Game Title: Corrupted Labyrinth

Candidate Number: 310990

2025-12-18

Contents

1. Problem Statement & Overview

The problem states that an adventure game must be created where a player is "moving through space", where it features different mechanics and an end goal in the game. For this project a player is placed in a digital labyrinth after being locked into their VR headset.

The player has to move through this corrupted labyrinth in search of a means to escape and leave back to the real world. There are 11 locations in the game that the player can travel through where they will find items and puzzles to guide their way. Monsters may be encountered in which players can use weapons to defeat them - they get stronger as the player progresses. Keys can be used to unlock certain exits that are locked and the final exit requires two keys.

2. Instructions

Firstly, open the folder in terminal by right clicking the `game_code` and click `open in terminal`. Ensure Python 3.10+ is installed (this can be checked with `python --version`). If on Windows, install the curses compatibility package using `pip install windows-curses`. Then, type: `python game.py` to launch the game.

An introduction message will play where the space bar can be used to skip. After this, the '/' key can be used to display all commands / key inputs. The player can start by moving with the arrow keys and scanning the room (using the key 'r') for anything that's inside (e.g. items). In the starting room, items can be picked up using 't' and an option to equip for the player to use as a weapon and to heal. The player then needs to traverse the game space where doors may be unlocked, monsters may be encountered and puzzles that can be solved for rewards. The end goal is to find the exit - the system kernel.

