ASSIGNMENT 3

SEG 2105 - INTRODUCTION TO SOFTWARE ENGINEERING

Fall 2021

School of Engineering and Computer Science

University of Ottawa

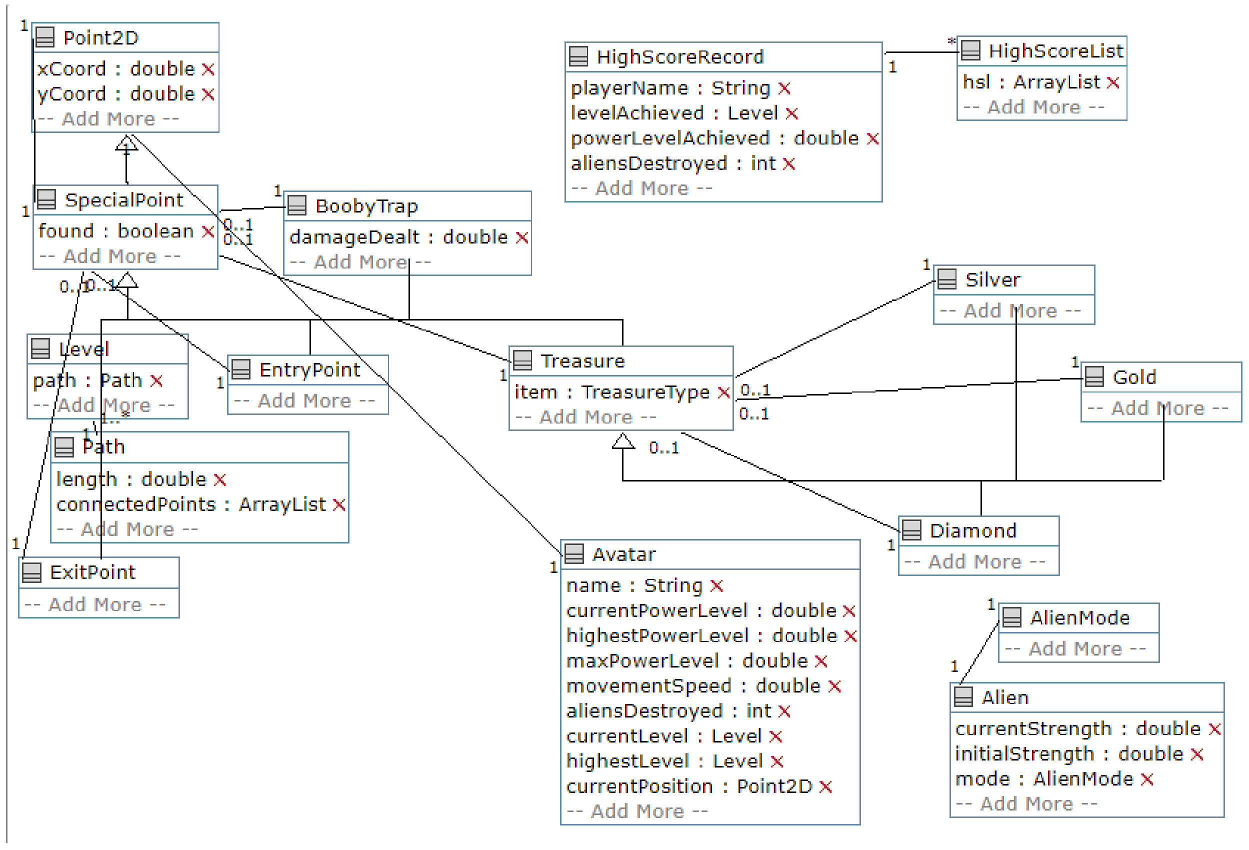
Course Coordinator: Miguel Garzon

Teaching Assistant: TBD

Jayden Lachhman, 8791694

Submission date: 2021/12/04

PART 1: CLASS DIAGRAM



Façade class:

The Façade class for this implementation is the Level class. This design pattern works constructively here since there are many sub-level elements that are unique to each level such as; pathway design, treasure, alien, booby trap, entry and exit locations. The system would suffer logically if the user had access to each of these sub-elements before instantiating an object of the Level class through which the user must discover their environment.

