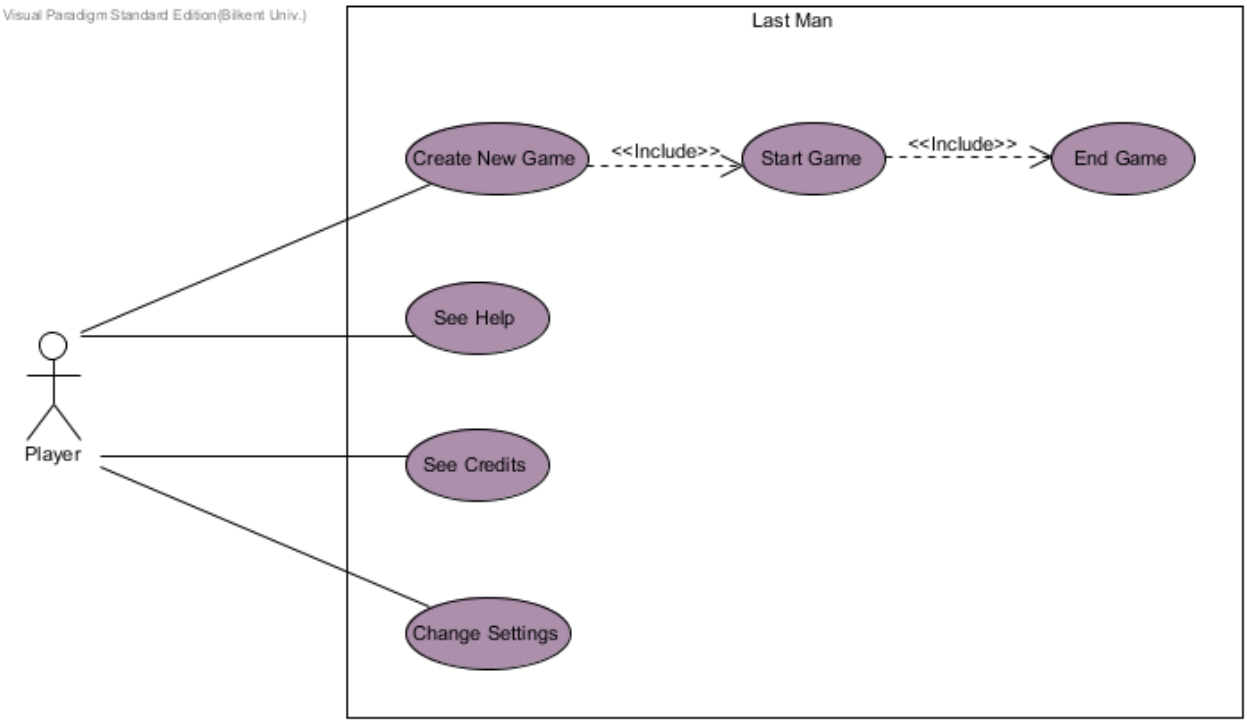

Figure 1 Use Case Diagram

Above is the main use case diagram for “Last Man”. From the main menu, the player can create new game, see user documentation/help, see credits and change settings. During creating new game, they can choose to start the game and start playing. While playing they can choose to end the game. The use cases are investigated in detail below.

3.6.2 Use Case Definitions

Use case name	**CreateNewGame**
Participating actors	Initiated by Player
Flow of events	1. Player selects the "New Game" option from the main menu of the game 2. The program creates a default Game and starts to provide the Player with a series of selections to set up the game. The program dynamically updates the game after each selection according to the Player's choices. 3. The first selection is to determine the number of characters and general Bot skill level. Player chooses to play a 2-4 character-game and with Bot skill from 3 difficulty levels: Easy, Medium, Hard. 4. Player chooses a Hero from the Hero list provided based on their preference. 5. Player chooses a Map from the Map list provided. 6. Player sets up the maximum time for the game by entering time in seconds. 7. The program checks the value entered for validity, asks until a valid response. 8. The program finishes updating the game and provides Player with the option "Start Game".
Entry condition	Player has opened the game and chosen the "New Game" option.
Exit conditions	Player is provided with a set-up game of their preference to play or not.

Use case name	**SeeHelp**
Participating actors	Initiated by Player
Flow of events	1. Player chooses the "SeeHelp" option from the main menu of the game. 2. The program creates another window with the user documentation provided in picture & text form in a scroll tab. 3. Player finishes observing the Help information and closes the window.
Entry condition	Player has opened the game and chosen the "See Help" option from the main menu.
Exit conditions	Player is provided with the user documentation.

Use case name	**SeeCredits**
Participating actors	Initiated by Player
Flow of events	1. Player chooses the “See Credits” option from the main menu of the game. 2. The program creates another window with the credits. 3. Player finishes looking at credits and closes the window.
Entry condition	Player has opened the game and chosen the “See Credits” option from the main menu.
Exit conditions	Player is provided with the credits.

Use case name	**ChangeSettings**
Participating actors	Initiated by Player
Flow of events	1. Player chooses the “Change Settings” option from the main menu of the game. 2. The program shows the available settings to the Player: volume & music 3. Player adjusts the settings according to their preference and chooses to save the settings OR chooses to go back to the main menu. 4. The program saves Player’s preferences if instructed. The program goes back to the main menu.
Entry condition	Player has opened the game and chosen the “Change Settings” option from the main menu.
Exit conditions	The changes have been made according to Player’s choice. Player has chosen to go back to the main menu.