3. Game Map

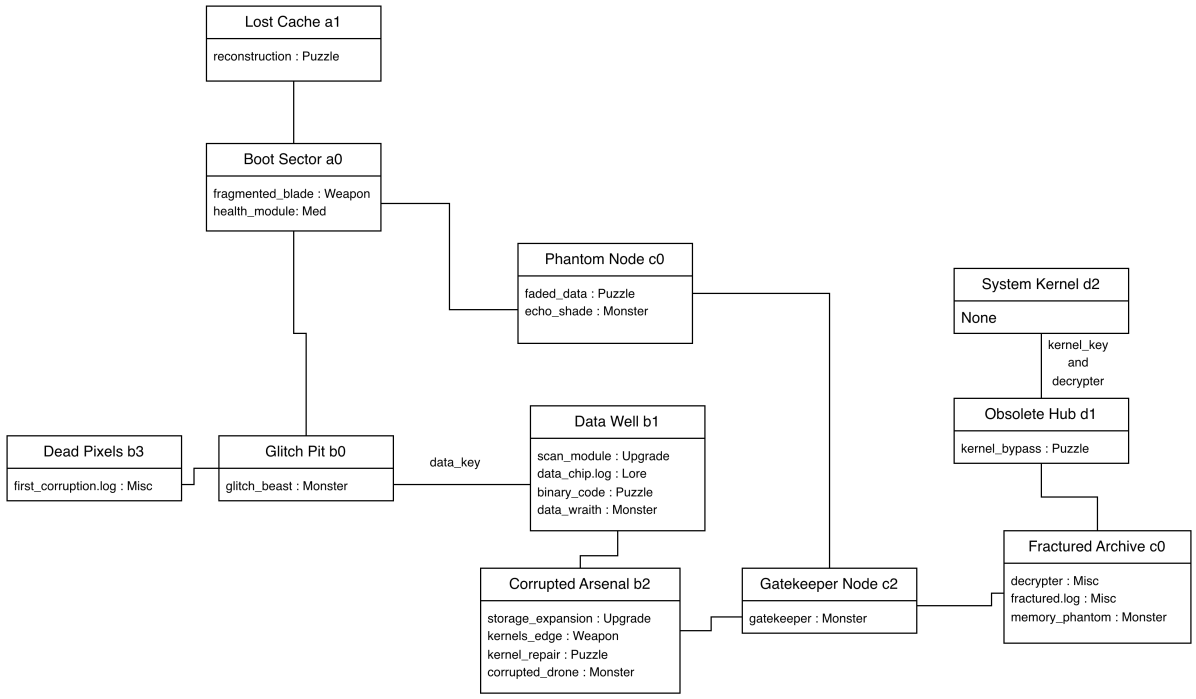

Figure 1: Game map featuring items and monsters in each room

4. UML Diagrams

4.1. Entity

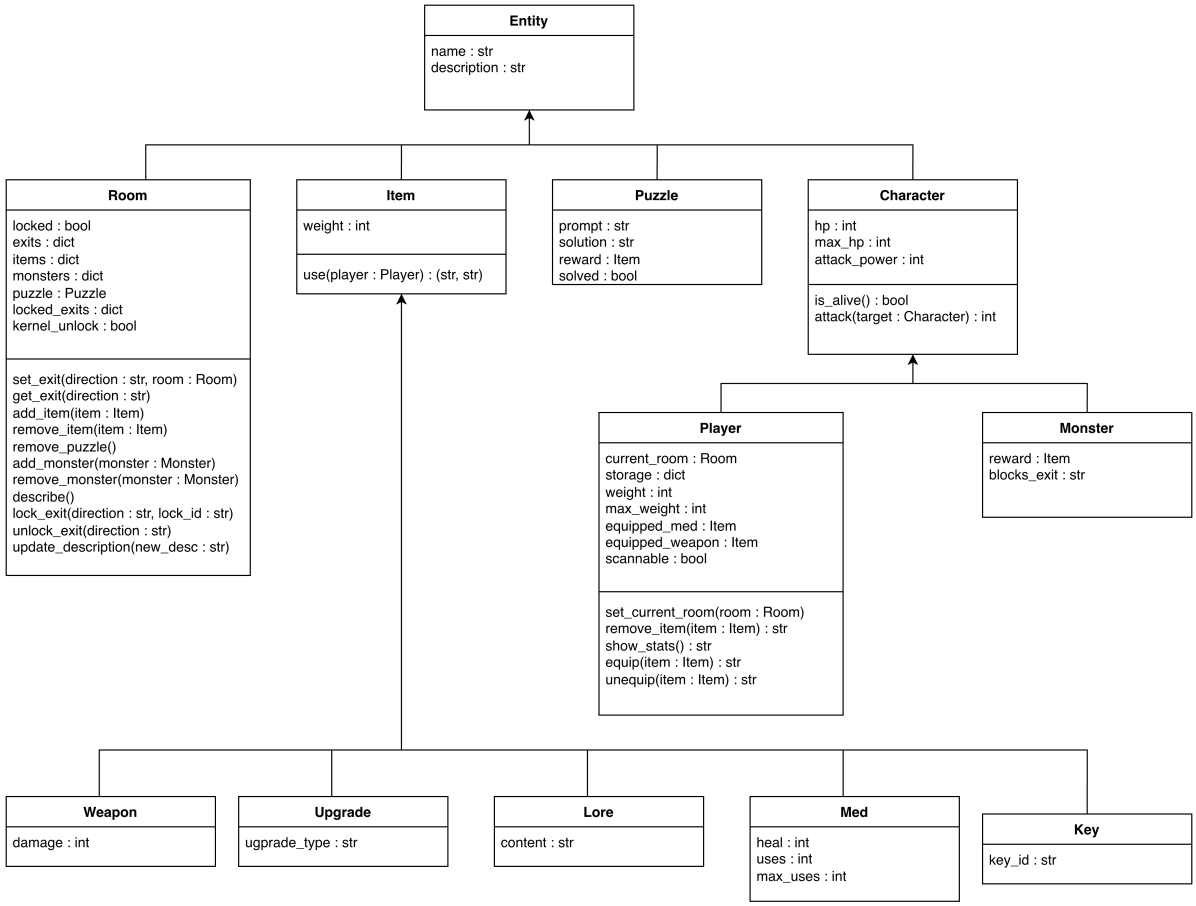

Figure 2: Entity UML showing related classes

4.2. Game Controller/Systems

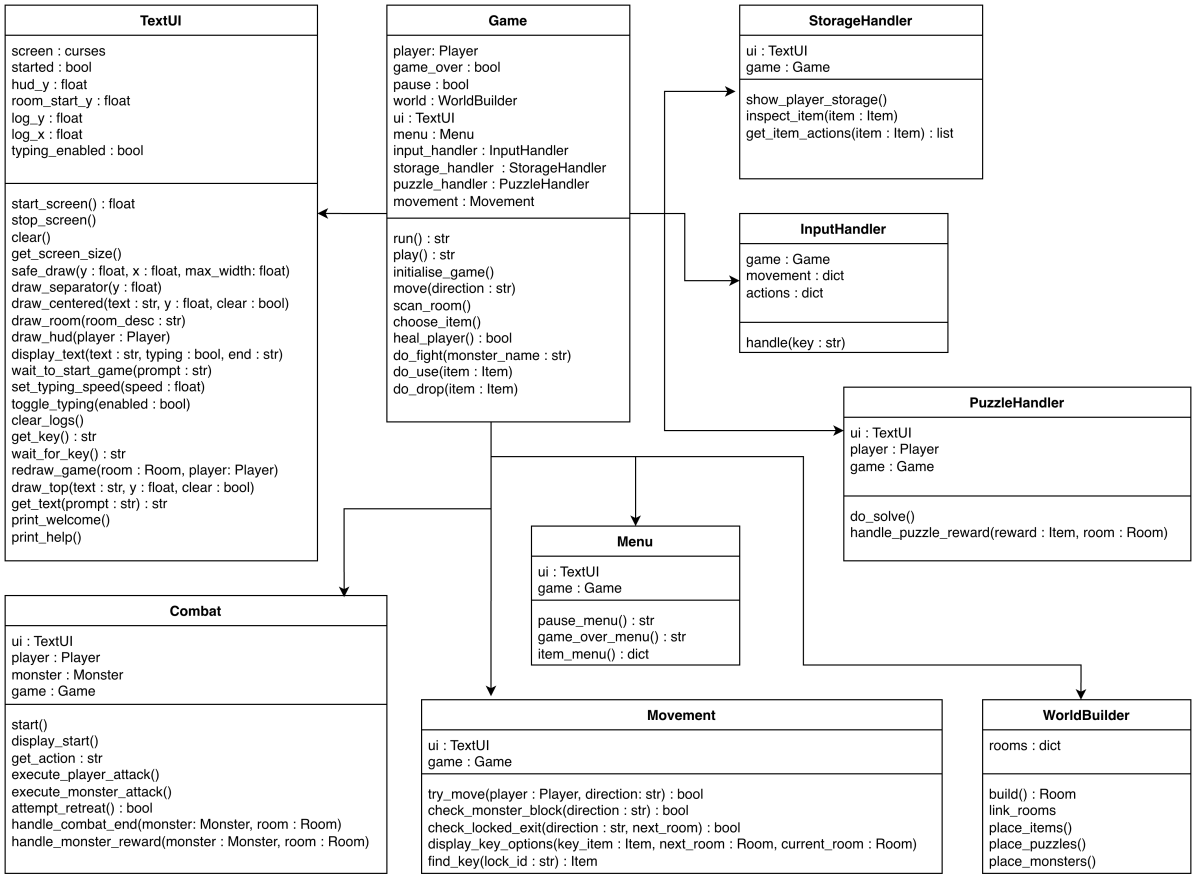

Figure 3: Main controller UML showing relationship between the Game class and other systems

5. Modifications

The starting code was modified to ensure better designing principles. The Game class was updated so that it orchestrates the gameplay at hand, by using class systems in place - it mainly handles the player's actions and movement. It also includes a new feature where the user can pause the game at any point and display the pause menu. Messages and printing are moved to the TextUI class and commands are processed in the InputHandler class.

The TextUI class was changed so that it allows the terminal to be used as a single screen that constantly refreshes the screen after every world event. It displays the room contents, the player's stats as a persistent HUD, and the results for the player's actions or event.

The Backpack class refactored to a new name: StorageHandler. This was more appropriate its use case; handling the player's storage. It was changed such that it handles displaying of the storage and whether the user wants to drop, use, or equip an item. Adding and removing item methods were moved to the Player class.

The Room class was updated to include more features: adding entities (monsters, items, puzzle), locking and unlocking exits and for updating the room description (for dynamic room changes).

5.1. New Classes

5.1.1. Entity

Represents any object that is within the world, holding Character, Item, Puzzle, and Room objects.

5.1.2. Character

This represents a character in a game that has health points and attack power. These characters can attack another character that is chosen, reducing the other character's health points.

