Bilkent University

CS319 Project / Group: 2B

Endless Dungeon: Progressive Dungeon Crawler RPG

Analysis Report

Can Özgürel, Alperen Kaya, Alemdar Salmoor, Batuhan Erarslan

Supervisor: Uğur Doğrusöz
TA: Hasan Balcı

Iteration 1 Analysis Report
Feb 17, 2018

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Object-Oriented Software Engineering course CS319.

Contents

Analysis Report

Endless Dungeon: Progressive Dungeon Crawler RPG

1 Introduction

In Endless Dungeon, you will follow the path of an adventurer and see through his eyes as he decimates his foes. With different classes and several advantages and disadvantages that come with these classes, you will have many ways to (try to) survive the Endless Dungeon. You may find certain items to help you on your journey, but will they be enough to keep you alive? You may feel stronger with each enemy that you kill, but for how long will you be able to keep killing them?

The Endless Dungeon awaits, with endless opportunities as well as endless dangers. Are you up for the challenge?

2 Proposed System

2.1 Overview

- A retro FPS perspective will be used that shows the enemies as seen by the character face on
- Enemies will attack in waves and waves will consist of 3 enemies
- Every fifth wave will be a boss wave and the enemies on the two sides will be led by a boss enemy
- Maps will change as the player progresses further
- After beating every twentieth wave the player will transition to a new map with harder enemies
- Players will be able to improve their characters by slaying enemies and gaining experience
- Enemies will have a chance to drop beneficial items for the player to use, with bosses having a higher drop rate
- Endless Dungeon will consist of 3 classes (Warrior, Mage, Assassin)
- Players will choose a class at the start of the game
- Each class will have certain advantages and disadvantages
- Enemies will have different types that set them apart from one another
- Players will have to employ convenient tactics to thrive against each enemy type

2.2 Functional Requirements

2.2.1 Play Game

The player will be represented with a simple image or animation on the mid-bottom of the screen. Enemies will be presented on the middle of the screen on each level. The player will be able to choose from various attacks and items in order to attack these enemies. Once all the enemies on the screen are destroyed, the player will progress to the next level. With each level the enemies' health and strength will increase. The player will be able to collect items and improve their hero's skills and stats as they progress.

2.2.2 Pause

The player can pause the game to access an in-game menu. They can save the game, resume the game or quit from this menu.

2.2.3 Save

The player can save the game at any point. The system will record the level and hero progress.

2.2.4 Load

If the player has saved a previous game they haven't finished, they can continue playing that session by choosing this option. The system will load the game with the same level and hero progress as they have left.

2.2.5 Change Settings

The player will be able to adjust some aspects of the game. Currently there are two options:

- Change Difficulty: The player will be able to change the difficulty of the game by choosing one of the three options (Easy, Normal, Hard).
- Enable Sounds: The player will be able to enable or disable in-game sounds.

2.2.6 Help

Information about the game such as instructions and game lore will be shown.

2.3 Non-functional Requirements

- The game will be in RPG format

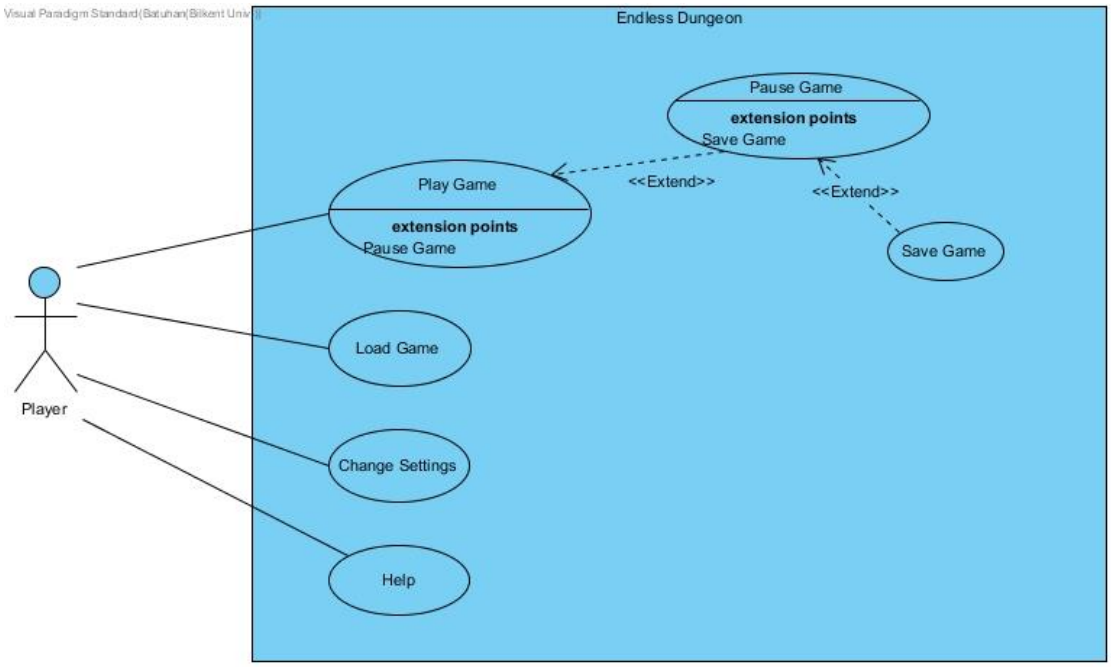
- Players will be able to get stronger as they play
- Enemy styles will be in correspondence with the current map

2.4 Pseudo Requirements

- The game will be developed in Java
- The game will be a desktop application for PCs

2.5 System Models

2.5.1 Use-Case Model



Use Case #1

Use Case Name: Play Game

Participating Actors: Player

Entry Conditions:

Player has opened the game and chose the "New Game" or "Load Game" button.