Design Patterns:

Observable-Observer: For Aliens health so that the system does not have to constantly check what the Aliens health is after each action to determine if that Alien should start running away from the Player. This design pattern would accommodate all functions of the game that require real-time response systems, such as discovering special points and reacting to booby traps.

Read-only Interface: The ‘High Score List’ should be read-only, as it would be unfair for players to have permission to alter the information on the list.

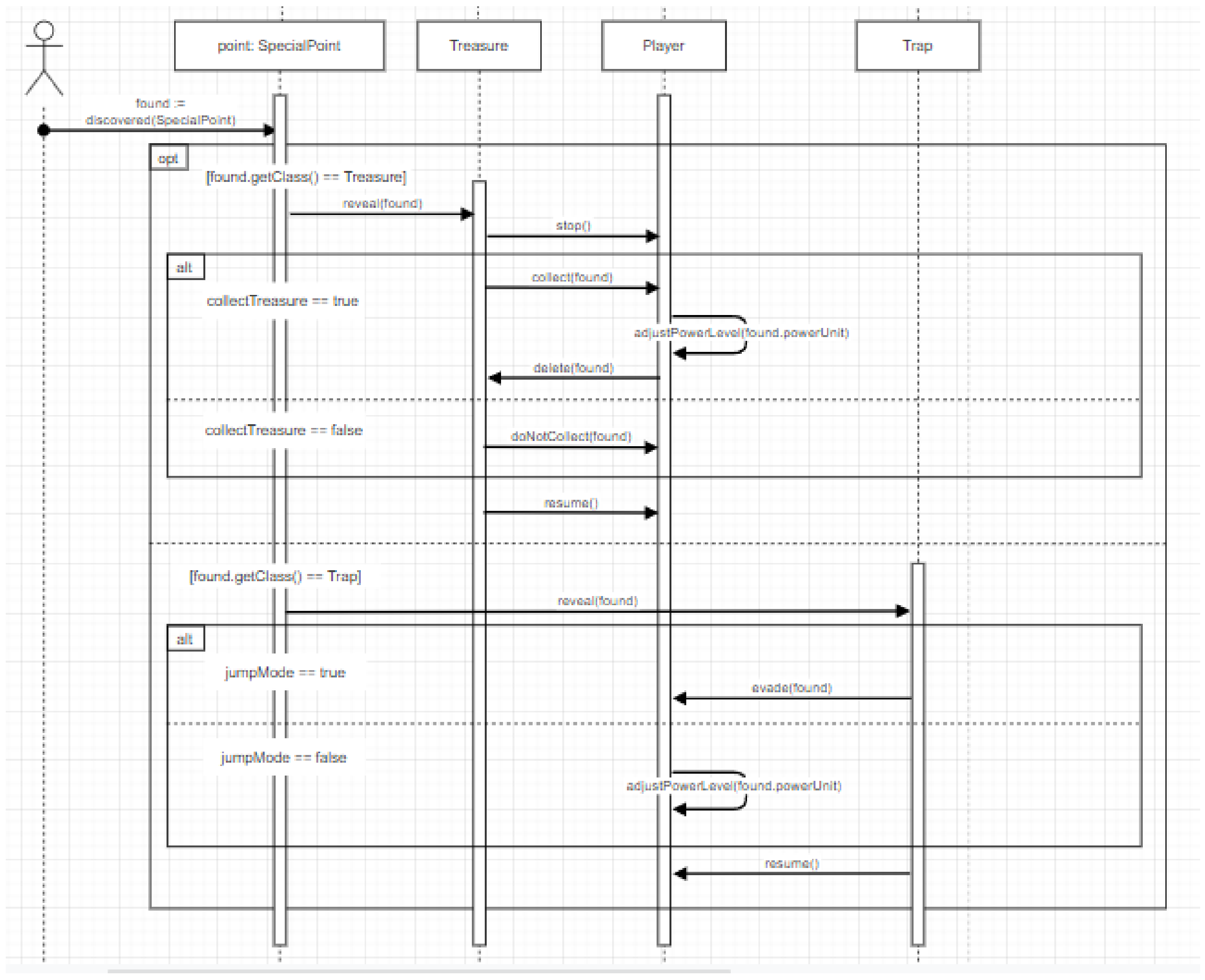
Immutable: The value of each type of treasure must be immutable to ensure the correct number of power units are always being transferred to the user’s avatar when collected.