Player and Monster classes are subclasses of this class. The Player class is the user controlled character who can move through rooms, use items, pick up items and rewards. The Monster class can block an exit in a room and can drop a reward for the Player to pick up.

5.1.3. Item

This type of class can be 'held' by the player or in a room. These items have several subclasses: Key, Lore, Med, Upgrade, Weapon; they all inherit the item weight and its use method. The use method is over-ridden by its subclasses: Key unlocks exits, Lore describes the story content, Med heals the player's HP, Upgrade changes the player's stats (e.g. max capacity of the player's storage is increased), and Weapon is an item used by the Player to fight against monsters.

5.1.4. Puzzle

Another entity in which holds a prompt for the user to solve. If the user gets the answer correct, then the player is rewarded an item (if the Puzzle holds one).

5.1.5. System Classes

There are several systems put in place for the game to function properly; these are: Combat, InputHandler, Menu, Movement, and PuzzleHandler. The other classes, StorageHandler and TextUI, are described previously.

The Combat class handles combat between the Player and a given Monster, and ends the game when the player dies or rewards the player after the Monster dies.

The InputHandler class handles keys in the game loop, allowing the user to do several actions within the game.

The Menu class handles different menus within the game, such as a game over menu for when the Player dies; giving the option to restart or quit the game.

The Movement class handles Player movement within the game space; checking for locked exits or blocked exits from monsters.

The PuzzleHandler class allows the user to solve a puzzle that is in the current room and rewards the Player if there is one.

5.1.6. WorldBuilder

This class builds all the rooms in the game and links them together. It adds items, puzzles, and monsters into the rooms where each monster and puzzle may contain a reward item. Certain exits are locked and/or blocked by a specific monster.

6. Design Features

6.1. Terminal-Style UI

The game features a typing animation for when text is displayed to the user, and the animation can be skipped with the space bar. It uses the Curses library to execute this.

6.2. Persistent HUD

The HUD that displays the player's stats can be displayed at all times during the game.

6.3. Puzzles

Certain rooms contain puzzles for the player to solve, and once solved, the player is rewarded an item. An example item is a key that the player can use to unlock a secret exit in the starting room.

6.4. Movement

The starting code was changed so that the user won't need to type commands and then press enter to input. Instead, when a certain button is pressed, immediate feedback is given where the screen is updated accordingly. This includes movement where the user can use the arrow keys to navigate the game space; replacing the need to type `go [direction]`.

6.5. Pause/Game Over Menus

When the game ends/player dies, the user is given the option to quit or restart the game completely. These options are also available to the user throughout where the user can also pause the game to do these actions.

6.6. Blocked/Locked Exits

Room exits can be blocked by a monster where the player has to fight to get through. In addition, they can also be locked and the player has to acquire the key for the door to unlock.

6.7. Automation

When the player picks up an item that is equippable, the game asks the user if they want to equip the item straight away. Also if they try to go towards an exit that is locked and the player has the key for that door, they can be asked to unlock using that key.

6.8. Combat

When engaging with a monster, a turn based combat occurs. The player starts first where they can either attack, heal or retreat. The player attacks only with their equipped weapon and can only heal with their equipped healing item. The player also only has a 60% chance to leave the battle and if they fail, the monster attacks them.

6.9. Storage Management

The player can organise their storage by dropping items in a room, equipping them or using the items. Items have certain effects in the game such as being able to read story content.

7. Problems

The main problem that was encountered was converting typed commands to single user key presses; this resulted in many bugs being encountered while testing, such as not being able to move in a certain direction. Another problem was creating the pause function, discovering a bug where the user had to press ESC twice to pause the game. This was because the curses library waits for another button press after the first ESC press, and this was fixed by preventing it from waiting. Other issues involved the player's actions where items were dropped and not put back in the room, the player could enter rooms when they were locked, the player couldn't heal with an item that's equipped, and that the player couldn't equip weapons and healing items. To resolve these problems I used unit testing and system level testing for the game.

8. Testing

8.1. Unit Testing

Unit testing was used to test and logging player and game events:

Attacking the player and monsters were checked if it affects the HP of the characters.

