CS 319 - Object-Oriented Software Engineering

Analysis Report

Nightmare Dungeon

Group 2-C

Mehmet Oğuz Göçmen

Berk Mandıracıoğlu

Hakan Sarp Aydemir

Hüseyin Emre Başar

Table of Contents

Table of Figures

1. Introduction

Our group have decided to design and implement a game called Nightmare Dungeon. In this game our character “Alice” is trapped in her dream and she tries to wake up by passing all the layers of her dream by facing off against her imagination.

This game is going to be a 2D dungeon crawler/rogue-like game, and it is going to be developed using JAVA.

2. Game Overview

Nightmare Dungeon is going to be a 2D dungeon crawler/rogue-like game. This means that players will collect passive and active items which contribute to Alice's stats, attack animation or appearance. Alice will have "x" stats called "Attack Damage", "Attack Speed", "Movement Speed", "Health", "Size" and etc. The passive items will give positive or negative effects to these stats and they may change Alice's attack animation, appearance. For example, at the beginning of the game Alice will be shooting "tears", but after player finds an item she may start to shoot "flaming tears". At the start of the game character is placed at the beginning of a dungeon which is called "dream" in our game. Main purpose of game is to save Alice from her dream by collecting items, getting stronger and beating her imaginary creatures. Player will shoot at creatures to kill them. There will be various creatures with different strength. Don't forget! As it is a nightmare, there will be a lot of absurdity in game!

2.1 Gameplay

Player is going to use "W" for going up "S" for going down, "A" for going left, "D" for going right on the screen. Again player is going to use arrow keys for shooting. Character will shoot on the arrow keys direction. For example, when character is not moving and player pushes the left arrow key, projectile will go left, but for example if player pushes the left and up arrow key at the same time, projectile will go top left corner. "E" button is going to be used for planting bombs for destroying obstacles on characters' way. "Space" button is going to be used for using active items. For changing room, player is just going to move Alice to the door. Players are able to pause game by pressing "P" button. There will be a mini-map at the right top of the screen for showing players where

they are but players have to explore the room to see it in the mini-map, in other words it is similar to fog of war.

2.2 Levels

In this game, there will be “x” layers of Alice’s dream and every layer will have “x to y” rooms in it. Every layer will have different concept. In every layer, there will be an item room for Alice to pick an item if she wants. There will be bosses at the end of every layer. These bosses will be made from Alice’s biggest fears in life for example her Math Teacher. Every boss will have different types of attack style. Players will need different tactics in every boss. After beating the bosses, there will be a gate on floor and Alice will be headed to the next layer of her dream. Obstacles and creatures will spawn randomly according to some boundaries. For example neither obstacles nor creatures are not going to spawn at the entrance of a room. Item drop is going to be random, but not totally random. Stronger items will have less chance to drop.

2.3 List of Items & Power-ups

- **Passive Items**

-Magic Mushroom; Never eat things that you don’t know, this is an exception! Attack damage and speed up! Projectile size down.

-Moms Underwear; Sometimes we can see things that we don’t want to see. Attack speed and range down by x!

-Poop; new way of attacking! Instead of tears, she starts firing poop. Damage up!

-Steroid Rage; Alice makes herself stronger but vulnerable. Health down, attack speed and damage up.

-Pentagram; A deal with the Satan. Increases damage.

-Lab Assignment; Student's nightmare, takes the joy of living. Attack and movement speed down!

-WWA BELT; Every little boys' dream, winning WWA belt. Attack damage and size up!

- A Pack of Cigarette; Alice has to stop smoking; she can't run anymore. Speed down!

-Sziget Ticket; All stats up!

-Growth Hormone; Size up!

-Dog Food; Yummy! Increases maximum health.

-Lucifer Head; Alice's attacks become laser now.

-Binky; Good times… Size down, speed up.

-Inner Eye; Now, firing 3 tears at a time.

-Holy Mantle; Refreshes in every room. Allows Alice to dodge one attack per room.

-Basys3; Speed down!

-Mom's Razor; It's time to shave. Attack damage up.

-Heart; If health is missing, it recovers.

-Dead Cat; Oh no! Alice's cat is dead. Damage up.

-Lighter; Now, Alice fires flaming tears!

-Reverse Engineering; Alice's head and butt swaps location.

-Forever Alone; Now Alice's attacks have chance to fear enemies.

-Tick; It stuck and it sucks Alice's blood. Unable to pick active item anymore.

-Sweet Love; Projectile size up.

-Goat Head; God accepts Alice's offering.

-Bomb; Now Alice can plant a bomb to destroy obstacles.

-Desert Eagle; Now Alice attacks with pistol.

- **Active Items**

-Dice; It allows you to reroll all of your items.

-Dragon Breathe; Blows a strong flame breathe.

-Mom's Slipper; Mass fear to the creatures in the room.

-Perfume; Creatures fell in love with Alice's smell. 2 seconds stun.

-Transcript; When used Alice looks at her transcript and sees F and fires a 4 way laser.

-Music Box; Alice opens a cursed music box. Mass room damage.

3. Requirements

3.1 Functional Requirements

- Player should be able to control the character with keyboard.
- Player should be able to pause the game.
    1. Continuing game.
    2. Changing settings.
    3. Quitting game.
- Player should be able to choose settings
    1. Changing the volume of background music.
    2. Changing the volume of interaction sounds.
- The game should include different attributes that makes it interesting.
    1. Having different upgrades for the player.
    2. Having various levels.
    3. Having diverse enemies and bosses.
- User should be able to see high scores of players.
- User should be able to see help which includes information of the game's mechanics.
- User should be able to see information about credits.

3.2 Non-Functional Requirements

- Graphics should show that Alice is in a nightmare to increase immersion.
- The game should run smoothly to offer quality gameplay.
- The controls should be responsive since the game is fast paced.

- All levels should allow the player to progress, meaning that randomly generated obstacles shouldn’t block the paths to other rooms.

3.3 Pseudo-Functional Requirements

- The game will be implemented in java.
- JavaFX will be used for UI.

