Group 14 — Phase 2 Report
Raccoon Simulator

Alex Impert, Ryan Wittkopf,
Daniel Todd, David Yao

March 2022

1 Introduction

The purpose of this report is to chronicle the implementation of our SFU Raccoon simulator. Any changes or modifications made to our initial design phase, including the class UML diagram and the list of use cases, will also be documented. Additionally, the tools used along with the delegation process we chose will be detailed.

2 Implementation & Modification: Use Cases

Our use cases aptly defined how we wanted the game to be played. The main "loop" of the game would be as follows:

1. Main menu loads, player can select to exit or start the game.
2. Gain score from collecting rewards (garbage and raccoon lady).
3. Lose the game from colliding with an enemy, score dropping below 0, or lose score from hitting a trap.
4. Take the main exit door after collection of all items to end the game.
5. At any point in the game, Esc can be pressed to pause the application.

Of these use cases, we successfully satisfied all of them according to the constraints we gave them, with the exception of a main exit. Due to time limitations, the game "ends" once all rewards are collected.

3 Implementation & Modification: UML Diagram

Our 2D game was built using the Java language and therefore operates on the Java Runtime Environment. In order to produce a window (GUI), the JFrame library was used in unison with JPanel to implement containers in order to have multiple visual components inside one.

As expected given our UML diagram (Figure 1 below), Java runs our main function which constructs a JFrame window as well as our singleton RaccoonGame object and adds it to the window. Additionally, some of RaccoonGame's methods are called to initialize assets and start the game loop which is an ongoing thread that updates our game continuously.

Our RaccoonGame constructs a variety of composites for our game to function accordingly to our constraints and use cases. They are broken down as follows:

MapManager Loads up our block sprites into an array that is later accessed to draw the map.

ObjectHandler Similar to the MapManager, this class checks where items are spawned and processes them in.

CollisionHandler Scans objects surrounding a subject and checks if they collide.

KeyHandler Associates commands with certain keystrokes (movement).

EnemyHandler Spawns enemies into our map.

Player Spawns the playable character into the map, as well as contains methods for movement and reward retrieval.

These classes, which we mostly labelled as "Handlers", are the building blocks of the game as they are the ones that actually construct and operate our world. Although we had mostly the right idea in our initial UML diagram (fig 1), we chose to separate the handling of different game objects rather than having them all within one MapManager (GameMap in the UML diagram).