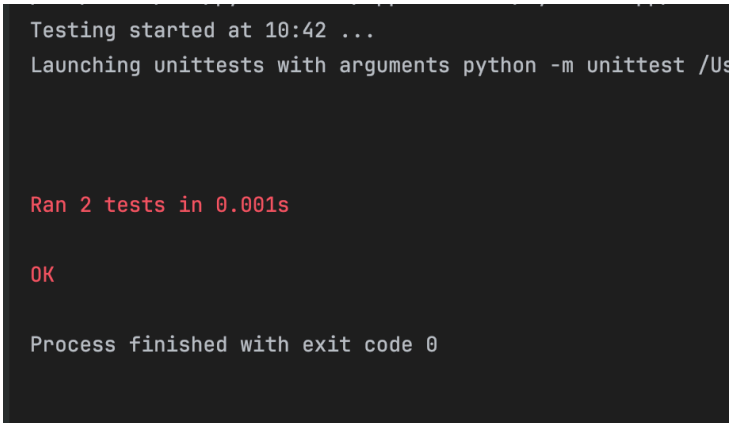

Figure 4: Unit testing combat results

Checking if picking up a certain item stores it inside the player's storage when within capacity, and if over the capacity, the player is unable to.

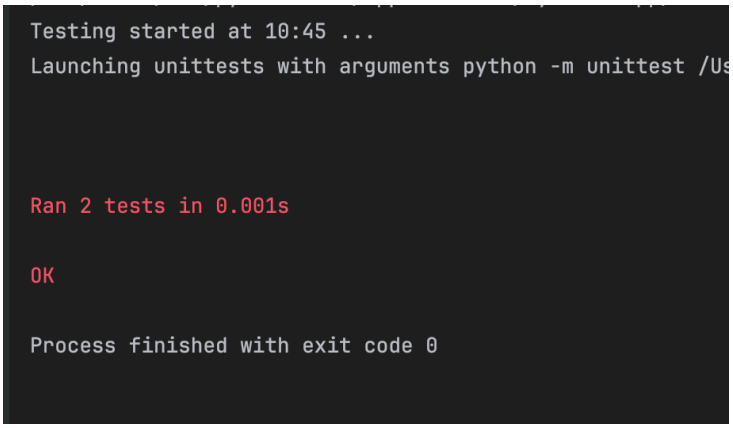

Figure 5: Unit testing storage results

Testing if the player can heal, equip and un-equip weapons where attack power and weight is updated, and the same for healing items. It also tests when the player is alive or dead by checking their HP.

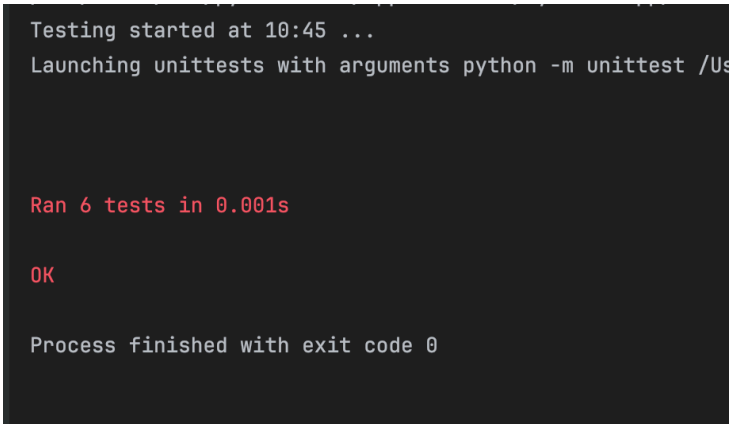

Figure 6: Unit testing player results

Checks for puzzle solving in the game, True if the answer is the solution, False if not.