4. System Models

4.1 Use Case Model

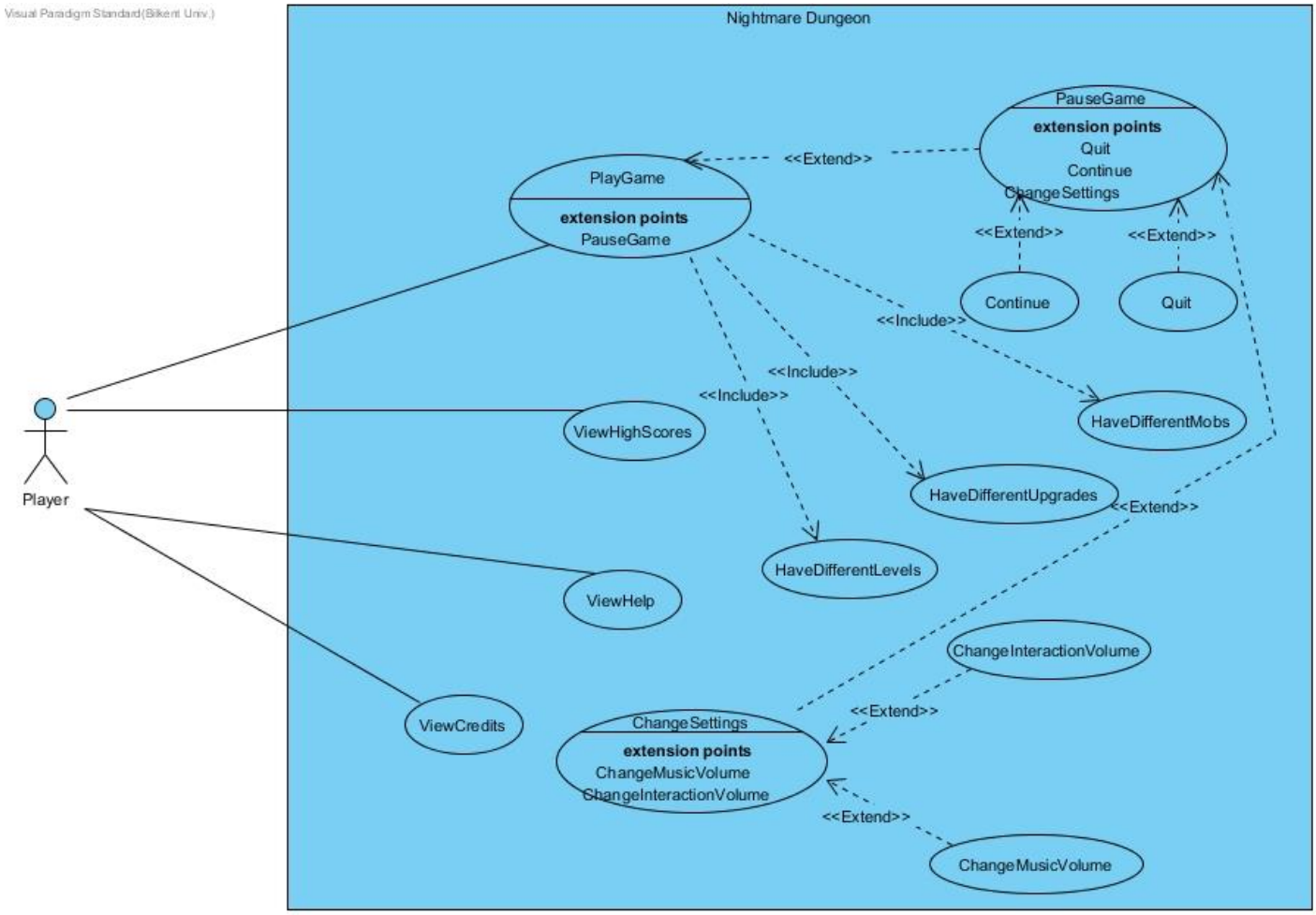


Figure 4.1.1. Use Case Diagram of the System

4.1.1. Use Case Descriptions

4.1.1 Play Game

Use case name: PlayGame

Participating actors: Player

Entry condition: Player is on the main menu.

Exit condition:

- Player has defeated the final boss and won the game, OR
- Player ran out of lives and game is over, OR
- Player chose “Quit” from the pause menu.

Main flow of events:

1. Player starts the game.
2. Levels are randomly generated by the system.
3. Player starts from the neutral room without monsters or obstacles.
4. Player proceeds through the floors.
5. Player defeats the final boss.
6. Score of the player is displayed on the screen and if the score is in the top 10, game asks for the player’s name to save it to the top 10 list.
7. Player returns to main menu.

Alternative flow of events:

1. Player loses all his/her hearts and the game is over, player returns to the main menu.
2. Player exits the game with his own will.

4.1.2 View Help

Use case name: ViewHelp

Participating actors: Player

Entry condition: Player is on the main menu.

Exit condition: Player returns to main menu.

Main flow of events:

1. Player clicks on “View Help” on main menu.
2. Help is displayed.
3. Player chooses to return to the main menu.

4.1.3 View High Scores

Use case name: ViewHighScores

Participating actors: Player

Entry condition: Player is on the main menu.

Exit condition: Player returns to main menu.

Main flow of events:

1. Player clicks on “View High Scores” on main menu.
2. High scores and the names of the owners of high scores are displayed.
3. Player chooses to return to the main menu.

4.1.4 View Credits

Use case name: ViewCredits

Participating actors: Player

Entry condition: Player is on the main menu.

Exit condition: Player returns to main menu.

Main flow of events:

1. Player clicks on “View Credits” on main menu.
2. Names of the developers and their contributions to the game are displayed.
3. Player chooses to return to the main menu.

4.1.5 Pause Game

Use case name: PauseGame

Participating actors: Player

Entry condition: Player is playing the game.

Exit condition: Player returns to playing the game.

Main flow of events:

1. Player presses pause button while playing the game.
2. Pause menu is displayed.
3. Player continues playing the game.

Alternative flow of events:

- Players chooses to change the settings and goes to settings menu.
- Player quits the game.

4.1.6 Quit Game

Use case name: Quit

Participating actors: Player

Entry condition: Player has paused the game.

Exit condition: Player quits the game.

Main flow of events:

1. Player presses “quit”.
2. Player quits the game.

4.1.7 Change settings

Use case name: ChangeSettings

Participating actors: Player

Entry condition: Player has paused the game.

Exit condition: Player returns to pause screen.

Main flow of events:

1. User changes the interaction volumes or the music volume.

2. User goes back to pause screen.

4.1.8 Continue Game

Use case name: Continue

Participating actors: Player

Entry condition: Player has paused the game.

Exit condition: Player returns to game.

Main flow of events:

1. User presses continue.
2. User goes back to playing the game.

4.1.9 Change Music Volume

Use case name: ChangeMusicVolume

Participating actors: Player

Entry condition: Player is in the settings menu.

Exit condition: Player returns to settings menu.

Main flow of events:

1. User adjusts the music volume.
2. User goes back settings menu.

4.1.10 Change Interaction Volume

Use case name: ChangeInteractionVolume

Participating actors: Player

Entry condition: Player is in the settings menu.

Exit condition: Player returns to settings menu.

Main flow of events:

1. User adjusts the interaction volume.
2. User goes back settings menu.

4.1.11 Have Different Upgrades

Use case name: HaveDifferentUpgrades

Participating actors: Player

Entry condition: Player is playing the game.

Exit condition: Player exits the game.

Main flow of events:

1. The game offers various upgrades to the player.

4.1.12 Have Different Mobs

Use case name: HaveDifferentMobs

Participating actors: Player

Entry condition: Player is playing the game.

Exit condition: Player exits the game.

Main flow of events:

1. The game has different mob types.

4.1.13 Have Different Levels

Use case name: HaveDİfferentLevels

Participating actors: Player

Entry condition: Player is playing the game.

Exit condition: Player exits the game.

Main flow of events: 1. The game has different level layouts.

4.2. Dynamic Models

4.2.1. Sequence Diagrams

Scenario Name: Change Settings

Scenario: Oğuz wants to change the game's settings. He selects the "Change Settings" icon from either the main menu or the pause menu. The system displays 2 settings: Adjust interaction voice and adjust music voice. Oğuz then selects the setting that he desires to change and changes it. Oğuz then exits the settings window and the system updates the changes that were made.