Exit Conditions:

- Player has successfully completed the game.
- Player has died.
- Player has chosen to quit the game.

Main Flow of Events:

1. Player starts the game.

2. The system loads the appropriate level.
3. Player chooses an attack and attacks the enemies.
4. Player destroys all the enemies.
5. Player progresses to the next level.
6. Player encounters a boss.
7. Player destroys the boss by repeating step 3.
8. Player gains item(s) and experience.
9. Player improves their skills and stats.
10. Player completes the game by repeating steps 3-9.

Alternative Flow of Events:

1. Player loses all of their health and dies.
    a. Player chooses to continue from a saved game.
    b. Player chooses to start a new game.

2. Player chooses to quit the game. (return to main menu)

Use Case #2

Use Case Name: Pause Game

Participating Actors: Player

Entry Conditions:

- Player is in-game and chooses to pause the game.

Exit Conditions:

- Player quits the pause menu and returns to the game, OR
- Player quits the pause menu and returns to main menu,

Main Flow of Events:

1. Player chooses to pause the game.
2. Player resumes the game.

Alternative Flow of Events:

1. Player chooses to pause the game.
2. Player quits the game from the pause menu and returns to main menu.

Use Case #3

Use Case Name: Save Game

Participating Actors: Player

Entry Conditions:

- Player is in the Pause menu.

Exit Conditions:

- The game is successfully saved.

Main Flow of Events:

1. Player chooses to save the game in the Pause menu.
2. The system records the current progress.

Use Case #4

Use Case Name: Load Game

Participating Actors: Player

Entry Conditions:

- Player has opened the game and chose the "Load Game" button, AND
- Player has a saved game.

Exit Conditions:

- Player has successfully completed the game, (?) OR
- Player has died, (?) OR
- Player has chosen to quit the game.

Main Flow of Events:

1. Player starts the game on the saved level.
2. Player follows the steps 3-10 from the New Game case.

Alternative Flow of Events:

1. ???

Use Case #5

Use Case Name: Change Settings

Participating Actors: Player

Entry Conditions:

Player has opened the game and chose the "Settings" button.

Exit Conditions:

- Player quits the Settings menu and returns to main menu.

Main Flow of Events:

1. Player selects the "Settings" button.
2. Player adjusts the difficulty of the game.
3. Player goes back to main menu.

Alternative Flow of Events:

1. Player does not want to change any settings.
2. Player goes back to main menu.

Use Case #6

Use Case Name: Help

Participating Actors: Player

Entry Conditions:

Player has opened the game and chose the "How to Play" button.

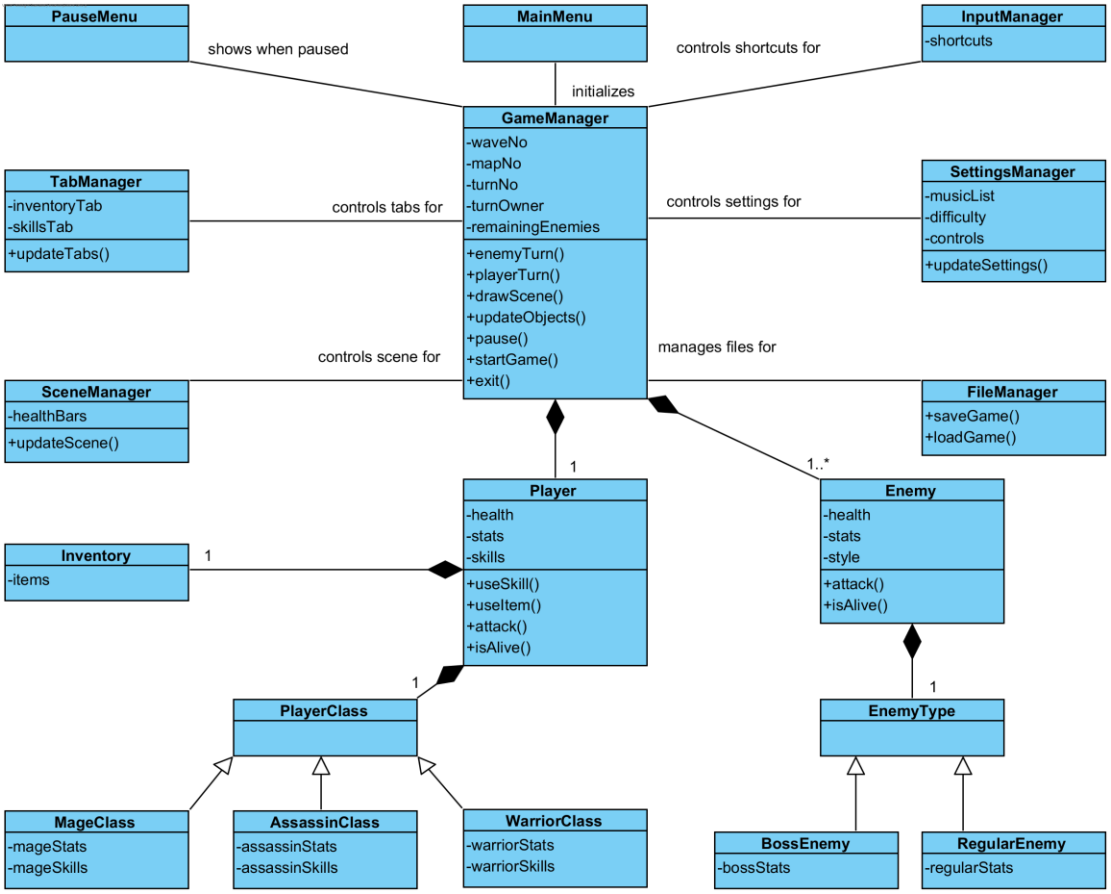
Exit Conditions:

- Player quits the help menu and returns to main menu.

Main Flow of Events:

1. Player chooses the How to Play button from the main menu.
2. Game instructions are displayed on the screen.

2.5.2 Object and Class Model



MainMenu: This class will be the first window that the user sees after opening the game up from their desktop. It will have options to start a new game, change settings, load a saved game or quit the game.

PauseMenu: This class will be the window that the user sees after pausing the game from the GameManager while playing the game. It will have options to see the help related to the game and to exit the game.

InputManager: InputManager will be the class that controls the user-defined shortcuts and input selections (choices to use the keyboard for shortcuts)

GameManager: GameManager will be the main class of Endless Dungeon. It will essentially work as the game engine. All main functions of the game will be controlled from here. Its attributes will contain:

- **waveNo:** The current wave number
- **mapNo:** The current map number

- **turnNo:** The current turn number
- **turnOwner:** The player or the enemy that is supposed to play during the current turn
- **remainingEnemies:** Total number of enemies remaining to pass the current wave

and its methods will consist of:

- **enemyTurn():** Makes the enemies play their turns
- **playerTurn():** Makes the game engine ready to take player input for their next turn
- **drawScene():** Draws the current scene elements like tabs and enemies
- **updateObjects():** Updates the states of objects that are on the current scene like the stats of the player and the enemies
- **pause():** Pauses the game and shows the pause menu
- **startGame():** Initializes the game objects and starts the game
- **exit():** Terminates the game objects and exits the game

TabManager: This class will control the tabs of the game manager. Its attributes will contain:

- **inventoryTab:** A tab on the screen to give the user the option to open the inventory and see their items
- **skillsTab:** Works similarly to the inventory tab but shows the skills of the player's character

and its methods will consist of:

- **updateTabs():** Updates the contents of the tabs depending on the outcome of the last turn played

SettingsManager: This class will control the settings of the game. It will allow users to personalize their game experience by changing certain aspects of the game to their liking like the music and the difficulty level. Its attributes will contain:

- **musicList:** List of the music choices that the user has to play in the game
- **difficulty:** Difficulty level of the game (Easy, Normal or Hard)
- **controls:** The control inputs to use while playing the game

and its methods will consist of:

- **updateSettings():** Updates the settings of the game after the user is done changing them

SceneManager: This class will control the layout of the current scene. Its attributes will contain:

- **healthBars:** The health bars of the enemies and the player

and its methods will consist of:

- **updateScene():** Updates the contents of the scene according to the outcome of the last turn

FileManager: This class manages the saved game for the game manager. Its methods will consist of:

- **saveGame():** Saves the current state of the game to a save file
- **loadGame():** Loads the chosen saved game file and allows the player to continue from wherever they left off

Player: This class will be the player that the user uses to play the game. The user will experience the world of Endless Dungeon as the player and every player's game experience will be different with the help of our progression system and different classes. Its attributes will contain:

- **health:** The current health of the player
- **stats:** The stats (Strength, Agility, Intelligence etc.) of the player. It will be determined with the class of the player
- **skills:** The skills that the player can use to more easily overcome their enemies. Similarly to the stats, it will be determined with the class of the player

and its methods will consist of:

- **useSkill():** Lets the player use the chosen skill
- **useItem():** Lets the player use the chosen item
- **attack():** Lets the player attack the chosen enemy
- **isAlive():** Checks if the player is still alive or not

Inventory: This class will symbolize the inventory of the player. Its attributes will contain:

- **items:** The items that the player managed to gather during their adventure

PlayerClass: This class will be the profession (called class in RPGs) of the player character. This choice will determine the most of the experience that the user gets from Endless Dungeon. Its subclasses will contain:

- **MageClass**
- **AssassinClass**
- **WarriorClass**

and they will each determine the stats and the skills of the player to correspond to their specialties in their own respective fields (More Strength for Warriors, more Intelligence for Mages etc.)