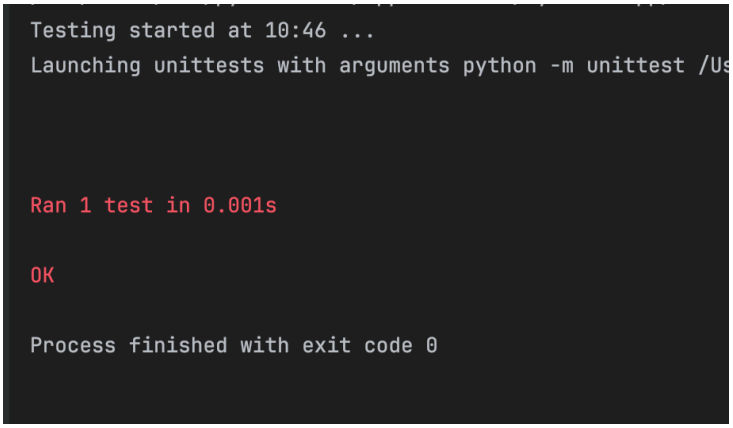

Figure 7: Unit testing puzzle results

8.2. System Level Testing

No.	Description	Inputs	Expected Outputs	Actual Outputs	Pass/Fail
1	User skips displayed welcome message.	Press space bar twice.	Progress to starting room.	Progressed to starting room.	Pass
2	User stores an item from starting room.	Press in order ‘t’, ‘1’, ‘1’, ‘s’	Chosen item is in storage	Chosen item is in storage	Pass
3	User can move to a valid room.	Arrow keys	Prints ‘moving “direction”…’ and then room changes description.	Prints ‘moving “direction”…’ and then room changes description	Pass
4	User moves in an invalid direction	Arrow keys	Prints ‘moving “direction”…’ and then prints “You can’t go that way!”	Prints ‘moving “direction”…’ and then prints “You can’t go that way!	Pass
5	User moves towards a locked exit but does not have the key	Arrow keys	Prints ‘moving “direction”…’ and then prints “The path to “room name” is locked (“key_id”)”	Prints ‘moving “direction”…’ and then prints “The path to “room name” is locked (“key_id”)”	Pass
6	User moves towards a locked exit but has the key	Arrow keys and ‘1’	Prints ‘moving “direction”…’, then asks the user to use the key to unlock, and	Prints ‘moving “direction”…’, then asks the user to use the key to unlock, and	Pass

No.	Description	Inputs	Expected Outputs	Actual Outputs	Pass/Fail
			room updates.“	room updates.“	
7	Player dies during combat with a monster	Using NUM keys to attack monster	Game over menu displays	Game over menu displays	Pass
8	Player receives a reward after killing monster	Using NUM keys to attack monster	Reward item is in player’s storage	Reward item is in player’s storage	Pass
8	User can auto equip a stronger weapon	Using NUM keys to equip	Weapon is displayed on HUD under WPN	Weapon is displayed on HUD under WPN	Pass
9	User can pause and unpause the game	Pressing ESC and ESC again	Pause menu appears and disappears revealing the room again	Pause menu appears and disappears revealing the room again	Pass
9	User can view storage and see item weights	Pressing ‘s’	Storage appears with a list of items and their weights	Storage appears with a list of items and their weights	Pass

8.3. Logging

I used logging to track the user's actions as shown in Figure 8. This allowed me to track where the player was at when the game showed a bug/error.

```
INFO:root:User starts a new game
INFO:root:Player picked up fragmented_blade
INFO:root:Player equipped fragmented_blade
INFO:root:Player picked up health_module
INFO:root:Player equipped health_module
INFO:root:Player moved south to glitch_pit
INFO:root:Player moved north to boot_sector
INFO:root:Player moved south to glitch_pit
INFO:root:Player moved west to dead_pixels
INFO:root:Player picked up first_corruption.log
INFO:root:Player moved east to glitch_pit
INFO:root:Player starts fight
INFO:root:Player heals
INFO:root:Player uses data_key
INFO:root:Player picked up scan_module
INFO:root:Player uses scan_module
INFO:root:Player heals
INFO:root:Player starts fight
INFO:root:Player heals
INFO:root:Player picked up debugging_lance
INFO:root:Player equipped debugging_lance
INFO:root:User starts a new game
INFO:root:Player picked up health_module
INFO:root:Player equipped health_module
INFO:root:Player picked up fragmented_blade
INFO:root:Player equipped fragmented_blade
INFO:root:Player moved south to glitch_pit
INFO:root:Player starts fight
INFO:root:Player heals
INFO:root:Player heals
```

Figure 8: Simple logging for the game