Use case name	**StartGame**
Participating actors	Initiated by Player
Flow of events	1. Player chooses to “Start Game” after Game creation is complete. 2. The program sets all characters’ health points and weapons to default values, starts the timer and starts the game. 3. Player starts to play the game with their Hero. 4. At any point if the Player’s health points count reaches 0 OR if the timer finishes OR if all of the bots’ health points counts have reached 0 OR if Player chooses to manually end the game, the program ends the game. 5. The player is directed to the results screen to see their ranking.
Entry condition	Player has already created a game of their preference using the “Create New Game” system and has chosen to start the game.
Exit conditions	Game is over according to the games’ rules and the Player is provided with a results screen from which they can go back to the main menu.
Quality Requirements	At any point while the Player is playing the game with their hero, this use case can include the **EndGame** use case. The EndGame use case is initiated when the Player chooses the “End Game” option from the Game view. When invoked, the program manually ends the game and directs the Player to the results screen.

3.7 User Interface

3.7.1 Main Menu Screen

Figure 2 Main Menu Screen of Last Man

This is the main menu screen for our project. The user can click on buttons to navigate through the program.

3.7.2 Game Creation Screen

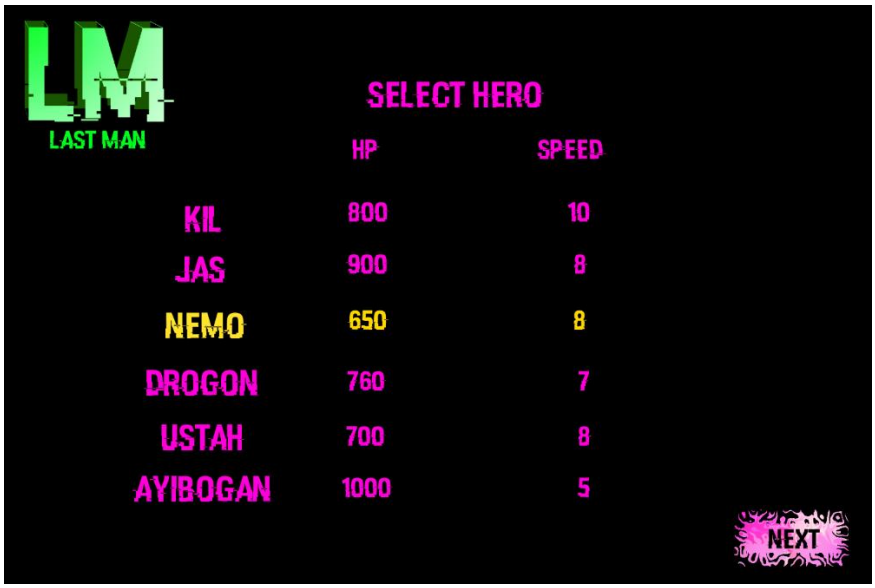

Figure 3 Hero Selection Screen

This is one example screen from the game creation process. The user can click on a hero and click next to select the hero.

3.7.3 Gameplay Screen

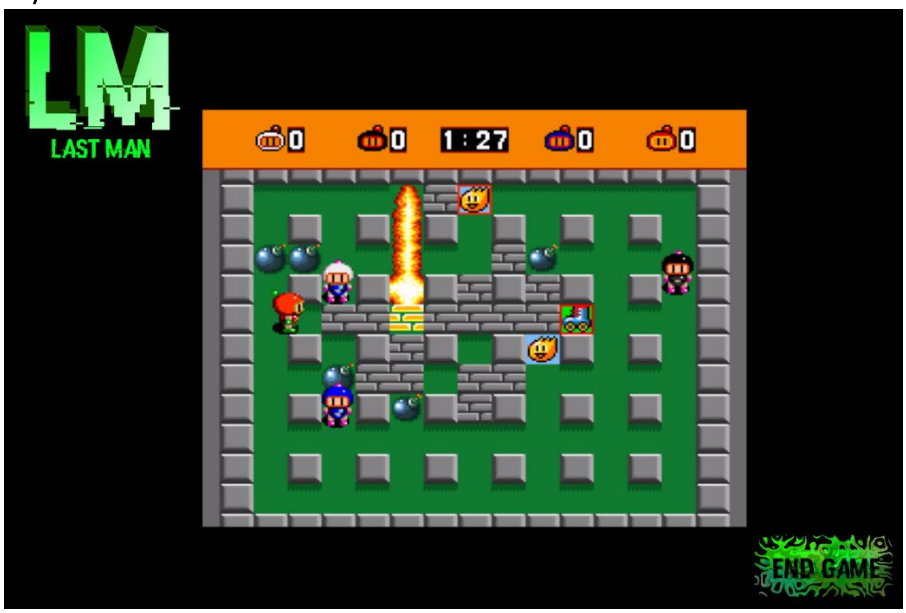

Figure 4 Gameplay Screen

The user navigates in the game play box via keyboard controls. The information on the bar in our program will be similar, except instead of zeros we will display HP counts[1].

3.7.4 Results Screen

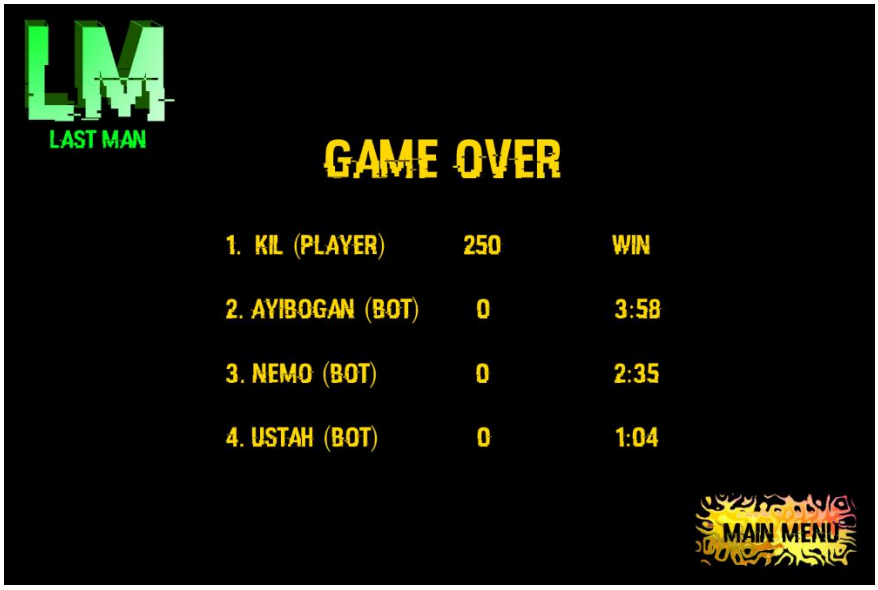

Figure 5 Results Screen

This is the basic results screen. It appears after game ends. It displays the characters' remaining HP by the end of the game, and their death times.