Enemy: This class will be the enemies that the player has to fight against in Endless Dungeon. There will be 2 enemy types that determine the significance of that specific enemy in the game and they will contribute in different ways to the experience of the player. Its attributes will contain:

- **health:** The current health of the enemy
- **stats:** The stats of the enemy. It will be determined by the type of the enemy and the difficulty of the game as well as the current map number that the player is fighting on
- **style:** The style of the enemy visual model to be used that will change depending on the map to fit the background style

and its methods will consist of:

- **attack():** Makes the enemy attack the player to deal damage
- **isAlive():** Checks if the enemy is still alive or not

EnemyType: This class will be the type of a specific enemy. It will determine how much hardship the enemy should prove to be against the player. Its subclasses will contain:

- **BossEnemy**
- **RegularEnemy**

and they will each determine the stats of the enemy to make it harder to beat if it is a boss enemy or make it comparatively easier to beat if it is a regular enemy.

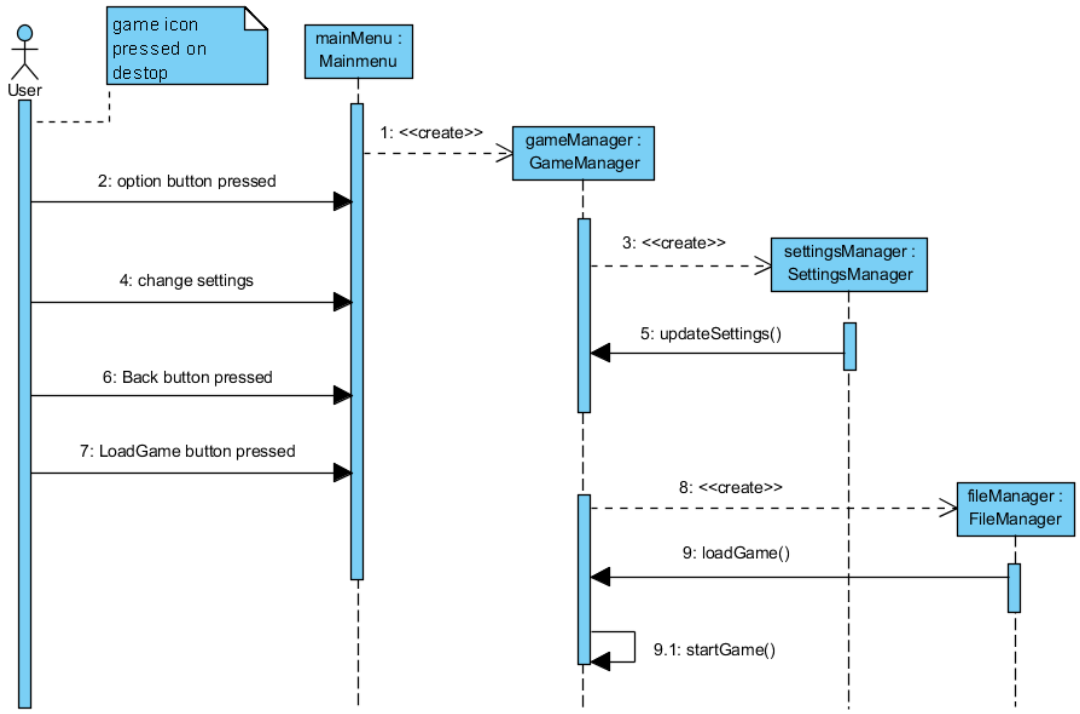
2.5.3 Dynamic Models

2.5.3.1 Sequence Diagrams

Scenario: Changing Settings and Loading Game

User wants to change the game settings and than wants to load the game that he played before. To do that, user opens the Main Menu screen. If user changes the