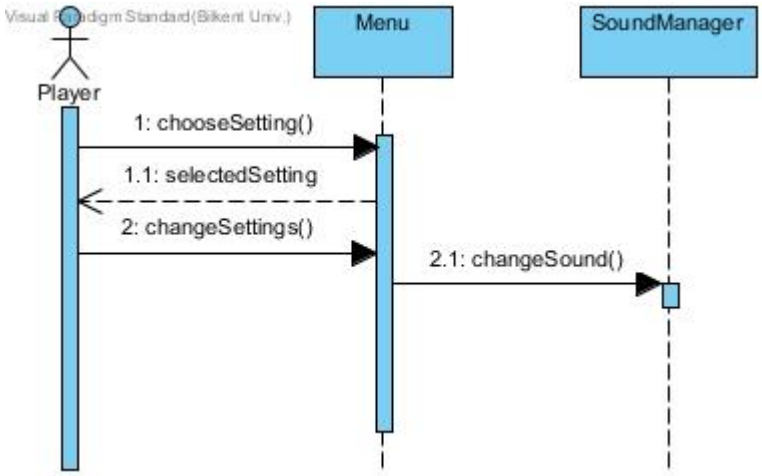


Figure 4.2.1.1 Sequence Diagram of Change Settings Scenario

Scenario Name: Move Player

Scenario: Oğuz needs to move the Alice character to progress through the game. Oğuz presses the arrow keys to move the Alice character up, down, left or right. Alice moves as Oğuz directs her.

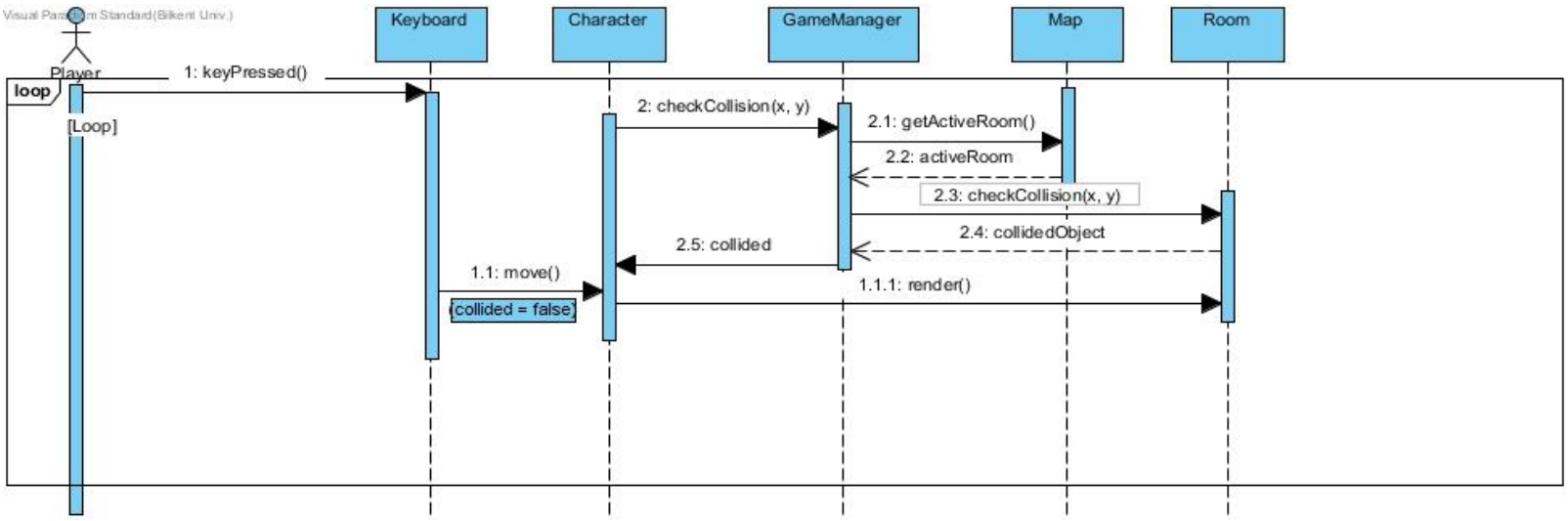


Figure 4.2.1.2 Sequence Diagram of Move Player Scenario

Scenario Name: Player Attack

Scenario: Oğuz wants to attack to a minion or a boss. He presses WASD keys on keyboard in order to give direction to the projectile. When he clicks on these buttons player shoots immediately.

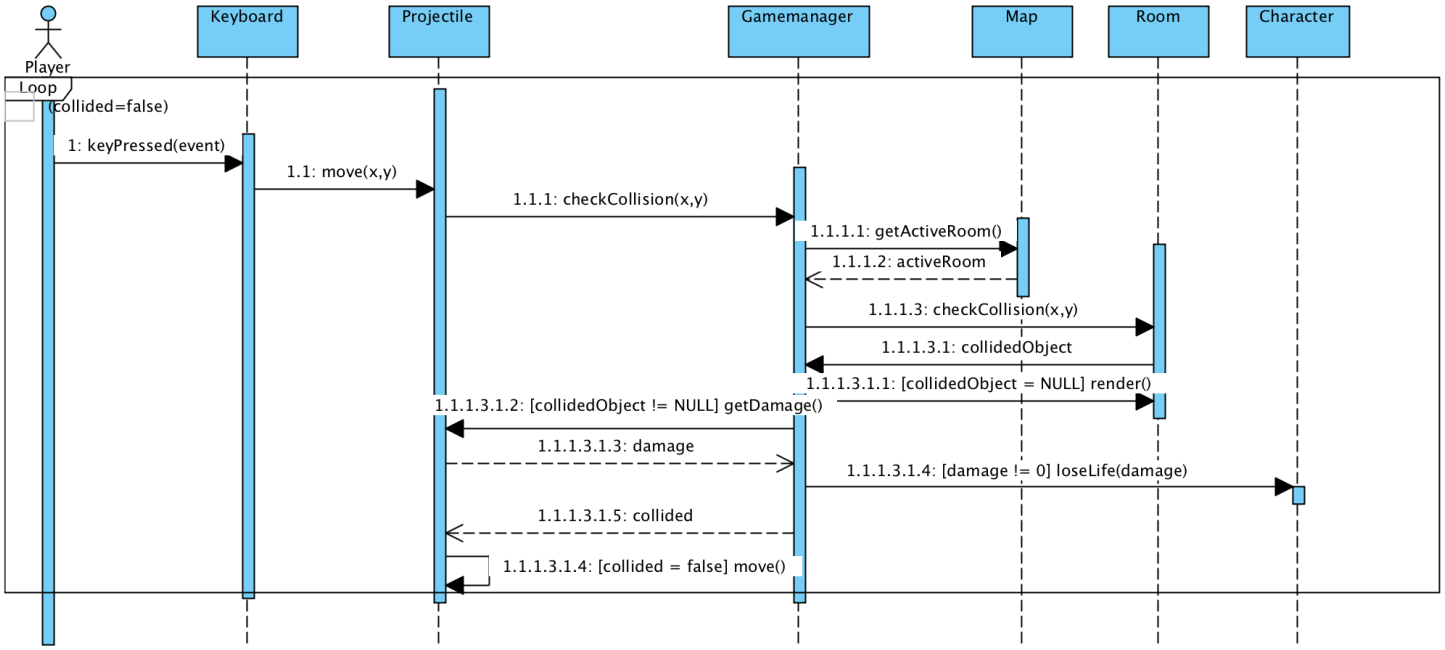


Figure 4.2.1.3 Sequence Diagram of Player Attack Scenario

Scenario Name: Monster Attack

Scenario: Monster wants to attack to our player. It detects the location of Alice and then it shoots a projectile in its direction.