[1] The inner picture is taken from http://gamefabrique.com/games/super-bomberman/

3.7.5 Settings Screen

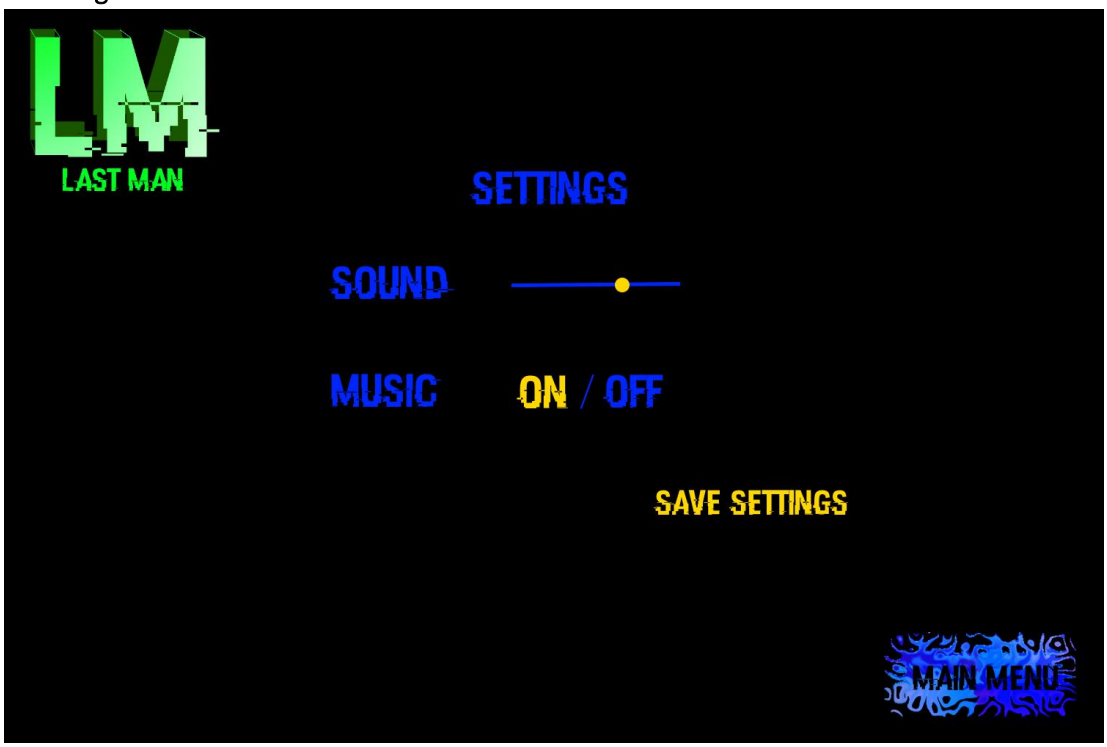

Figure 6 Setting Screen

This screen is for adjusting the game's settings. The user may choose to save the settings before going back to the main menu or not.

