Team 12, CSCI 205 Simulator Design Manual

Introduction

The system is a 2D dungeon-style simulation game implemented in Java. It is designed with an object-oriented architecture and rendered using standard Java libraries such as javax.swing, java.awt, and java.imageio. The game runs on a 60 FPS loop. It simulates the dungeon crawler game type where the player must navigate through map levels, collect items, solve riddles (mostly based on CSCI204 and CSCI205 concepts), fight enemies (TA goons), and eventually defeat the final boss, Prof. Lily, who has been possessed and needs to be cured.

The system comprises several major components: a game loop controller (GameUI.java), rendering logic (EntityRender.java and MapRenderer.java), input controller (InputController.java), collision controller (CollisionController.java), game state and entity management (GameController.java and Map.java), and the core domain model (Player.java, Enemy.java, Item.java, Tile.java, etc.). Entities in the game inherit from an abstract Entity.java superclass, applying polymorphism and encouraging code reuse. A robust map system initializes the game, loads item and enemy spawn locations from a map text file, and reflects updates on the screen as the player moves. The design ensures modularity, separation of concerns, and extensibility, making it suitable for future expansions such as additional levels or layers, boss logic, or multiplayer options.

The project structure demonstrates good object-oriented design and encapsulates game logic within separate classes. For instance, combat interactions and movement are separated from rendering logic. A player's movement updates the camera angle by adjusting coordinates, and renderers dynamically adjust what is visible on the screen based on the player's position. This enhances the performance of the system and makes it maintainable. Sprites are loaded via ImageIO, and the use of collision detection and camera logic ensures smooth gameplay even as the number of items and enemies scales up.

User stories

The game development was guided by several user stories that were identified throughout the project:

”As an explorer, I want to move around the map using the keyboard so that I can explore it and collect items.” This is implemented via InputController.java and integrated with GameUI.java game loop. The game takes the keyboard input, updates the player’s coordinates accordingly, and reflects it on the screen.

“As an achiever, I want to solve the riddles on the laptops so that I can unlock weapons.” This is implemented through Laptop.java, which poses computer science-related questions.

“As an achiever, I want to collect swords and other items so that I can use them later in combat.” This is implemented via Item.java, and an inventory system is created in Player.java. Items like Sword, MagicDust, and RiddleChest are subclasses of Item, each with different behaviors.

“As a killer, I want to defeat the TA goon squad so that I can save Lily.” This is implemented via Enemy.java and EnemyStatus.java, which control enemy behavior (e.g., peaceful vs. hostile). Enemies wander randomly in the map until they can detect the player are close to them, at which point they become aggressive.

“As a achiever, I want to cure Lily so that I can win the game.” The story is built from LilyFinalBoss.java as a subclass of Enemy.java.

Object-oriented Design

The main design of the project follows object-oriented programming principles, focusing on encapsulation, inheritance, modularity, and polymorphism.

The object model centers around the abstract superclass Entity, from which both Player and Enemy inherit. This structure encapsulates shared behaviors such as movement, directional sprite loading, coordinates, and HP management. Polymorphism is reflected through EntityRenderer, which dynamically decides how to draw all entities based on their class.

Items are implemented as an abstract Item class with subclasses that define different effects. Sword increases damage. MagicDust can cure Lily. Heart items visually represent health status. Laptop provides quiz functionality, including randomized computer science-based questions and tracked player answers.