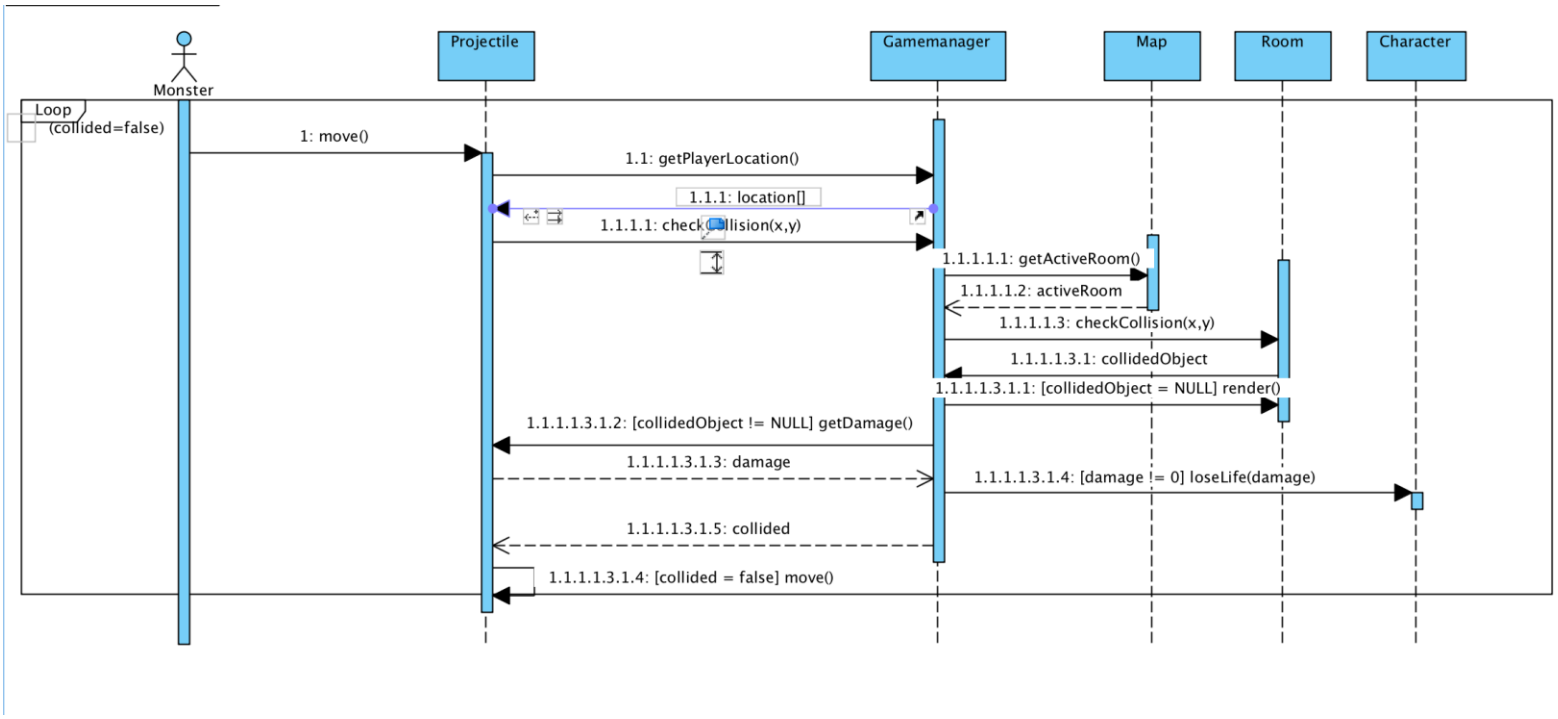


Figure 4.2.1.4 Sequence Diagram of Monster Attack Scenario

Scenario Name: Monster Move

Scenario: Monster wants to move in room. It selects a direction the location of Alice and then it shoots a projectile in its direction.

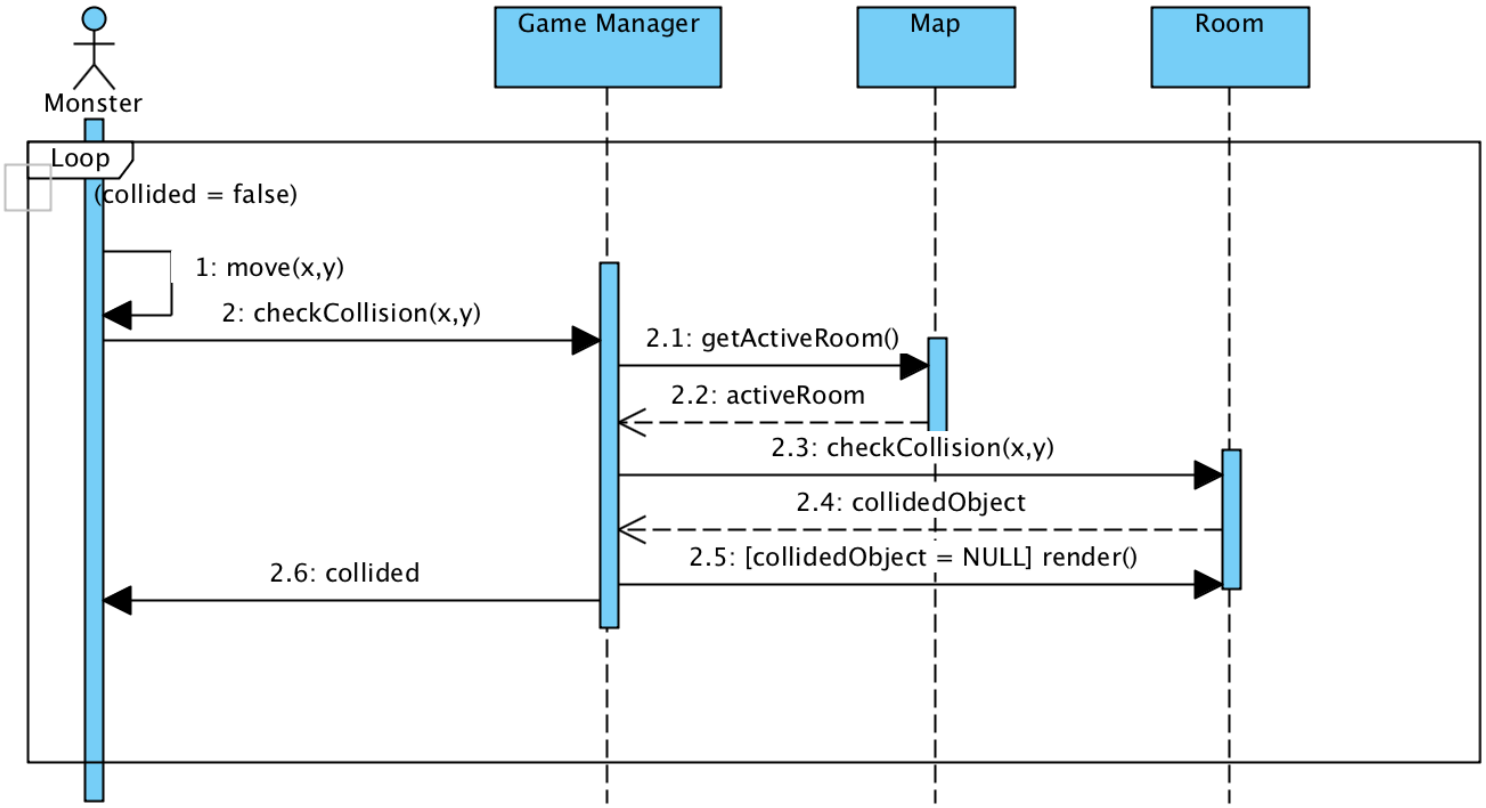
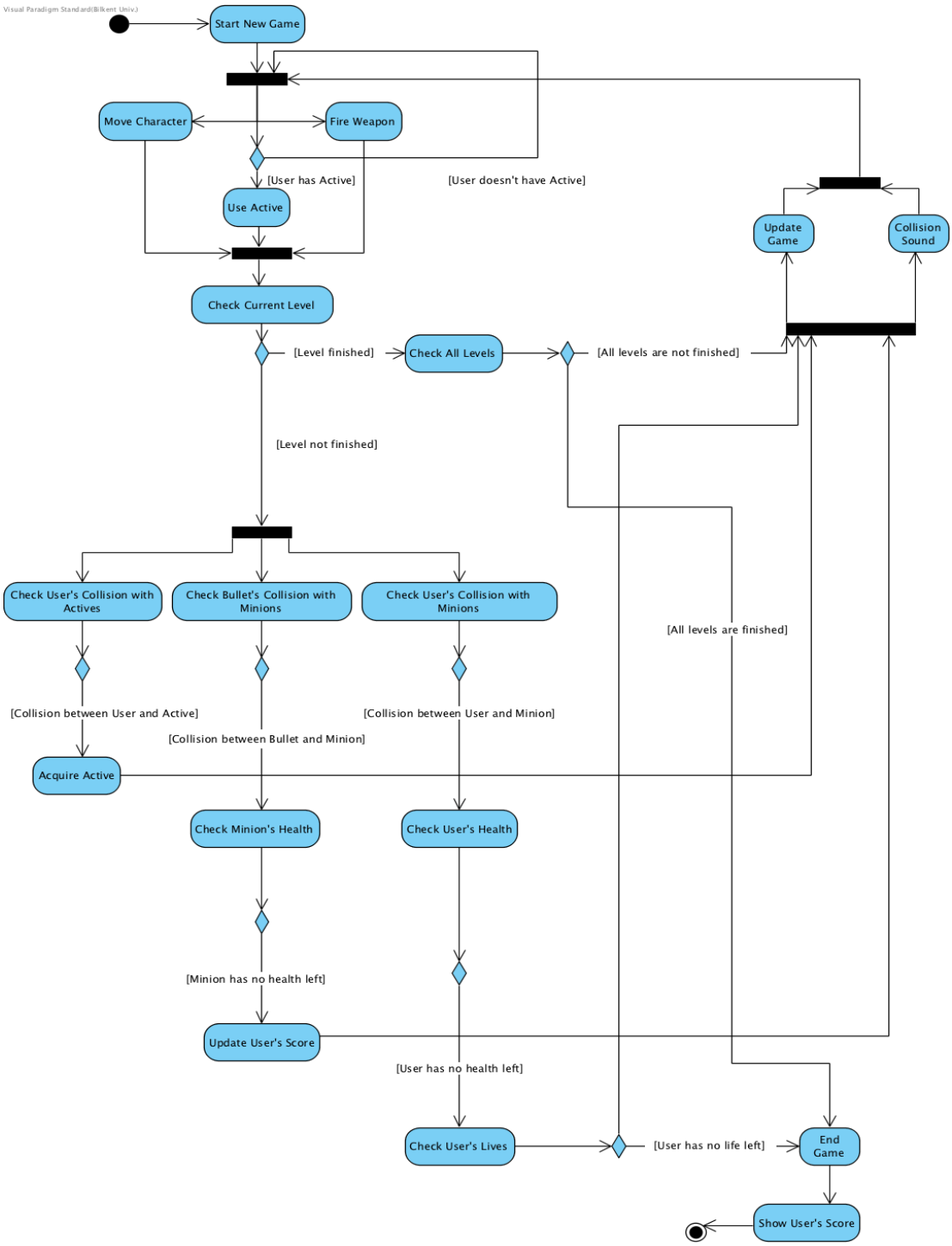