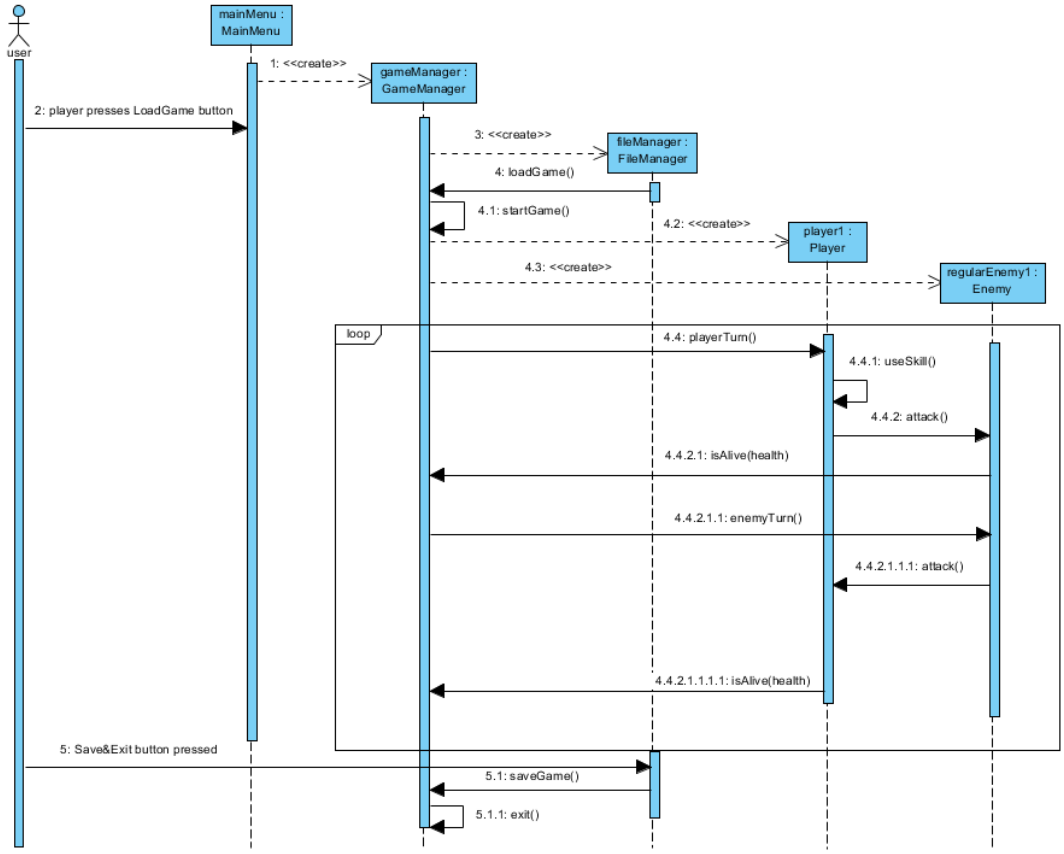
settings, it is updated by Setting Manager by updateSettings() method. After updating settings according to user preference, user can go back to Main Menu by clicking back button.If user does not change the settings, back button will direct him/her to the Main Menu without updating the Setting Manager. Later user wants to load a game that he/she loaded before by clicking load game button on the Main Menu. by clicking that button File Manager will load the game by loadGame() method and after loading complete Game Manager starts the game by startGame() method.