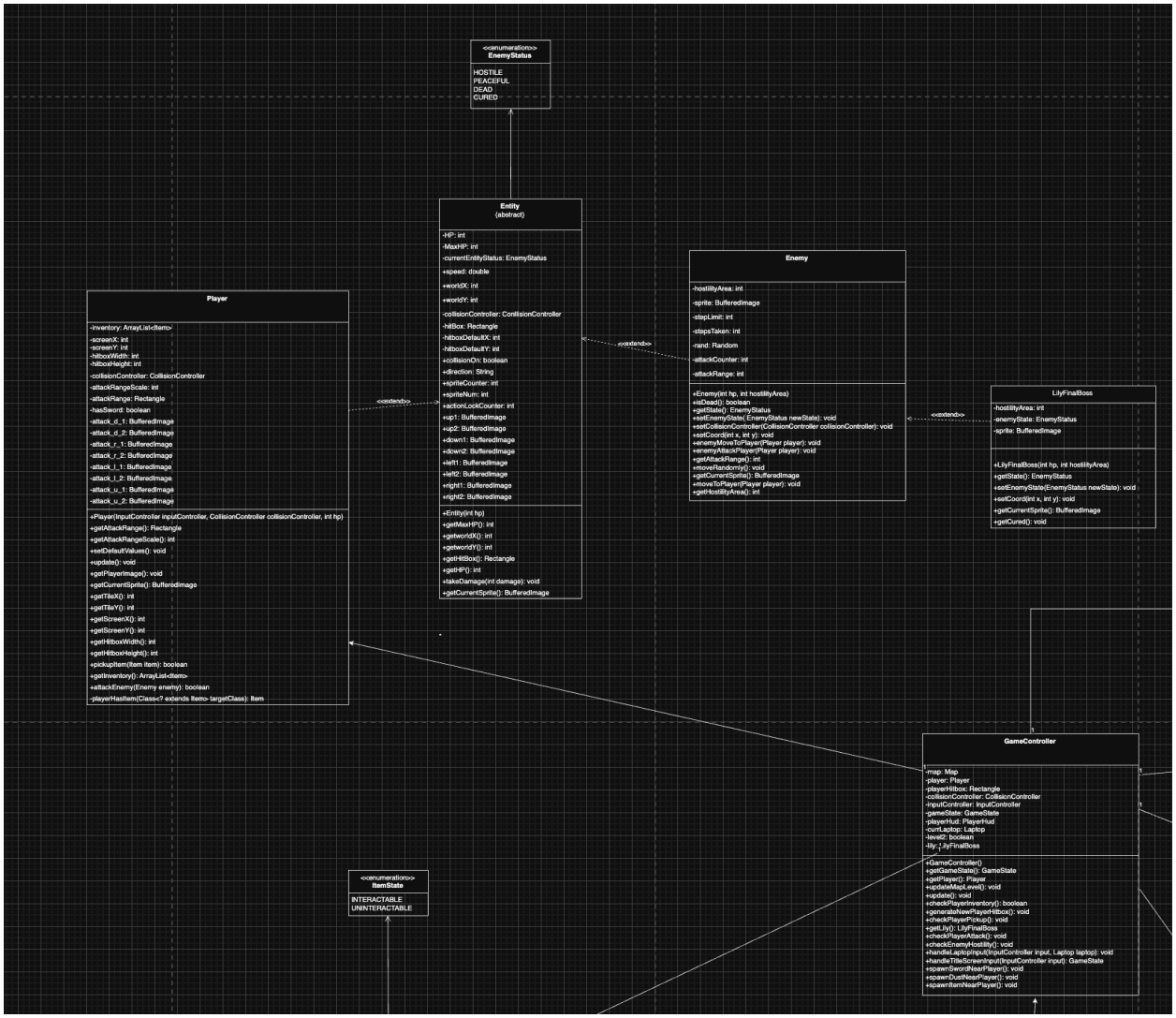
The Map class handles the game's logical structure. It loads layout data from a .txt file, converts numbers to Tile objects, and stores all game items and enemies. It manages a grid of Tile objects, each of which may contain an enemy, an item, and properties like isObstacle or isPortal and functions as the central placeholder for all in-game locations and enables precise tile-by-tile interaction tracking.

The rendering system is separated into MapRenderer for background tiles, EntityRenderer for dynamic objects, and PlayerHud overlays UIs like the title screen, death screen, win screen, and quiz interface. This separates the game visuals and logic. Rendering is resolution-scalable using tileSize, and camera logic keeps the player centered while drawing only visible entities around the player.

Controllers follow the MVC pattern and manage user input, collision checking, and gameplay updates. Game logic is managed by GameController, handling movement, item interaction, state transitions, and quizzes. InputController takes in key presses and translates them into movement flags or triggers interaction. CollisionController checks whether the player or enemies can move to specific tiles based on obstacles and other entities.

Enums are used throughout the system to manage clean state transitions. EnemyStatus manages the phases of enemies (PEACEFUL, HOSTILE, DEAD, CURED). GameState controls global game flow (PLAYING, QUIZ, PLAYER_DEAD, LILY_CURED, PAUSE, END). ItemState defines item interactivity.

