Team 12, CSCI 205 Simulator User Manual

Introduction:

Our project was to create a 2D dungeon exploration game, something similar to the original Legend Of Zelda for the NES. The player will be able to explore the map, collecting items, fighting monsters, and eventually fighting the final boss to beat the game. The player must also be careful not to die along the way. Some challenges that must be addressed are:

- Map Generation and Level Design: How does the map generate and where can the player go? Where do the enemies and items spawn? How does the player progress?
- Player Mechanics: How does the player move, attack and interact with items, monsters, and the map itself?
- Enemy Mechanics: How does the enemy behave, is it randomly patrolling or is it attacking the player?
- User Interface and Input Controller: What does the player see on the screen (lives, map, enemies, themselves) and how does the player get feedback from doing certain actions?

The end product is a fully playable 2D game where the player: explores dungeon rooms, fights monsters using a sword, and progresses through levels by solving puzzles.

Background:

To begin implementing the 2D dungeon game, we needed to decide on how we wanted our project structure to look like. We decided to follow the standard MVC (Model, View, Controller) architecture. Model would contain all objects in the game like the Player, Enemies, Map (including obstacles), and any interactable objects (Sword, Riddle Chest) or visual objects (Hearts). View would contain classes that display the game onto a screen for the user to interact with. It also contains helper methods to generate the Map along with other visual indicators like health bars on enemies. Controller contains classes that control game logic, inputs, and collisions. We also included an additional directory: States, which contains Enemy and Item States. We also needed to decide what UI system we wanted to

use. While we had access to JavaFX, we decided to use JFrame instead since we were more familiar with classes and not JavaFX which was a challenge to work with in the past.

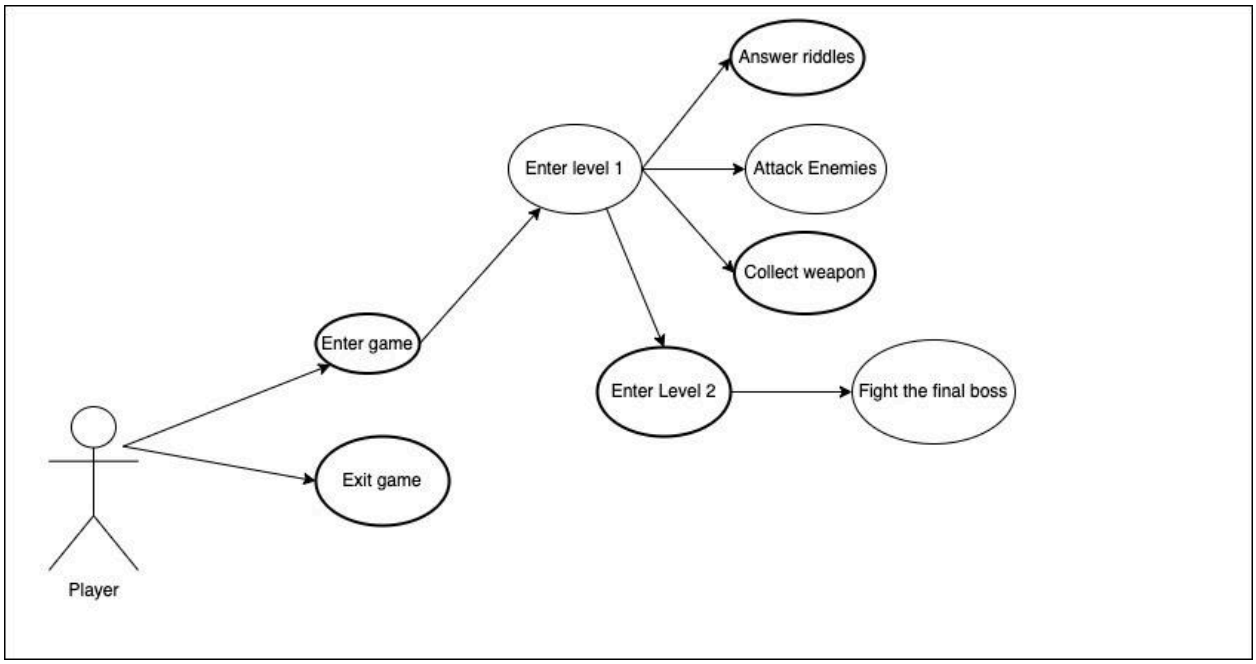
The Process

The first task was creating a central game loop that constantly updated the game sixty times per second which included updating game logic and displaying the results on the screen. This would be critical to the entire game flow. The second task was generating a basic blank screen to begin displaying things onto. We implemented this using JFrame. Next was to then generate a map onto the blank screen using a tile system. This tile system would help us also generate items and enemies on specific tiles. The tiles also help us define a collision area in squares which simplifies collision detection later on. We decided to have a txt file that contained relevant data like tile information, enemy spawns, and item spawns. This was easier to work with than randomly generating a map. From here on out, we were also developing sprites to display on the screen such as what tiles looked like or what the Player and Enemies also looked like. Going forward, this was also in development alongside the current task at the time.

After generating a basic map on the screen, we needed to create a Player for the user to control. This involved creating a Player class and an Input Controller to help control the player using the WASD keys. In addition to this, there would need to be a camera that follows the Player around the map as the Player moves. To improve rendering, the UI would also only render the things on the screen and not the stuff offscreen, a common practice in modern games nowadays. After creating something for the Player to control, we would then work on basic collision between the Player and the Map so that the Player would not be able to pass through the walls of the map. This also included how the Map would be sectioned off into two levels, one where the user had to fight enemies and acquire the sword, and the final level where there was the final boss.

At this stage, we had two avenues to choose from, that being working on the Items or the Enemies. We focused on the Items class first. There are only two items that the player can interact with, that being the Riddle chest which displays a riddle for the user to solve and the Sword which is used to attack enemies. These items had a range where the player needed to be inside in order to interact with

them. Next was the enemies and the combat system as a whole (this includes lives and damage). We needed to spawn enemies on the map, including keeping track of their behavior and hitboxes which would be used to calculate if the player hit one with the sword. Enemies include the final boss and the minions that appear. Interactions needed to be developed so that when the Enemies hit the Player, the Player took some damage and when the Player hit the Enemies, they took some damage. The game is extremely linear which is by design since we needed to focus on the core functionality of the game. Below is a UML case diagram to help explain how the user can interact with the game.



Instructions: If the user has Intellij and Gradle installed, then they can run the program directly using the "Run" button in Intellij from the DungeonGameApp Class. This class can be found in the following directories: src/main/java/org/team12

The user can also use the command in the console (exactly as is and without the quotations): "./gradlew run" in the console will also run the DungeonGameApp class. This will immediately launch you into the game so just be prepared to move quickly.

Once in the game, the user can move around freely using the standard WASD or arrow keys. The user can interact with items using the SPACE bar so long as they are close enough and the user can attack nearby monsters using the E key. The user must be facing in the direction they want to attack otherwise in the direction they are currently facing.

Below is a photo of how the game looks.