4 ANALYSIS

4.1 OBJECT MODEL

4.1.1 Class Diagram

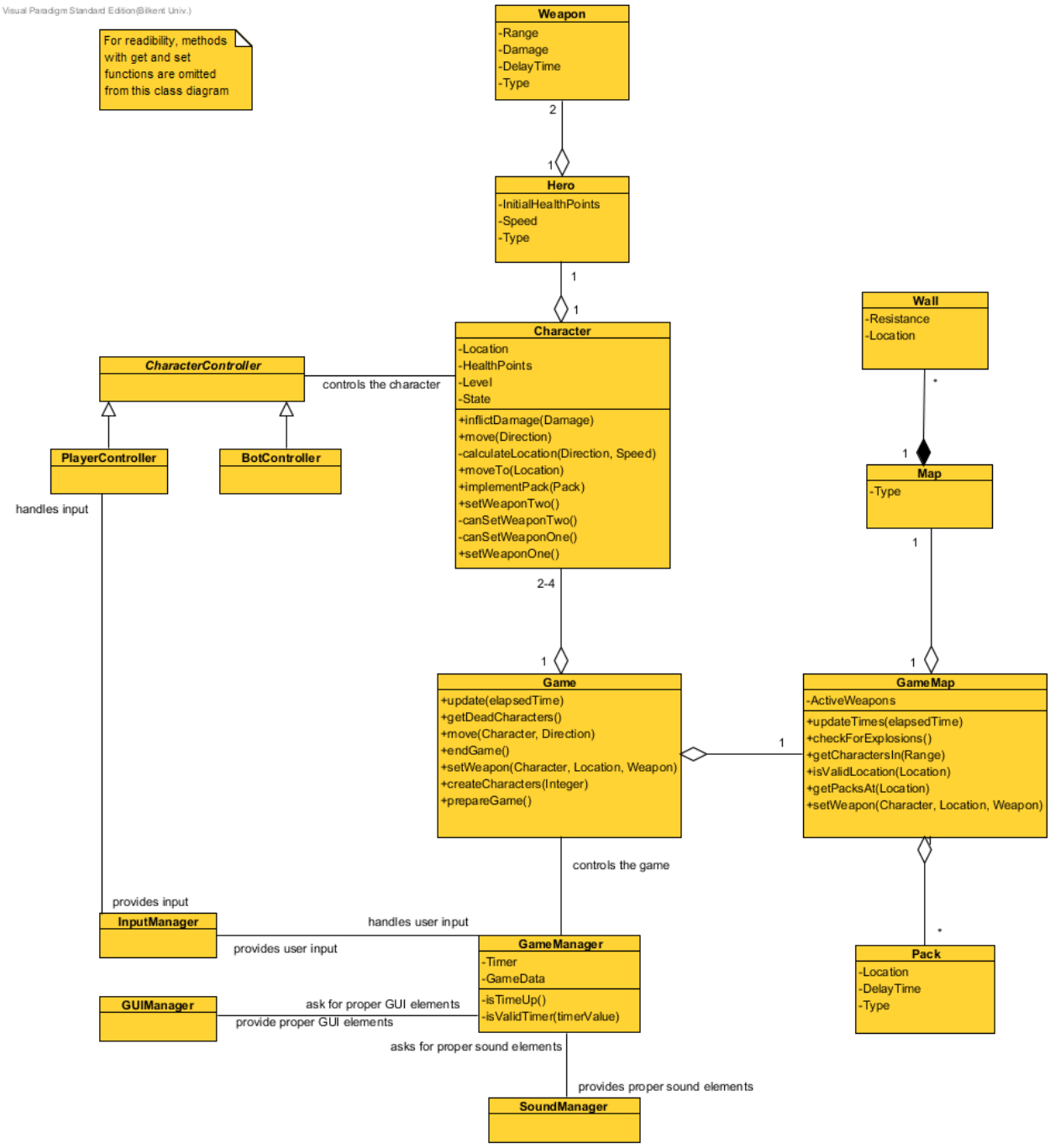

Figure 7 Class Diagram for "Last Man"

Above are the core functional classes for our project “Last Man”

4.1.2 Domain Lexicon

InputManager: This class will be used to provide the other classes with user inputs.

GUIManager: This class is responsible for the graphical user interface. This class will provide the views for the model classes shown above.

SoundManager: This class is responsible for providing appropriate sound elements.

GameManager: This class is one of the most important classes of the project. It is used to take user input to control the game. This class is responsible for creating and setting up the game, to organize the game dynamics during the game, to finish a game and to direct the Player to the Results Screen at the end of the game.

Game: This is the core model Game class of our project. A game has a GameMap and 2 to 4 Characters. Basic functionalities of this class include creating and managing the characters and the GameMap, updating itself as time progresses and preparing the game to start playing.

GameMap: This represents the active map the characters are playing on during the game. It has a list of active planted weapons. It has an underlying Map structure that determines the initial default state of the GameMap. It also has multiple Packs randomly planted or to be planted. Some functionalities of this class are updating the times of planted Game objects, checking for explosions, and managing the locations of the Game objects.

Map: This is the underlying map structure. The map has a type that serves as an ID, so that players will be able to choose them during the “Create New Game” stage. The map consists of walls.

Wall: This class represents a one unit-area sized block. It has a resistance value and a location on the map.

Pack: This class represents a pack, that is a power-up or power-down. It has a location on the map, a type that serves as an ID to determine the effect and a delay time that represents the maximum amount of time it can spend on the GameMap without being picked.

Character: This class represents all active players in the game, including both the Player (the user) and the bots. A character has a location on the GameMap, a health points count, and a level that determines its’ skill with picking up and using packs, and a state that determines whether it is allowed to use

weapons or not. (In the case of the Player, this is automatically set to hard since the Player can use the packs freely). A character has one hero and one controller.

CharacterController: This is an abstract controller class for the Character class.

- **PlayerController:** This is used to control the user's character.
- **Bot Controller:** This is used to control the bots' characters.

Hero: This represents the playing individual of a Character. A hero has an initial health points count, a speed level and a type that serves as an ID to distinguish from other heroes. A hero has two weapons. One weapon is common to all heroes, whereas the other weapon is special to each individual hero.

Weapon: A weapon has a range that represents how much area it will affect. It has a damage value that represents how much harm the weapon will cause and a type that serves as an ID. Its DelayTime attribute represents how often the weapon can be used if it's a special type weapon. If the weapons type is the default weapon (the bomb), the DelayTime attribute represents after how many seconds the bomb will set-off.