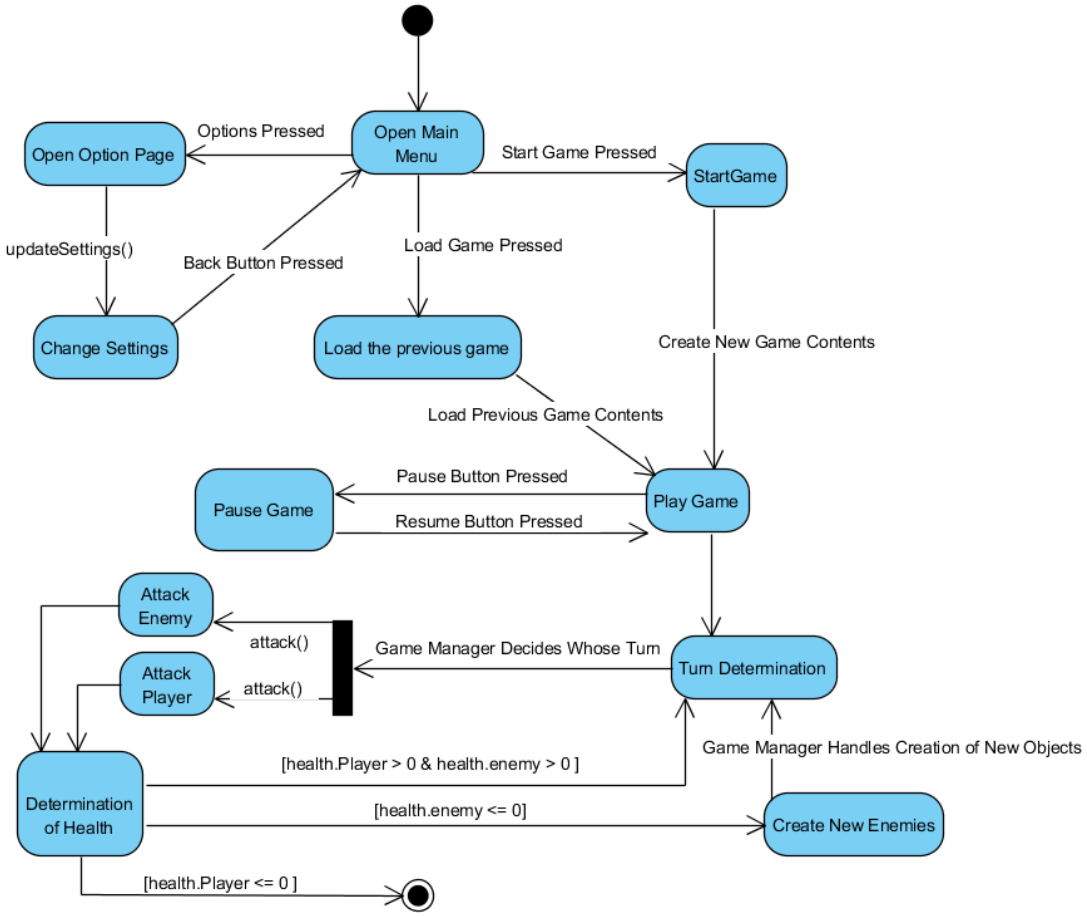
Scenario: Game Dynamic

To continue the game where the user left, player clicks the Load Game button. After loading the saved game by the help of the File Manager (provided by loadGame() method), Game Manager will start the game and Player and Enemy will be created by Game Manager. Game Manager will decide whether is this player's turn or enemies' turn through the playerTurn() and enemyTurn() methods. If it is player's turn user will decide which item to use by useSkill() method and then it will attack the enemy by attack() method. After attacking enemy, Enemy object will send a information to the Game Manager about its health instance.When it is Enemies' turn to attack, enemy will decrease the health of player by attack() method and this time player object will send the necessary information about its health to the Game Manager. Interaction between Player and Enemy will repeat in a loop until the heath of the one of these is equals zero. If the user wants to quit the game, he can click to the Save & Quit button then

this leads File Manager to save the current progress by saveGame() method. Then Game Manager will stop the program by exit() method.



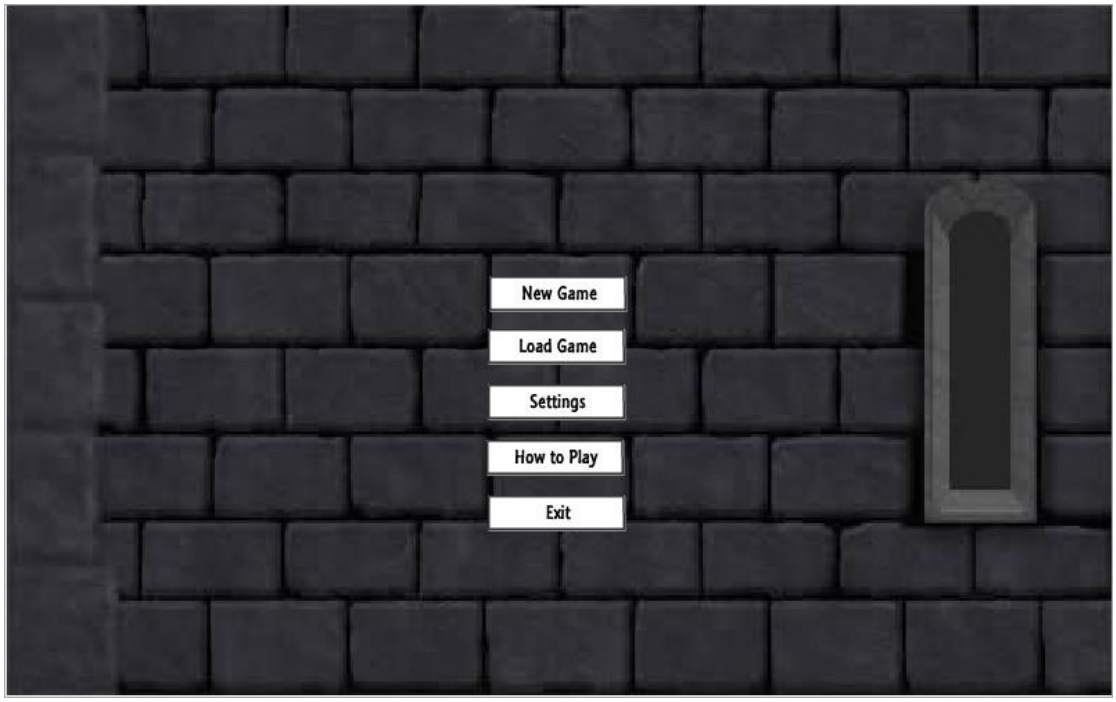
2.5.3.2 Activity Diagram



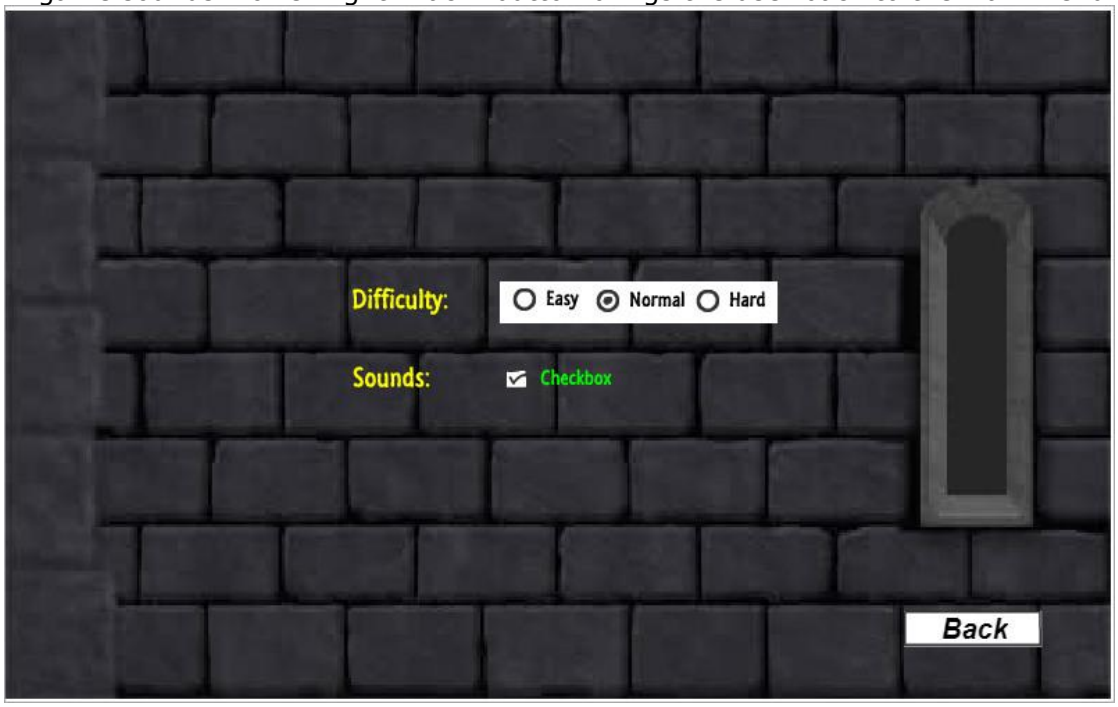
2.5.4 User Interface:

Main Menu:

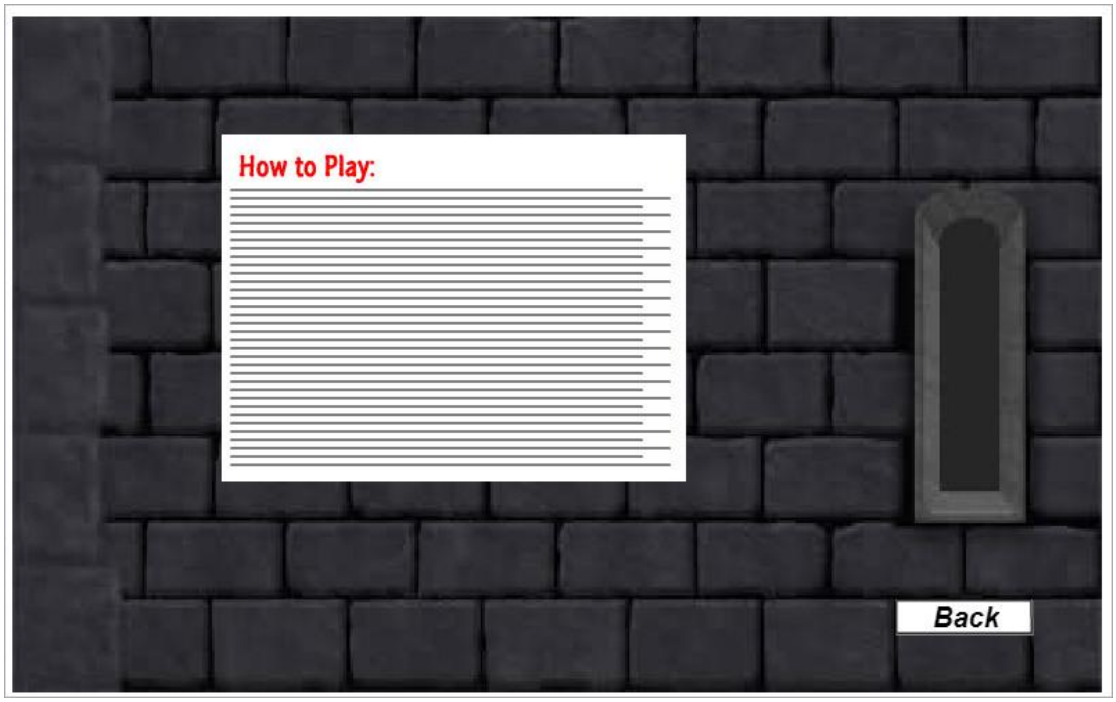
Main Menu is the first what the user comes across when launching the Endless Dungeon application. From here he/she has multiple options to proceed with the game in addition to exit option to leave the game altogether.



