Ankara Yıldırım Beyazıt University
Department of Computer Engineering

CENG 201 – Object Oriented Programming
Course Project

KIT-201

Project Analysis Report

Uğur Kuş – 22050111002
Aziz Önder – 22050141021
Emin Salih Açıkgöz – 22050111032

Instructor: Muhammed Abdullah Bülbül
Teaching Assistant: Elif Şanlıalp
Date: 30/11/2023

Table of Contents

1. Introduction

Our project, **KIT-201**, is a platformer game inspired by the Mega Man series. In this game, the player controls KIT-201(**Ki**lling**T**ool-201), a robot assassin that slowly gains sentience while completing tasks assigned to it. The game includes typical platformer elements: a character with health, movement abilities (like running, jumping, shooting), and the capacity to collect upgrades and charge energy. KIT can use upgrades, obtain power-ups, and equip secondary weapons for varied gameplay. There will be challenges like slippery platforms, moving obstacles, and many types of enemies at each level. There will also be friendly NPCs offering items and information. This analysis report covers various gameplay scenarios; it includes a UML use case diagram, a UML class diagram, concept art of user interfaces, and addresses both the functional and non-functional requirements of our project.

2. Requirements

2.1. Functional Requirements

2.1.1. Game Manager

- Manages game flow, transitions, saving, loading, and game state.
- Manages display and audio.
- Catches potential exceptions that may occur.
- Contains primary game loop.
- Manages keyboard input.
- Manages UIs

2.1.2. User Interfaces

- Manages player movement, NPC interaction, powerup handling, and shooting.
- Displays Health, Energy, Equipped Upgrades, Powerups, current weapon(s), and unused items using a status HUD.

2.1.3. Items (Power-Ups, Weapons and Upgrades)

- Includes two slots for powerups.
- Upgrades modify the player's stats (i.e. maximum health is increased).
- Power-ups give the player new abilities (i.e. double jump perk).
- Weapons modify the player's shooting attack behavior.
- The player can wield a secondary weapon that modifies the player's attack behavior.

2.1.4. Entity Behavior

- Implements Player Behavior.
- The player gets points for defeating enemies; enemies may drop items.
- Introduces enemies with hidden health that can deal and receive damage.
- Implements enemy behavior: movement, jumping, shooting, climbing, damage dealing.
- Boss enemies are a subtype of enemy.
- Implements NPC behavior: movement, dialogue, item exchanging.
- Implements obstacle behavior: movement, damage dealing.

2.1.5. Levels

- Includes tiles, hazardous tiles, NPCs, enemies, items, obstacles, pits, hidden areas, interactive entities, gravity modifiers, and movable objects.

2.2. Nonfunctional Requirements

- Provide consistent frame rate for players.
- Optimized loading times for better user experience.
- Providing a comfortable user experience during gameplay.

3. System Models

3.1. Scenarios

Scenario A:

1- The player enters the game.
2- The player encounters a "press any button" screen, and then the main menu appears.
3- The player enters the game from the main menu.
4- The game starts from the level or checkpoint where the player last left off. If there is no saved data, the first level is loaded.
5- The game receives keyboard inputs from the player. KIT-201 moves, jumps, shoots at enemies, and uses power-ups based on the player's input.
6- KIT-201 encounters a story NPC, talks to them, obtains a weapon, and continues on their way.
7- KIT-201 encounters an enemy, shoots at it, and the projectiles deal damage to the enemy. The enemy's health reaches zero, and the enemy is defeated.
8- KIT-201 collects collectibles that provide points.
9- KIT-201 runs into a power-up and unlocks the ability to double jump.
10- Initially, KIT-201 collides with a different kind of enemy's projectile, then the enemy itself, and finally, an obstacle. The player's health reaches zero, and KIT-201 dies.
11- A "Game Over" screen appears.
12- The player restarts the game, and the game restarts from the last checkpoint or beginning of the level.

Scenario B:

1- The player starts playing the game.
3- KIT-201 moves, avoids enemy projectiles, and collects all the collectibles.
4- The player pauses the game and resumes later.
5- KIT-201 falls into a pit and dies.
6- The player quits the game.

Scenario C:

1- The player enters the game.
2- The player changes settings.
3- The player quits the game from the main menu.