4.2 Dynamic Models

4.2.1 State Chart Diagrams

4.2.1.1 Game State Diagram

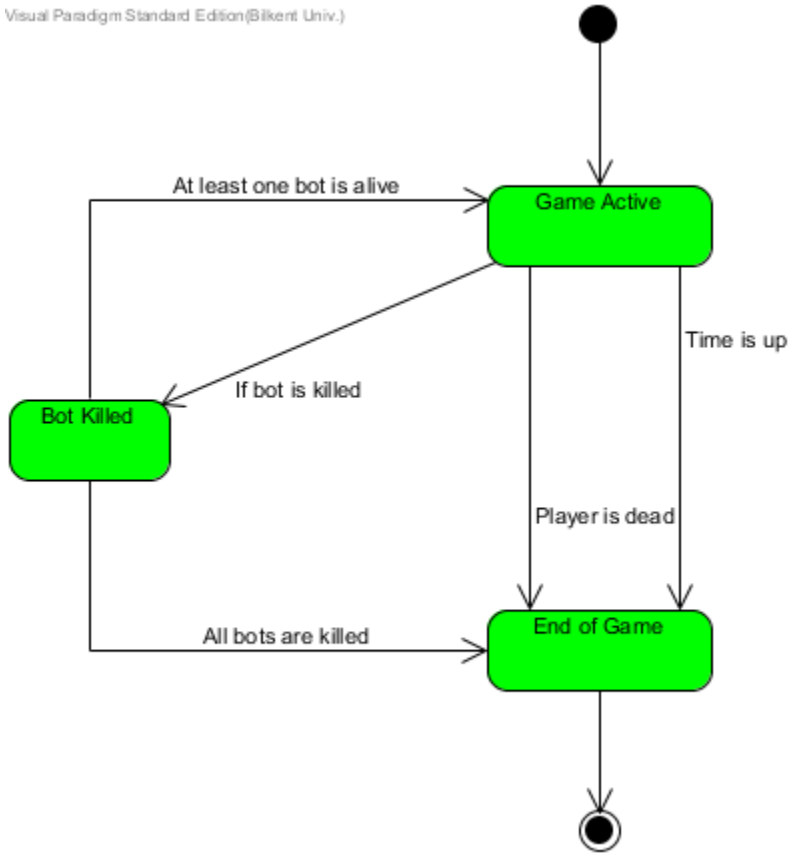

Figure 8 Game State Diagram

Last Man Survived starts with Game Active state and in this state characters can control their heroes. If player kills one of the bots, the game state is changed to Bot Killed. If all bots are killed, the state is changed to End of the Game state and game ends. When player kills one bot and there is at least one bot alive, the state would change back to the Game Active state again. On the other hand if player is killed, game immediately ends without waiting for the death of other bots. In addition to these, there is also another ending condition which is time. When time is up, the Game finishes and characters are sorted according to their current health points.

4.2.1.2 *Character State Diagram*

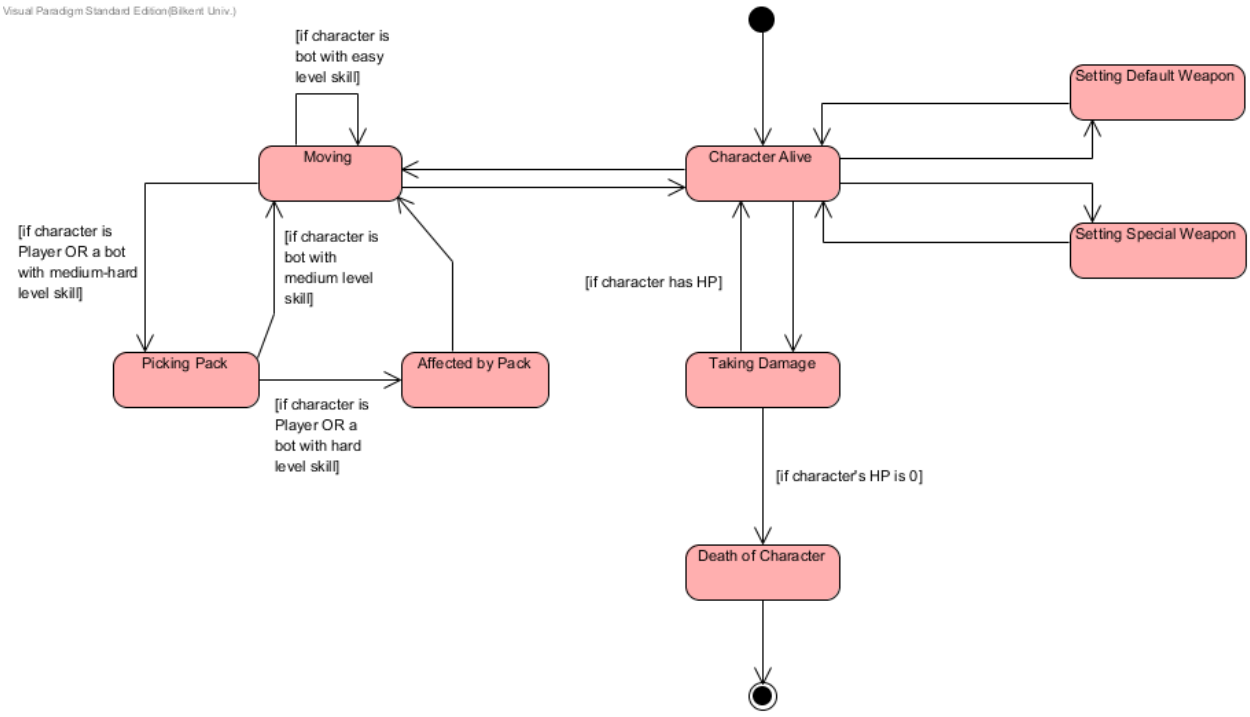

Figure 9 Character State Diagram

The character's default state is the Character Alive state. There are 4 states that the character can pass from this state. The character can set the default weapon and the special weapon. Then, state is changed to Character Alive again. In addition to using weapons, the character can move in the game map. While moving, the character comes across packs. If the character is a bot and the skill level of it is easy, the character stays in the moving state. Otherwise if the Bot's level is medium or hard or the character is the Player, the pack is picked. If the character is the Player or a Bot with the hard level, the pack affects the character. From the pack related states, the character goes back to the Moving state. If the character stops moving, it is returned to the Character Alive state. If character is affected by an explosion, the character is taken to the Taking Damage state. From this, if its' health points count has reached 0, the character is taken to the Death of Character state, otherwise Character returns to the Character Alive state.