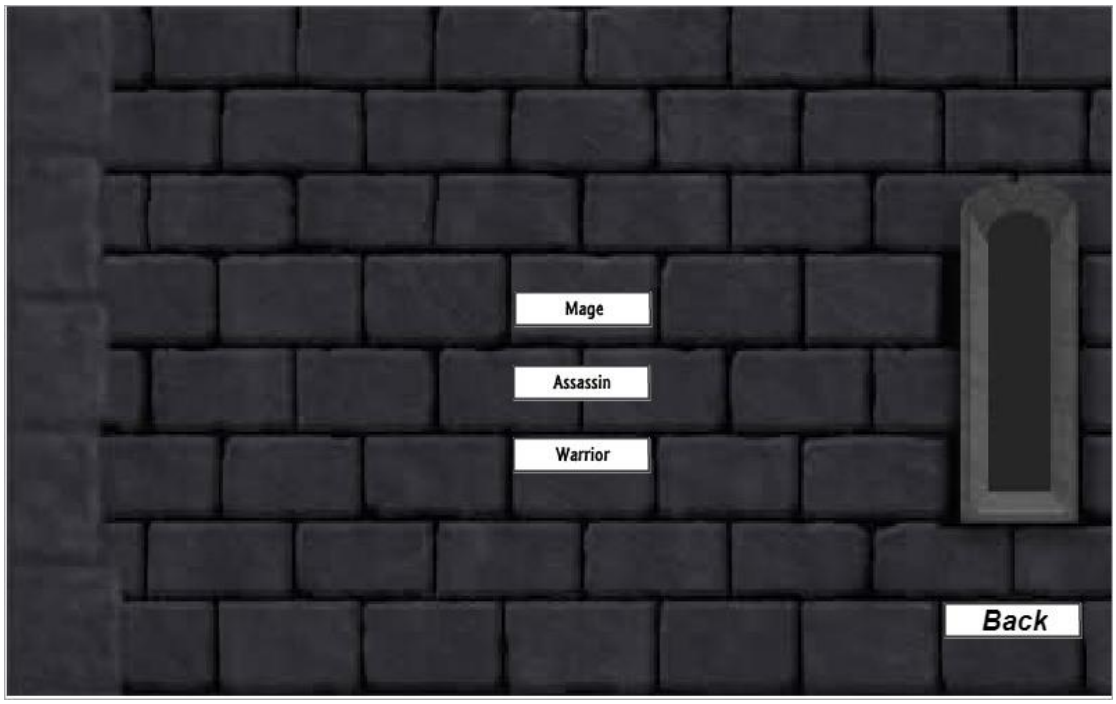
Settings:
"Settings" provides options to adjust game difficulty as well as enabling/disabling in game sounds. Lower right "Back" button brings the user back to the Main Menu



How to Play:
How to Play menu encompasses all the information that would be helpful for the user to successfully play the game

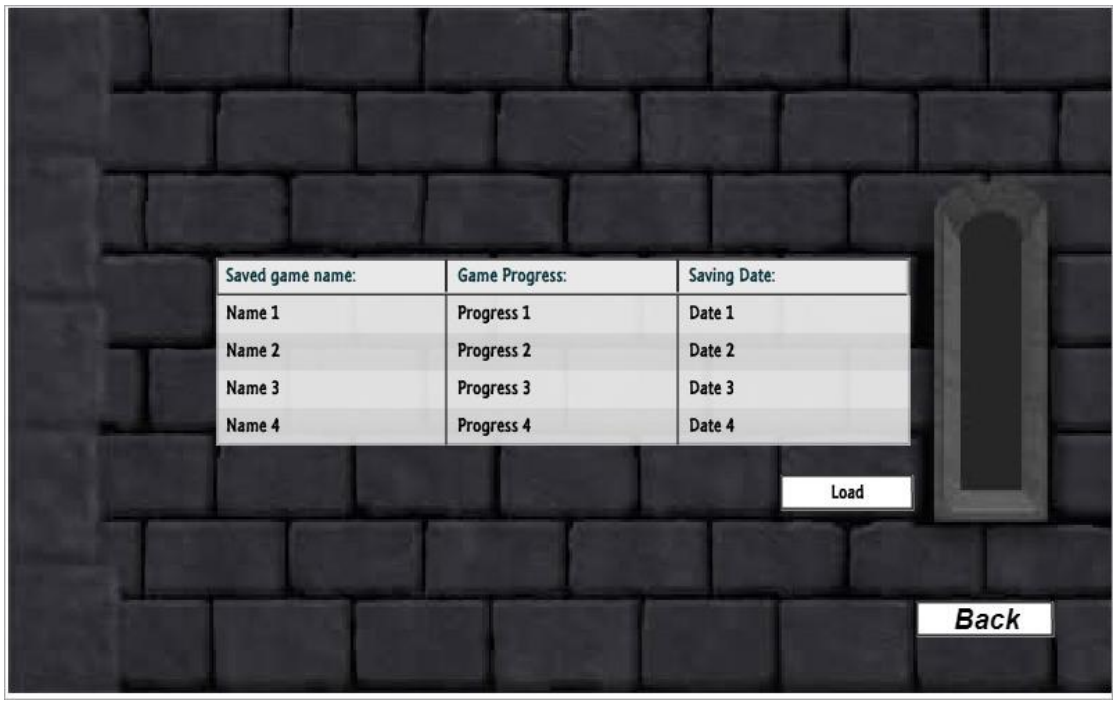


Choose Character:
Choose Character option is displayed when the user chooses "New Game" from the "Main Menu". The user can choose from three types of Characters that will directly affect the gameplay.



Load Game:

If the user wishes to proceed with previously saved game he/she is redirected to the "Load Game" menu, where he/she can choose from multiple(if available) previously saved game progresses.



Play Game:
Play Game is where the player is taken if he/she chose to proceed with the New Game or Load Game options. In the first options the game is started from the initial setup, in the latter one user proceeds with the loaded game progress.

3 References

[1] Object-Oriented Software Engineering, Using UML, Patterns, and Java, 2nd Edition, by Bernd Bruegge and Allen H. Dutoit, Prentice-Hall, 2004, ISBN: 0-13-047110-0.

[2] https://dota2.gamepedia.com/Creeps

[3] http://www.lumzy.com/