Figure 4.2.1.5 Sequence Diagram of Monster Move Scenario

4.2.2. Activity & State Diagram



This diagram shows how system maintains gameplay.

Figure 4.2.2.1. Activity Diagram of the System

Activity diagram given in figure 4.2.2.1 shows the main flow in game. Each time user presses “Play button”, game starts running and relevant components (map, user character, minions, obstacles, stat indicators etc.) are rendered to frame. After the process finishes, listeners will listen to keystrokes of user which will indicate whether user pressed movement keys, firing keys and active power keys. Additionally, if user presses active power key, game will check whether user has active power or not and act accordingly.

When a keystroke is detected, system will check if current level is proceeding or it’s over. If current level is over, system will check all levels for completeness. If all levels are completed, game will be over, else game will proceed and objects will be updated. If current level is not over, system will check interactions between main objects simultaneously. 3 processes will be checked concurrently.

System will check the collision between the user and active powers which will appear in map randomly. If collision is detected, user will be awarded with an active power.

System will check the collision between the bullets that user fires and minion. If collision is detected, system will check whether minion has health or not. If minion has no health, user will gain more score and update the score.

System will check the collision between user and minion. If collision is detected, user’s health will be checked. If user has no health left, user’s lives will be checked. If user has no life left to play, game will end. Otherwise user will start the map again.

In all other cases, game will update itself every time and collision sounds will be given. When user’s lives are ended or all levels are finished, game will end and system will show user’s score.

State Diagram

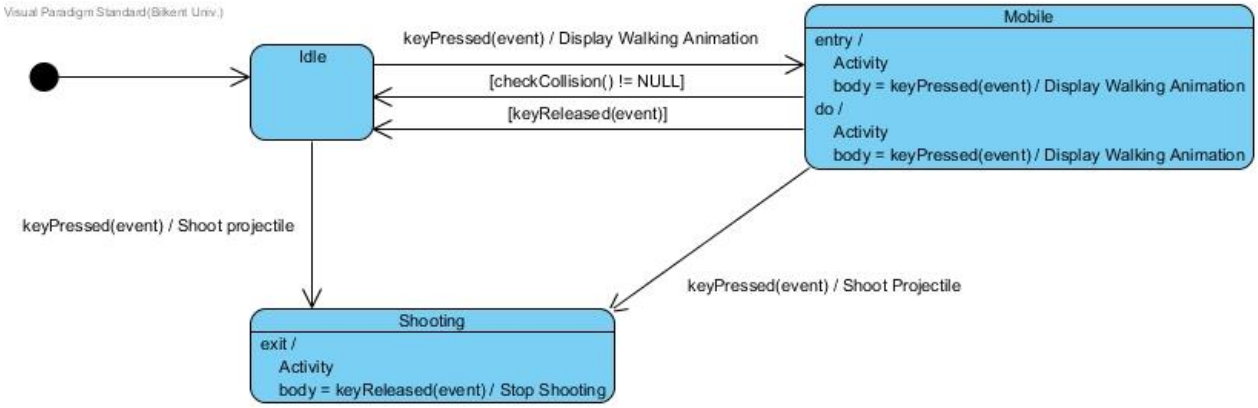


Figure 4.2.2.2. State Diagram of User in Game

Figure 4.2.2.2. shows the state diagram of a user. User has three states: idle , mobile and shooting. User is initially idle, when an arrow key is pressed user changes to mobile state. User stays at mobile state until key is released and turns back to idle state. In mobile state player moves and the walking animation appears on screen. Moreover, User can go into shooting state from both idle and mobile states. User stays in shooting state until a key is released.