Player-Role: Aliens can have many different states/behaviors depending on their vicinity to the player and their current power level; they can be in shooting mode if they are within a specified vicinity to the player allowing them to shoot assuming sufficient health, and will engage in following mode if the player extends the distance separating them to a certain measure. If an alien has less than half its initial health it transitions into escaping mode, where it tries to flee from the user’s avatar until a certain separation distance has been achieved.

Assumptions:

The system assumes that there is minimal input-lag to ensure that the user has a predictable gaming experience. The system assumes that the coordinates of the user’s avatar are being compared with the SpecialPoint objects in the level currently being traversed in real-time to ensure that entry points, exit points, treasure, and booby traps are revealed and made interactive as soon as they are discovered.

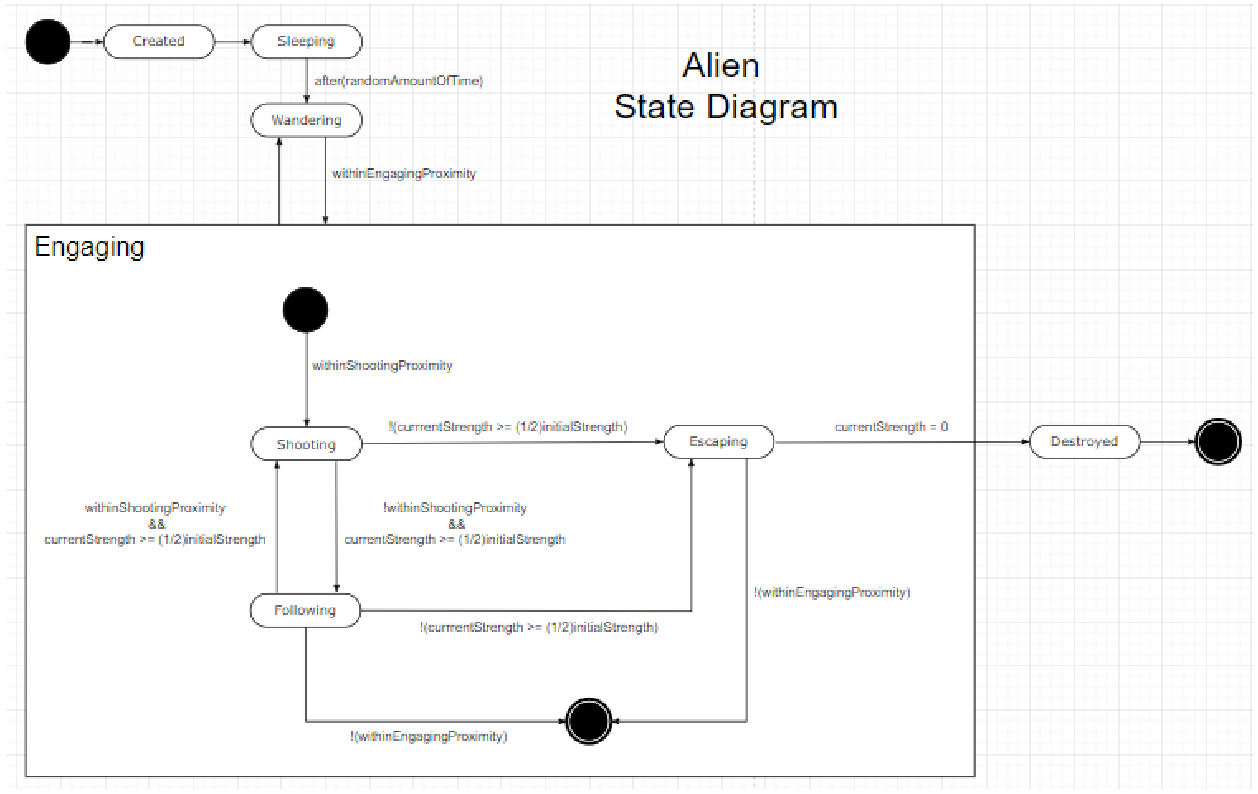
PART 2: PLAYER-TREASURE-TRAP SEQUENCE DIAGRAM