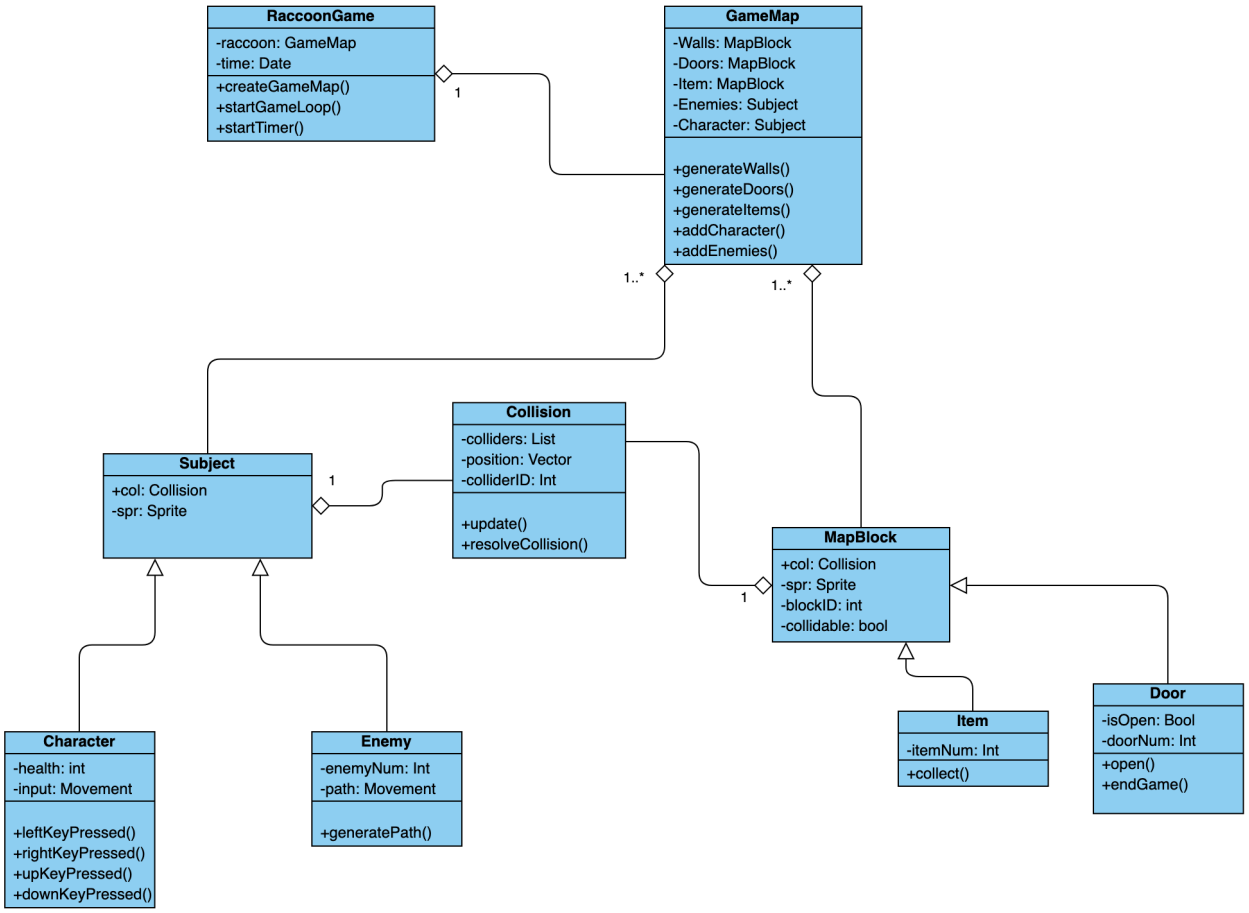

Figure 1: Initial Raccoon Simulator UML Diagram

Continuing, each of these handlers have different objects they work with. Most of these are self explanatory, but some examples are:

MapBlock Contains our buffered image for a map block, which is essentially just a floor tile or wall tile.

GeneralObject Also contains an image, but has more members to detail collision hitboxes. Items, rewards, and doors inherit from this class.

Subject General class for enemies and the character, carries spacial awareness information, sprite, and animation variables.

4 Management & Delegation

To effectively stay on track, especially given our various deadlines for other courses, we decided to implement the Scrum methodology in order to achieve incremental development. We collectively agreed on Thursdays to be our weekly Scrum meeting where we share what we have done and what needs to be done going forward. This is was especially beneficial as it let us focus on the most important things that need to be done first to meet our sprint deadlines.

As the Scrum method encourages cross-functionality, delegation came down to a pick and choose based off preference. However, in order to keep track of what needs to be done and who is working on what, we utilized GitLab's Issues Board as a way to visualize the tasks in our project. This allowed us a method of organizing our delegation throughout the weeks.