4.3. Object and Class Model

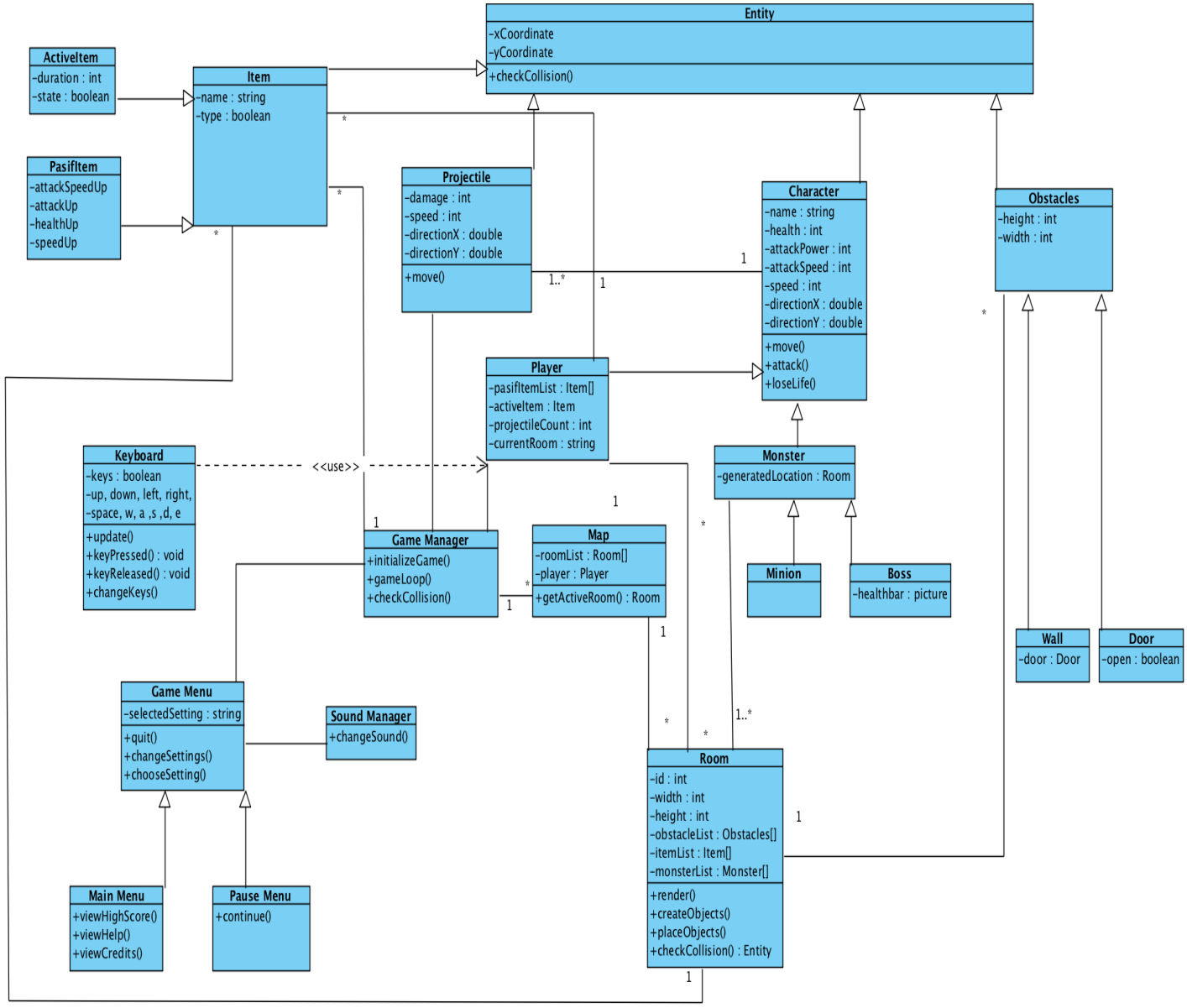
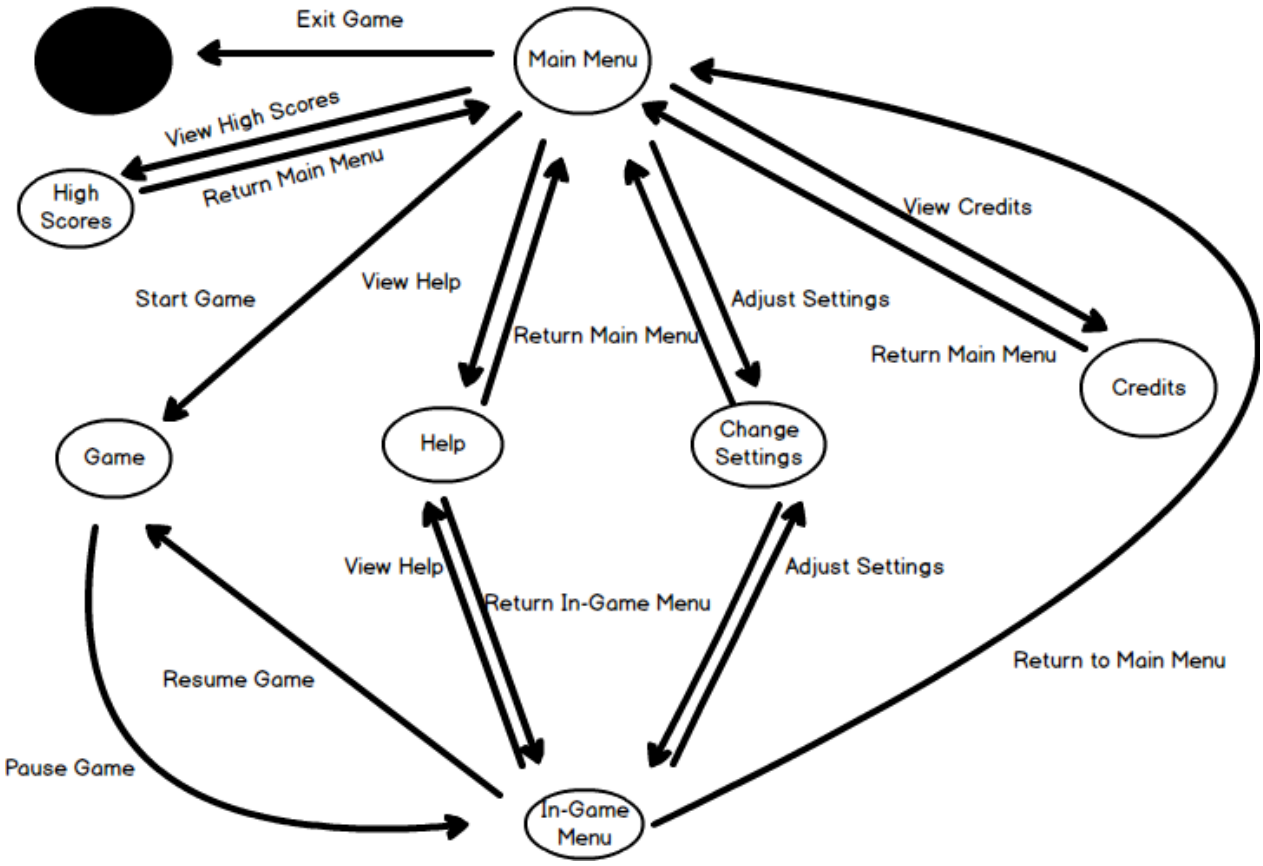


Figure 4.3.1. Class Diagram of the System

Class diagram of our game consists of several inheritances. It represents the general structure of our game. It is revised many times in the process of making Sequence, Activity and State diagrams. Class descriptions are as follows:

- **Entity** class is the parent class of Character, Item, Obstacles, Projectile classes. It has the basic positions of the characters and enables to check for any collisions.
- **Character** class is the parent class of Monster and Player classes. It has the basic attributes(name, health, attack power, etc.) and basic operations(move, attack, take damage). It also has association with projectile class which makes character shoot.
- **Player** class can have active item and passive items and it can have more than one projectiles different than monsters. It is the class of main character Alice.
- **Monster** class has specific spawn room attribute different than player.
- **Game Manager** class is the class that manages all the mechanics of the game. It has association with many of the classes and it is the most necessary one.
- **Keyboard** class is the input manager type of class which detects key press and release events. Player class uses Keyboard class to have an interaction with the game.
- **Map** class is the class that includes the rooms in the game.
- **Room** class is the class that generates the entities inside of it. It has association with the Game Manager class to detect collisions.
- **Game Menu** class s the parent of Pause and Main menus, it has the basic sett,ngs operatins.