The Sequence Diagram development tool “Diagrams.net” was used to generate the above sequence diagram.

One of the important implementation considerations made was that of when/if the SpecialPoint object should be removed from the level. Because, assuming the SpecialPoint object is of type Treasure, the problem description stated that a player could *choose* to collect the treasure. Thus, it should remain at its location in perpetuity assuming the player chooses to leave it such that they may return to it later to gain power units after potentially taking damage from other enemies. The treasure object in the diagram therefore cannot have a strict lifetime, since its lifetime depends on the player’s decision, and thus I made it so that the object persists regardless. On the other hand, the problem description did not mention if booby traps could be dismantled or destroyed, by the player or by the game after damaging a player. Hence, I assumed booby traps would persist in perpetuity regardless of if a player evades them or not, so the player always has the potential to be hurt by them.

PART 3: ALIEN STATE DIAGRAM


Alien
State Diagram
Created
Sleeping
after(randomAmountOfTime)
Wandering
withinEngagingProximity
Engaging
withinShootingProximity
Shooting
!(currentStrength >= (1/2)initialStrength)
Escaping
currentStrength = 0
Destroyed
withinShootingProximity && currentStrength >= (1/2)initialStrength
!withinShootingProximity && currentStrength >= (1/2)initialStrength
!(withinEngagingProximity)
Following
!(currentStrength >= (1/2)initialStrength)
!(withinEngagingProximity)

The State Diagram development tool "Diagrams.net" was used to generate the above state diagram.

PART 4: FUNCTIONAL/NON-FUNCTIONAL REQUIREMENTS

Example of a well-written requirement:
"The Online Banking System shall allow the Internet user to access her current account balance in less than 5 seconds."

Functional Requirements:

1. The system shall allow the user to input the name of their avatar into the system using a keyboard within 5 seconds of starting the game.
2. The system shall allow the user to pause and resume playing the game instantly following the pressing of a button dedicated to pausing and resuming.
3. The system shall output the name, game level reached, power level achieved, and number of aliens destroyed to a leaderboard within 5 seconds following either the completion of the game or their avatar's destruction.

4. The system shall store the maze of paths corresponding to each level of the game.
5. The system shall automatically update the power level of the user’s avatar within 3 second following changes from its current state as a result of treasure collected, alien attacks, and booby traps suffered.
6. The system shall graphically reveal SpecialPoint objects in the level within 1 second of discovering that the avatar’s and object’s respective locations are equivalent.