4.2.2 Sequence Diagrams

4.2.2.1 Create Game

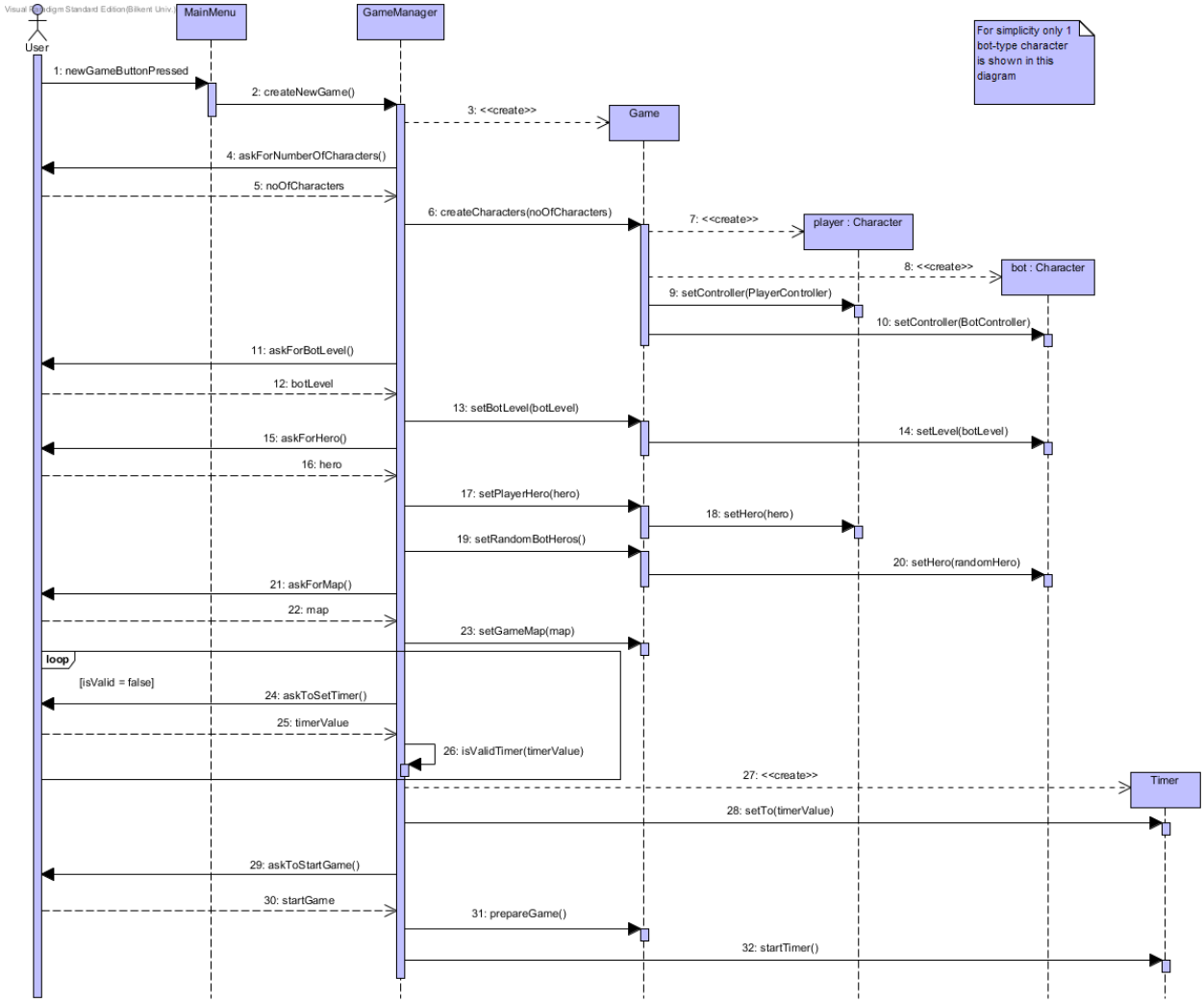

Figure 10 Create Game Sequence Diagram

In this sequence diagram, user interacts with GUI and presses the new game button which will create a request for GameManager to create a new game. Then GameManager sends the request which will ask the number of characters to the user, the value user enters is kept as an integer and the data will be sent to GameManager. With given data as a parameter GameManager tells Game to create characters, then Game creates characters as one Player and desired amount of Bots. After that, Game sets characters as the Player or as Bots by setting controllers of the created characters. GameManager asks the user to enter the level of the bot and takes the value from User. Game is requested to set the Bots' levels with the given value. Then Game assigns Bots' levels as the given botLevel. After assigning level of

the Bots, GameManager asks the user for hero selection and takes the selection data to send request to Game for setting the hero type with given value as parameter and sets hero for the Player. Hero types of Bots are assigned randomly so GameManager tells Game to set heroes for bots randomly. Then Game assigns random heroes to the Bots. Having hero selection done, map type should be selected so GameManager wants the user to select a map. The user returns map data to GameManager and GameManager asks Game to set map type using given data. GameManager tells the user to enter time value for the timer and checks if returned timerValue data is valid or not, repeats this operation until timerValue is a valid value. Then GameManager creates Timer and sets it to timerValue. With all required data taken and set in the Game, GameManager asks the user if they want to start the game. In this case it takes the startGame command and sends a request to Game to prepare the game. Lastly GameManager starts the timer and starts the game.

4.2.2.2 Character Movement & Pack Management

Scenario name moveAndPickPack

Participating actor instances lina: Player

Flow of events

1. Lina has opened the program, created a game and is currently playing the game.
2. Lina presses the up key from her keyboard.
3. The program calculates Lina's hero's new location considering her hero's speed and current location. The new location is in bounds and so the program updates the position of the hero. A pack exists in the new location and affects her hero.