4.4. User Interface – Navigational Paths and Screen Mock-ups



4.4.1. Navigational Paths

Figure 4.4.1.1. Navigational Path of the game

4.4.2. Mockups

Main menu: When application starts running, user will encounter the Main Menu frame in which there will be 5 buttons to press. This buttons are: Play, Settings, Help, Credits, Exit. Mockup of the Main Menu frame is given in figure 4.4.2.1.

Figure 4.4.2.1. Mockup of the Main Menu frame

Play: Play button will signal game to start with default settings from the first level. Mockup of the game itself is given in figure 4.4.2.2.

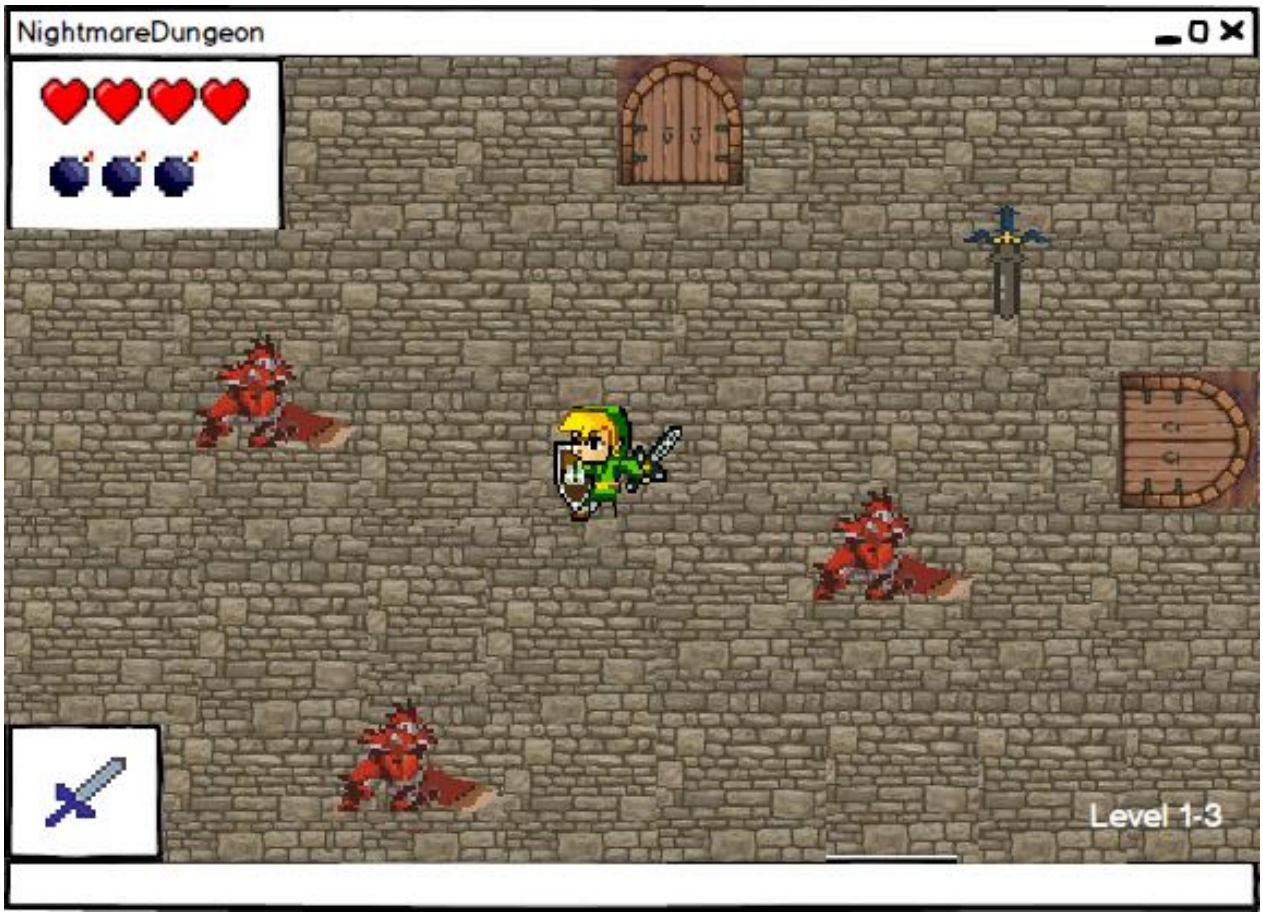


Figure 4.4.2.2. Mockup of game itself

Settings: Settings button will enable the transition to settings frame. This frame includes some optional settings relevant to gameplay. User will be able to turn on/off in-game effect sounds, music sound etc. This settings frame will also be available in in-game menu in case of an unpleasant experience of the game. Mockup of the Settings frame is given in figure 4.4.2.3.

Figure 4.4.2.3. Mockup of Settings Frame

High Scores: High Scores will open the High Scores Frame which will display the first 10 scores of the game. If player gets a score in this span, user's score will be added to this frame.

Figure 4.4.2.4. Mockup of High Scores Frame

Help: Help button will open the Help frame which includes the controls of the game, and upgrades that will come throughout the game. Controls include "WASD" keys to move, arrow keys to fire at minions. Gameplay will include relevant information about the game (not all of them!!!). Help frame will have buttons called "gameplay" and "controls" which will enable to transition within each other incase user wants to see them again before starting. There will also be

back button to go back to menu. Mockups of the Help frames are given in figure 4.4.2.4.1 and 4.4.2.4.2.