CRC Cards

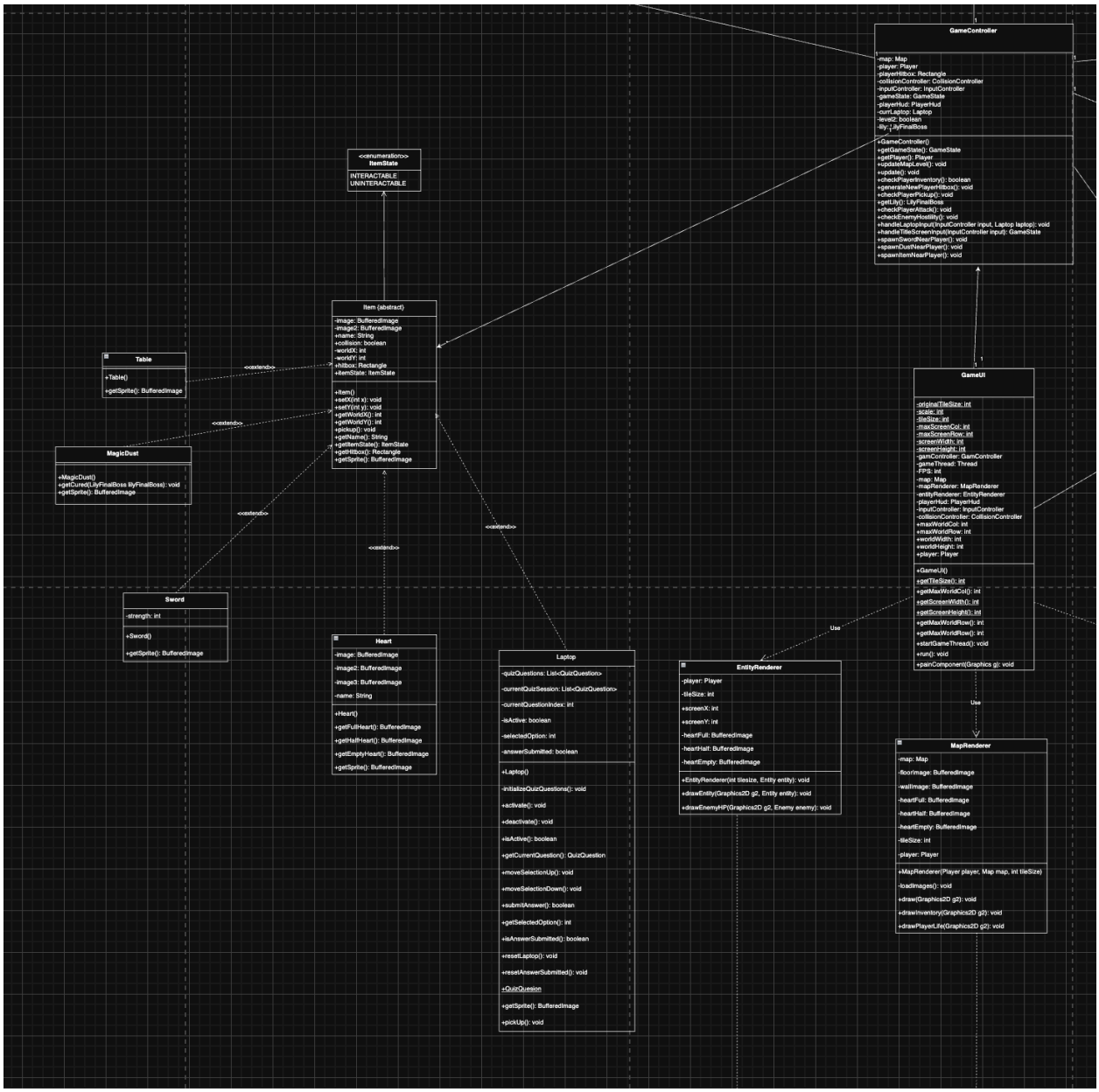


Entity is responsible for movement, health, and sprite direction, alongside with GameController for combat, EntityRenderer for rendering, and CollisionController for movement validation.