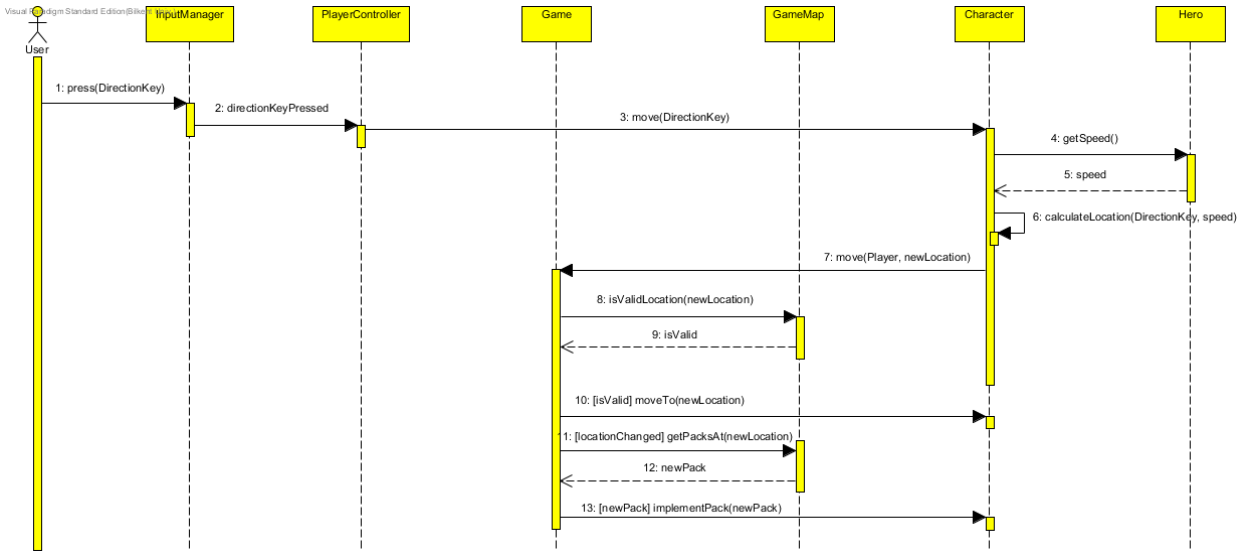

Figure 11 Move and Pack Sequence Diagram

This sequence diagram analyzes the movement and pack management of a character. It starts with the user pressing a direction key. When User presses a direction key, they tell InputManager to examine the key and warn PlayerController by telling a direction key is pressed. After that, PlayerController sends a request to Character to move the Player in the given direction. Since speed abilities of heroes vary, Character asks Hero for speed value and Hero returns the value. Character calculates the new location with the given direction and speed and makes a request to the game to move the Player to the new location. Game asks GameMap if given newLocation is a valid location or not and takes the answer in isValid stored as a boolean. If isValid is true, Game asks Character to move to the new location. If the location is changed, Game tells GameMap to check for packs in the newLocation. If there is a new pack found, Game tells Character to implement the pack.

4.2.2.3 Using Special Weapon

Scenario name useWeaponTwo

Participating actor instances oz: Player

Flow of events

1. Oz has opened the program, created a game and is currently playing the game.
2. Oz presses the ‘A’ key from his keyboard.
3. The program calculates if Oz is allowed to use another weapon. Oz is allowed to do so, so his hero’s special power is activated.

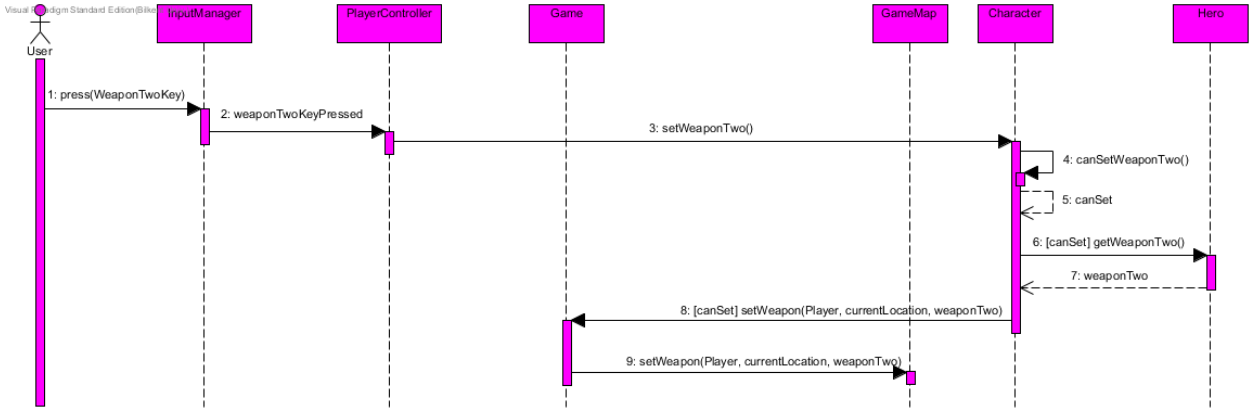

Figure 12 Use Special Weapon Sequence Diagram

Using Special Weapon sequence diagram simply explains how special weapon usage works. First, the user presses the WeaponTwoKey and sends information to InputManager. InputManager analyzes the request and tells PlayerController that weapon two key is pressed. PlayerController asks Character to set the desired weapon. For this, Character checks if it is allowed to use weapon two (weapons have delay times). Character checks its state and provides itself with a boolean canSet value. If canSet is true, Character gets its weaponTwo from its Hero and requests the game to set the weapon in its current location. Then Game sets and activates Player's weapon in the GameMap at Player's current location.

4.2.2.4 Updating Time & Explosions & Death

Scenario name oneSecondPasses

Participating actor instances gon: Player

Flow of events

1. Gon has opened the program, created a game and is currently playing the game.
2. One second passes in the game.
3. The game updates itself, a bomb near Gon explodes and Gon's character dies. The game ends.