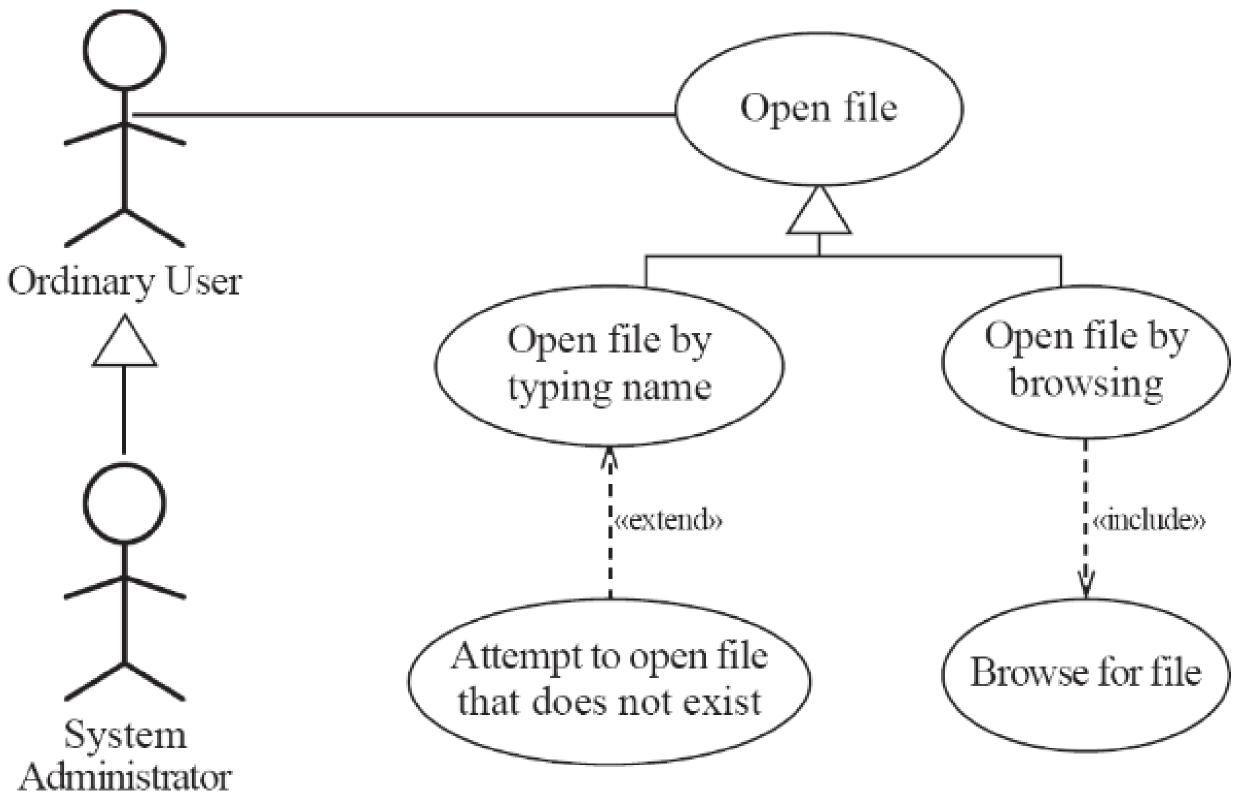
Non-functional Requirements:

1. The system shall actively store only the paths corresponding to the preceding, current, and successive levels traversed/to be traversed by the user’s avatar to reduce the total resources required for the game to be played without disrupting the continuous flow of gameplay.
2. The system shall allow interactions between a user’s avatar and the environment to be processed within a 1-second response-time since the initiation of the interaction.
3. The system shall store treasure types as enums to constrain the properties of each treasure item, this will allow for seamless expandability to new treasure types that can be introduced to the system.
4. The system shall store each level as an assemblage of paths to streamline the design and generation of other levels.

PART 5: USE-CASES & USE-CASE DIAGRAM WITH DESCRIPTION

Outline of a single use-case followed by a use-case diagram with a two-column format description:

A. Name: Short and descriptive.
B. Actors: Types of users that will do this
C. Goals: What are the actors trying to achieve?
D. Preconditions: State of the system before the use case.
E. Summary: Short informal description.
F. Related use cases.
G. Steps: Describe each step using a 2-column format.
H. Postconditions: State of the system aftercompletion.