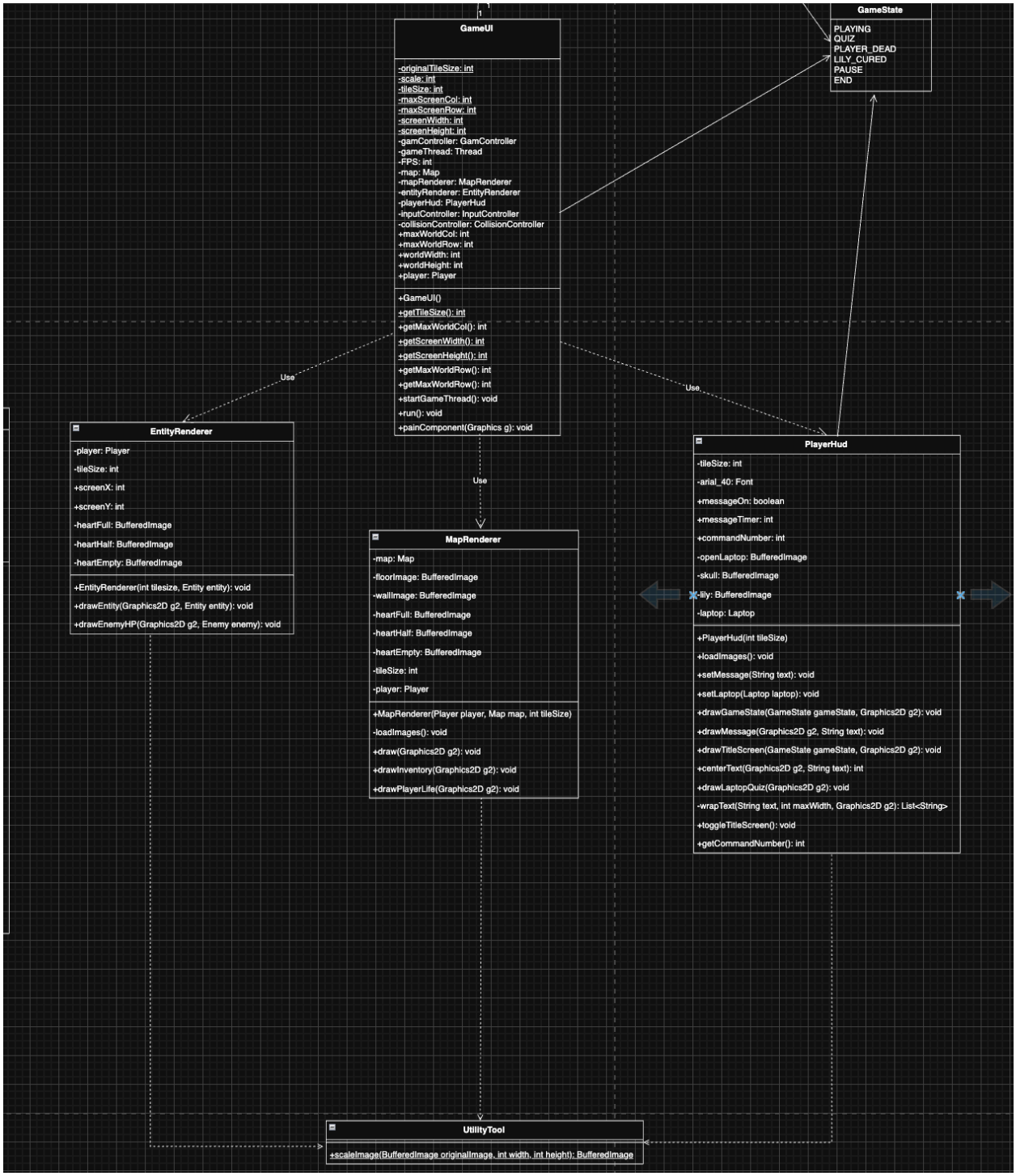
Players add user-controlled behavior, inventory management, and attack interactions. It collaborates with InputController and GameController for updates and with Enemy for the combat system.

Enemy has random movement and status transitions. It can become hostile when detect the player inside the hostile area. It collaborates with Player and CollisionController.

LilyFinalBoss extends Enemy, supports getCured() logic and sprite behaviors. It collaborates with MagicDust and Player.