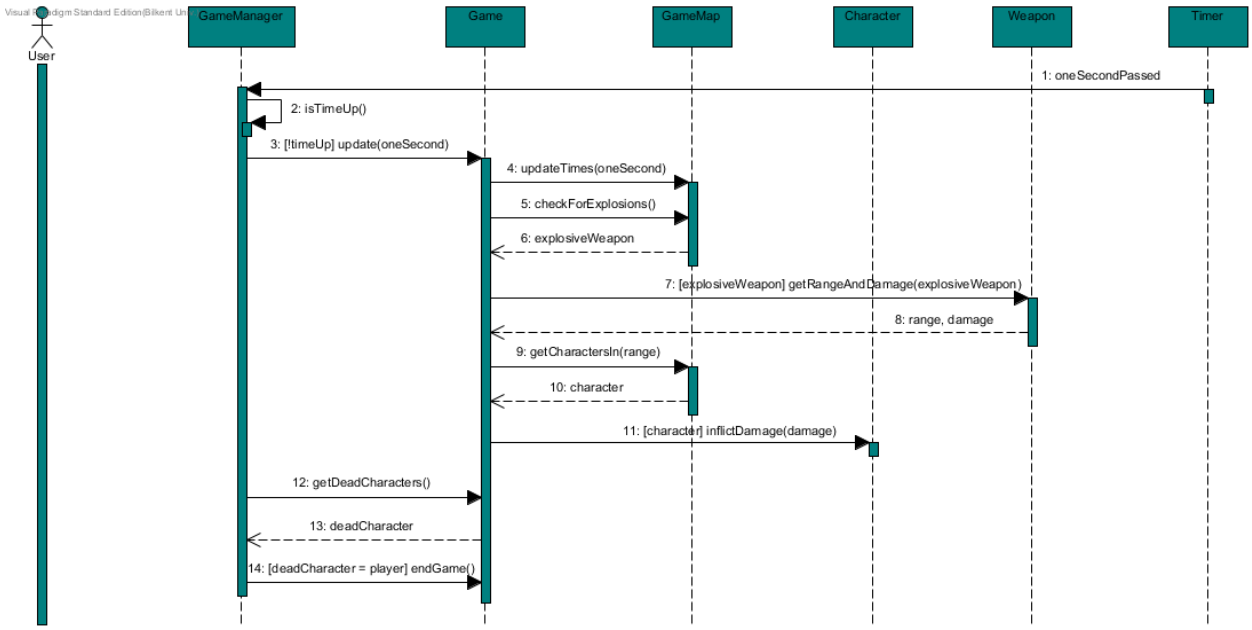

Figure 13 Time and Explosions Sequence Diagram

This sequence diagram starts with Timer warning GameManager by telling one second passed. GameManager checks if the time is up after that and if not, updates the Game as one second passed. Then Game updates times of game objects in the GameMap with the one second passed info. Then Game checks if there are any explosions. GameMap returns if the ready explosions. If there is one, Game wants the range and damage information of that explosive weapon from Weapon. Range and damage data are transferred to Game. Game asks the GameMap if there are any characters in the explosion range. GameMap returns the characters in the range to Game. If there is a returned character, Game tells Character to inflict damage to the character. After infliction of damage, GameManager asks game to return dead characters if there are any. Game returns if any. GameManager checks if the deadCharacter is Player, and if the condition is true, GameManager ends the game.

4.2.2.5 See Credits

Scenario name seeCredits

Participating actor instances gilbert: User

Flow of events

1. Gilbert has opened the program and chosen to see the credits.
2. He clicks on the “Credits” button from the main menu.
3. He observes the credits in the new window.
4. He closes the window.

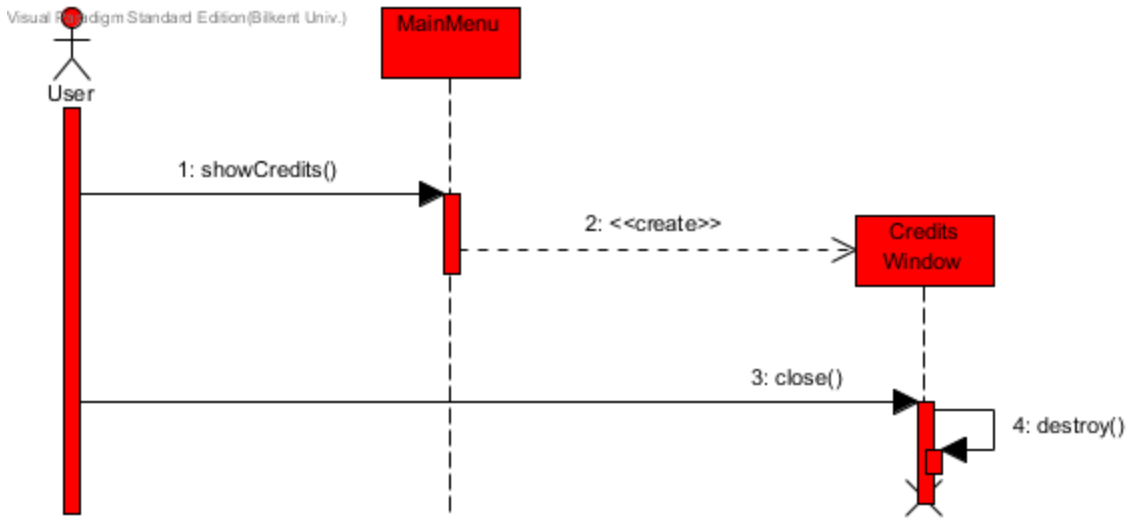

Figure 14 See Credits Sequence Diagram

See Credits sequence diagram basically provides credits info for user. First of all the user uses MainMenu GUI and requests from it to show credits. MainMenu creates a CreditsWindow. Then with User's attempt to close the CreditsWindow, window destroys itself and closes.

5 CONCLUSION

In this analysis report, we focused on the basic functionalities and design issues of our project. This helped us to consider the scope of our project and the methods we will use as well as the underlying programming structure.

In Requirements Specification, we tried to specify all the requirements for our project, both functional and non-functional. We also specified the design constraints for our project. It is our goal to fulfill the requirements we mentioned.

In the Use Case model, we wanted to explain the basic actions the user can take by writing scenarios and use case descriptions. We also made a class diagram for the fundamental objects in our project to describe their attributes and relations. We wanted to map the use cases we described to our objects using our sequence diagrams. We wanted to make the game states more clear so we prepared our state diagrams. We used VisualParadigm to create all of our diagrams.

We also aimed to keep our graphical user interface stylish, simple and easy to use.

We wanted to create a detailed and organized analysis report for both ourselves, so that we can handle the further steps of the project more easily, and for the reader, so they have a more concrete idea of our project.