3.2. Use Cases

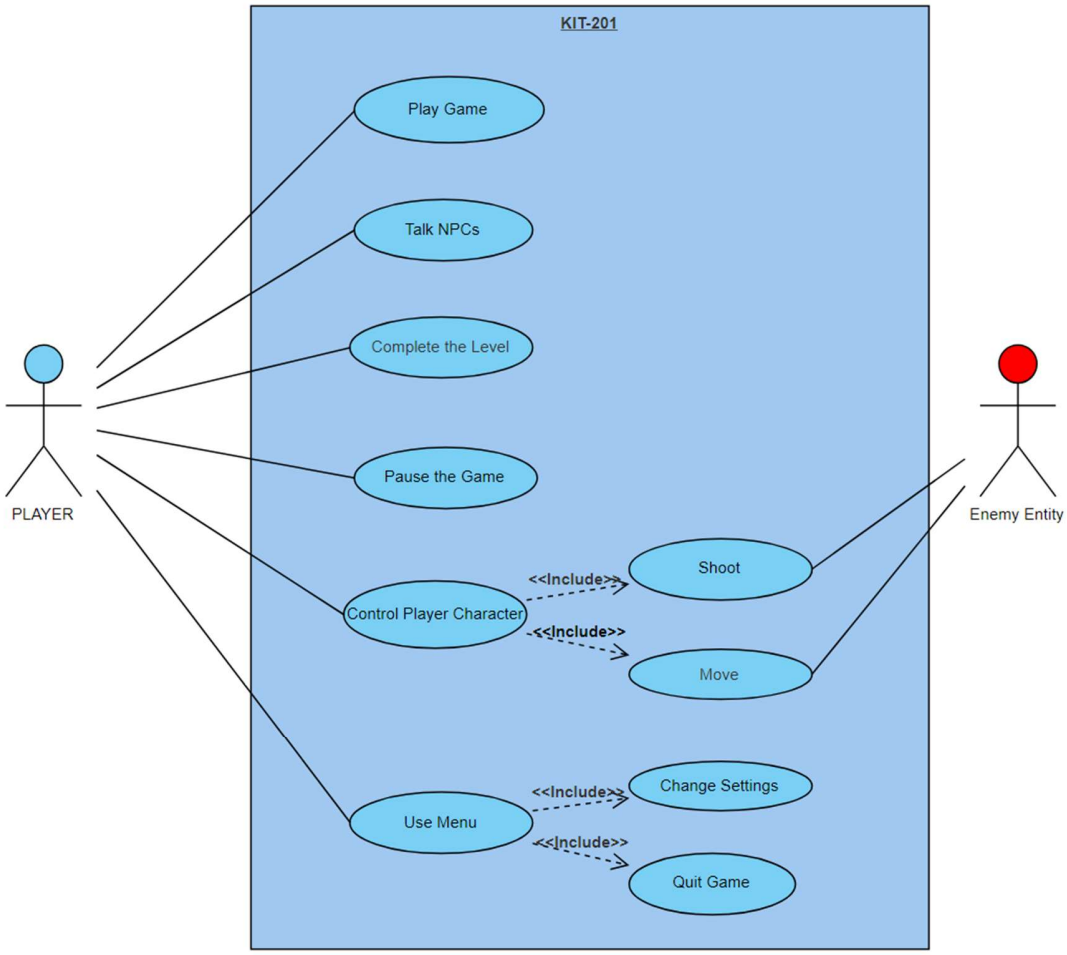

Figure 1.0 UML Use Case Diagram

3.3. Object and Class Model

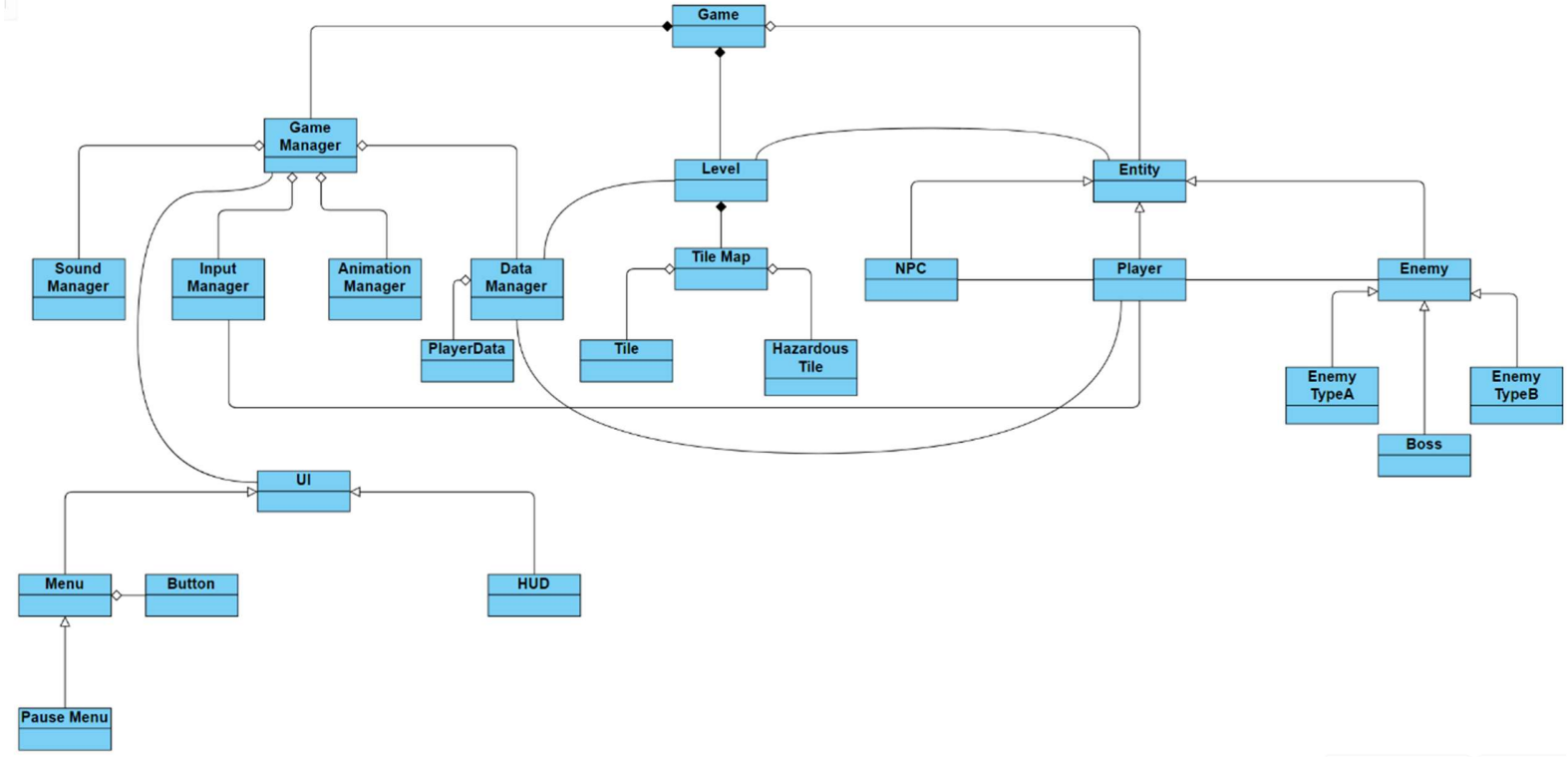

Figure 2.0 UML Class Diagram

UML Class Diagram.pdf

**See PDF for higher Resolution*

3.4. User Interfaces

Figure 3.0 Main Menu

Figure 3.1 Representative In-Game Screenshot

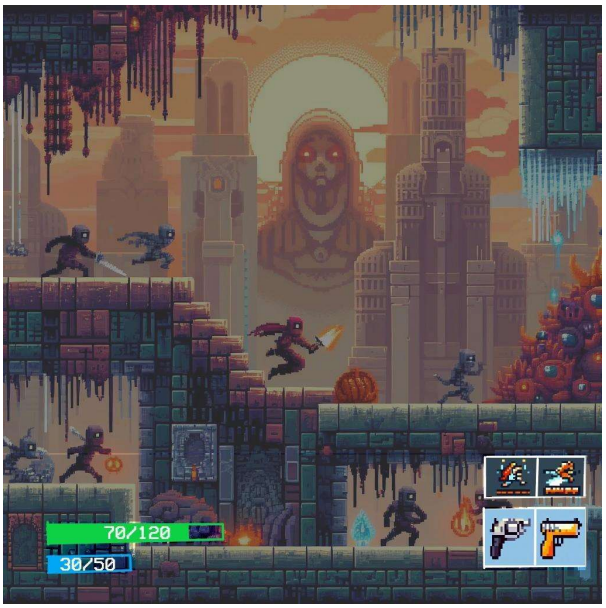

Figure 3.2 Pause Menu

Figure 3.3 Settings

**Background images and in-game icons (weapons and power-ups) have been created by AI.*

4. Conclusion

In conclusion, our report breaks down our project's concept into functional and non-functional requirements, a use case diagram, a class diagram, scenarios that may occur during gameplay, and user interfaces. To be concise, we grouped our functional requirements under the Game Manager, User Interfaces, Items, Entity Behavior, and Levels. The necessary behavior was elaborated under these titles. Our non-functional requirements address the performance and user experience of our project. Our class diagram elaborates on how our game's code will be structured and how the submodules interact with each other. Our use case diagram roughly represents what behaviors the player and a generic enemy will possess. Finally, we provided the concept art of the user interfaces that will be implemented in the game.

Contributions:

Aziz Önder: Class diagram design, scenario writing, use case design, revision of functional requirements.

Emin Salih Açıkgöz: Class diagram design, scenario writing, determining functional and non-functional requirements.

Uğur Kuş: Class Diagram design, UI design, concept art, revision of functional requirements.