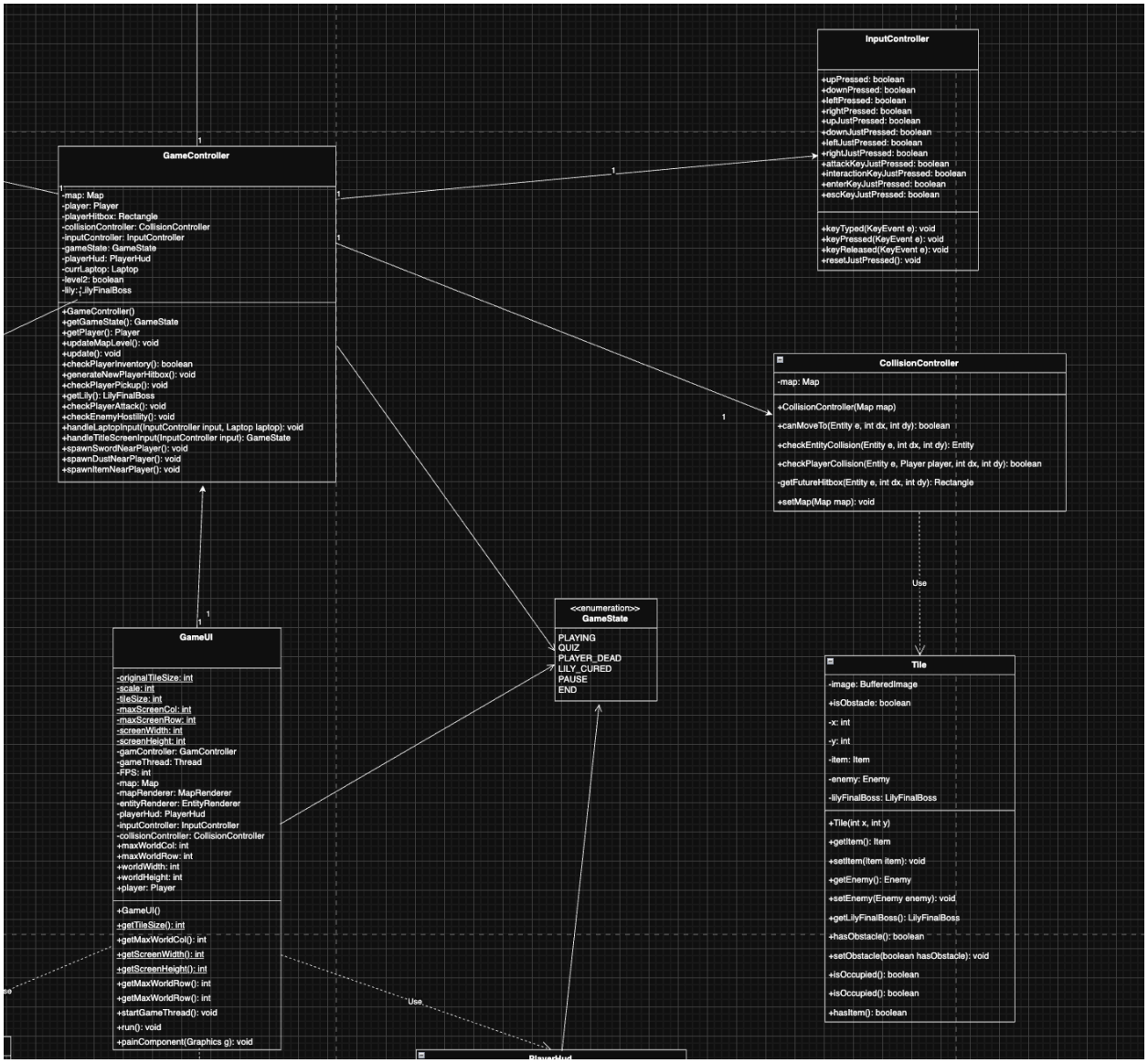


Sword, MagicDust, Laptop, Heart: Subclasses of Item with specific behaviors (e.g. combat, healing, quizzes). Laptop presents quizzes with UI integration, triggering item rewards when passed and collaborating with PlayerHud and GameController.



Map loads from a text file, places enemies/items, tracks level data. It collaborates with GameController, Tile, and MapRenderer.

Tile represents a location in the map and holds state like item, enemy, or obstacle. It collaborates with Map.



GameController manages logic such as enemy, item pickups, combat resolution, and winning conditions. It collaborates with Map, Player, Laptop and Enemy.

InputController translates keypresses into directional movement of the player. It collaborates with Player and GameUI.

EntityRenderer is responsible for drawing entities and health bars. It collaborates with GameUI.

MapRenderer is responsible for drawing static tile backgrounds, obstacles, inventory, and item pickups. It collaborates with GameUI and Map.

PlayerHud renders UI overlays like the title screen, end game screen, and laptop quiz browser. It collaborates with GameUI and GameController.