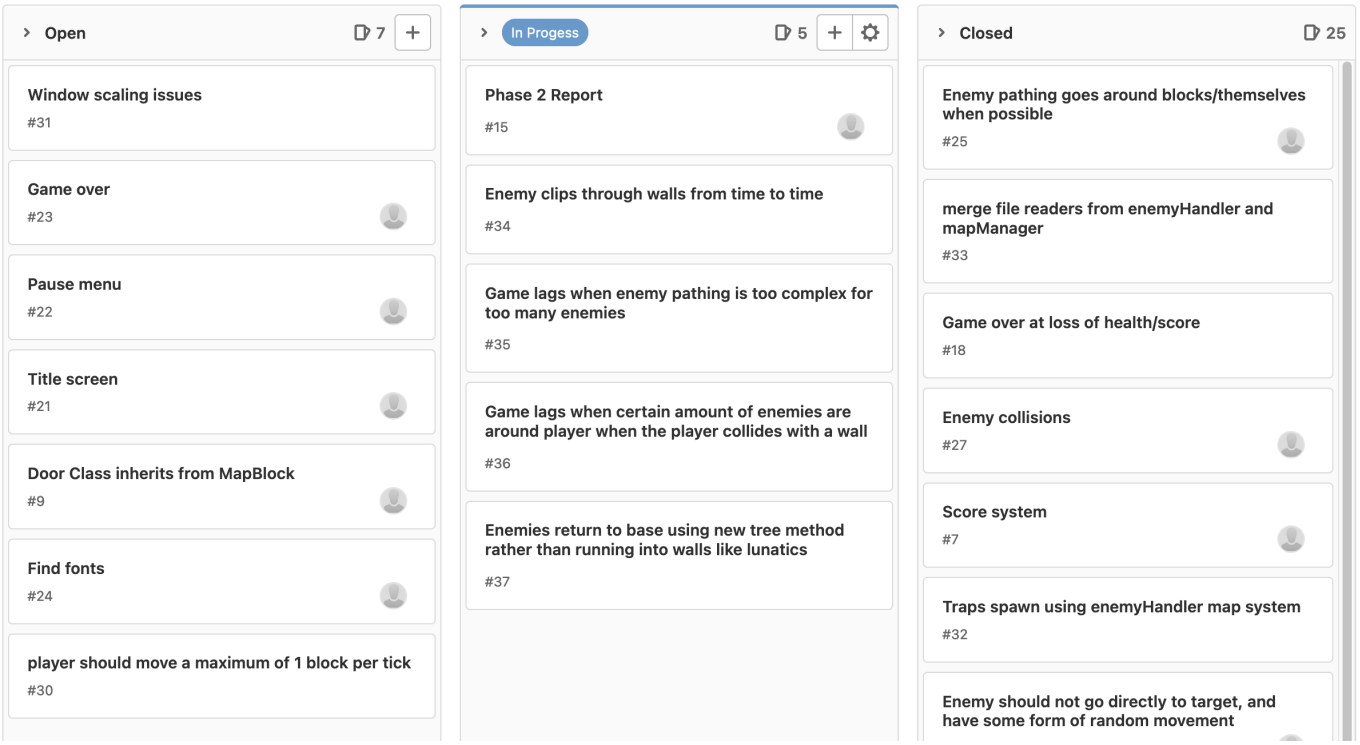

Figure 2: GitLab's Task Board

5 Reflection

Given the curriculum for most students, creating a video game is usually fairly different to what we are used to. As such, it felt moderately difficult to adhere to the most appropriate design principles as learning how to make a game within Java was the first step before we could focus on anything else. Going forward with testing and refactoring, we would like to implement more of these principles such as the single dot rule and further abstraction of methods to clean up code. However, as the development process rolled along, we made a keen effort on commenting our code in order for the other's to be able to get up to speed as fast as possible. Additionally, the act of breaking up our MapManager into separate handlers allowed for higher quality code as it was much easier to process everything according to the name of the class it belongs to.

The area we seemed to struggle the most was collision. Initially, the handling of collision was written on a pixel-to-pixel basis, as the character's movement could technically be "in between" tiles. The handler we wrote had to check both sides of a tile in order to process a collision. This bogged down our game and we ran into many issues with clipping between objects. Eventually, we settled on a tile-to-tile system and found that the game performed much better and was still very playable.

Our last challenge that we collectively agreed on adding to this report, is the continued absence of one of our group members. We value communication greatly. If a member is not holding up their end of the work but they let us know, then it won't be an issue as it can be resolved as a team. However this particular member was absent for the first phase of the project, and informed us he would finish the UI aspect for the second phase. Although to their credit they did work on it, it was never finished and did not communicate this to the rest of the group. As such the responsibility fell to another member close to the deadline.