Use case: Exit car park, paying cash

Actors: Car drivers

Goals: To leave the parking lot after having paid the amount due.

Preconditions: The driver must have entered the car park with his or her car, and must have picked up a ticket upon entry.

Summary: When a driver wishes to exit the car park, he or she must bring his or her car to the exit barrier and interact with a machine to pay the amount due.

Related use case: Exit car park by paying using a debit card

Steps:

Actor actions	*System responses*
1. Drive to exit barrier, triggering a sensor.	2a. Detect presence of a car. 2b. Prompt driver to insert his or her card.
3. Insert ticket.	4. Display amount due.
5. Insert money into slot.	6a. Return any change owing. 6b Prompt driver to take the change (if any). 6c. Raise barrier.
7. Drive through barrier, triggering a sensor.	8. Lower barrier.

All of the possible use cases of the system consists of: User begins game, user crosses paths with treasure and collects it, user crosses paths with treasure and does not collect it, user crosses paths with booby trap and is not in jump mode, user crosses paths with booby trap and is in jump mode, user crosses paths with alien, user shoots at alien, alien shoots at user, alien pursues user, user chases fleeing alien, alien destroys user, user destroys alien, user enters path, user exits path, user moves onto next level, user returns to previous level, user pauses game, user resumes game, user is destroyed.

Use case: User's avatar crosses paths with treasure and collects it.

Actors: User's avatar.

Goals: To reveal the treasure to the user and permit interaction with it.

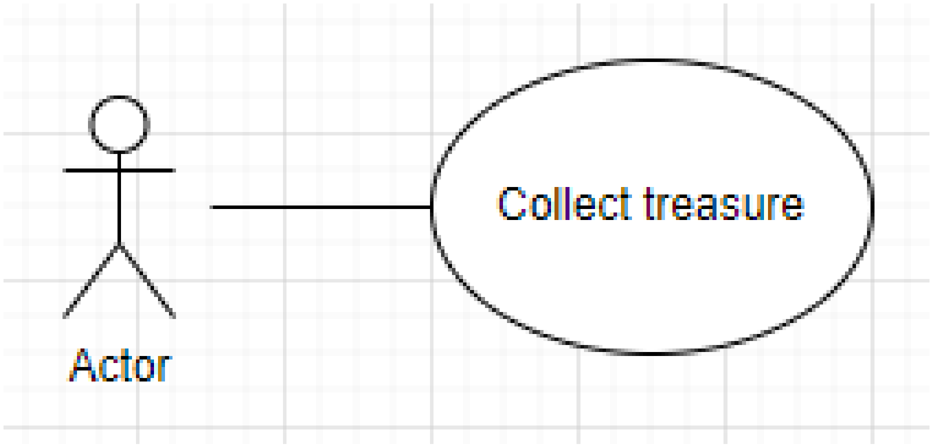
Pre-conditions: The user's avatar must have the same coordinates as the treasure.

Summary: When a user's avatar crosses paths with treasure, the treasure should be interactive and shall permit the user to collect it. If collected, it is then converted to its power unit value, added to the user, and vanishes from the level. If not collected, it remains visible and interactive until collected by the user's avatar.

Related use case: When the user's avatar discovers hidden entrances to secret paths.

Steps:	*System responses:*
1. User moves avatar across point corresponding to treasure.	
	2a. Detects coordinate equivalence between user's avatar and treasure. 2b. System stops user from moving avatar. 2c. Treasure is revealed and system prompts user with the choice to interact by collection of treasure.
3. User presses button to confirm choice to collect treasure.	
	4a. Increases the user's avatar's power level by the power unit value of the treasure discovered. 4b. Graphically and functionally removes treasure from the level so that it cannot be re-discovered and reaped again.

	4c. Returns motion control to the user.



Git Repo Link: https://github.com/SEG2105-uottawa/Assignment3_8791694

UMPLE Diagram:
https://cruise.umple.org/umpleonline/umple.php?model=2112081csuogzy8dz2b