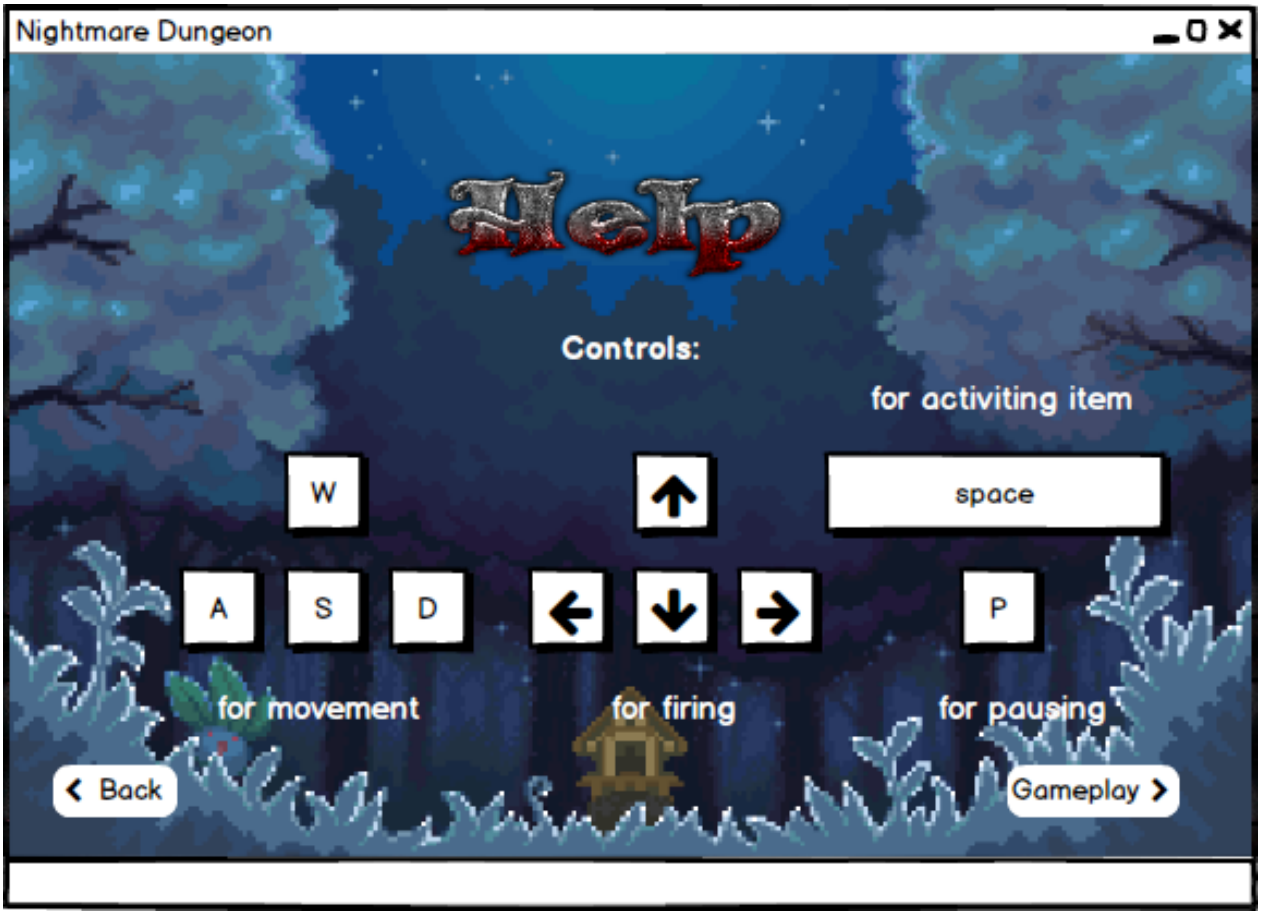


Figure 4.4.2.5.1 Mockup of Help frame which includes Controls

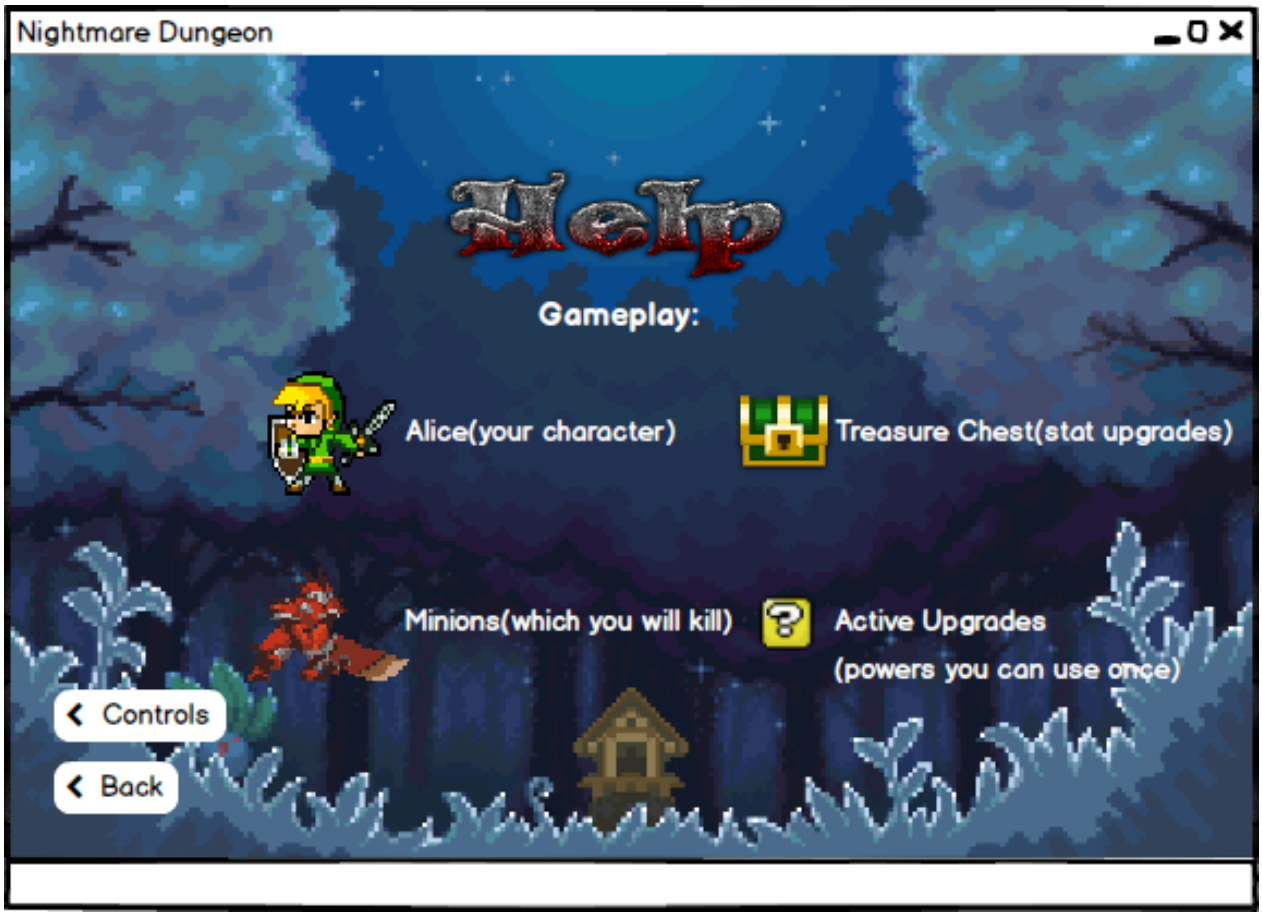


Figure 4.4.2.5.2 Mockup of Help frame which includes Gameplay

Credits: Credits button will open the Credits frame to show user the makers of the game. Back button will enable to go back to menu. Mockup of the Credits frame is given in figure 4.4.2.5.

Exit: Exit button is to shut down the game.

Figure 4.4.2.6. Mock-up of Credits Frame

In-Game Menu: Throughout the game, user will be able to pause the game and resume later by pressing P key. When user presses the key, in-game pause menu will appear as an overlay. This menu will include 4 buttons: Resume, Settings, Help, Return to Main Menu. Resume button will resume the game from the moment it had been paused. Settings and Help button will bring up the frames Settings and Help whose mockups are given before in figures …. Return to Main Menu button will direct the user to Main Menu and the progress of the game will be lost as well. Mockup of In-Game Menu is given in figure 4.4.2.7.

Figure 4.4.2.7. Mock-up of In-Game Menu Frame

5. References

- https://cooltext.com/
- http://store.steampowered.com/app/113200/The_Binding_of_Isaac/
- Images are found from Google Images (https://www.google.com.tr/imghp?hl=en&tab=wi).