ItoP: Assignment 3

Introduction

In this assignment you will use a provided OOP framework to build the logic for a character in a simple game. The goal of the assignment is to show you how a well-designed OOP framework can be easy to extend and use.

Context

It's so dark. How did you get here? What time is it? You sit still and wonder what's going on.

Oh no. Now what. You hear some footsteps to the left. A hushed "braaaiiinnnnsss" echoing throughout the room. Great.

You sit still for a little longer and listen carefully. The place sounds infested. You can't see anything, but you get up and start running. Figuring out what to do next. You're not about to die here.

Sorry for that, I never was good at creative writing...

Directions

Your job is to describe what you're going to do to survive the zombie apocalypse by filling in the code for the `PlayerCharacter` class in the `as3.py` file. The only goal is to remain alive until the end of the game, even if the zombies are still alive.

The game plays out frame-by-frame for a fixed number of frames (100 in the example code), where in each frame the player and zombies can perform a single action. The following actions are available:

- Move: walk or run away.
- Heal: recover up to 25% of your health.
- Scan: listen for zombies around your location.
- Attack: sneak attack a zombie.

Instructions continued on the next page

Moving

The game takes place on a 2D, finite grid. The player and zombies must be in integer positions on the grid. Each frame, you may move along the grid up to 3 units of Manhattan distance (note: this is the number of grid spaces) away. It is your job to pick a location that is in bounds. You can access the size of the room using the `MapView` object. The valid locations are anywhere in the intervals [0, width) and [0, height). Note that if the room size returned in a `MapView` is (5, 7), then location (5, 7) is **invalid** in both the x and y coordinates, as the positions are zero-indexed. If you pick an invalid location or move too far, the `IllegalMovementException` will be raised.

To perform a move action, return a `MoveEvent` object with the coordinates of the new position specified.

Healing

When you perform a heal action, your player restores up to 25% of their maximum health. If you heal when you have more than 75% of your maximum health remaining, you will only be restored back to original 100%. You have 1000 health points and the zombies have 100 each. You may only heal up to 5 times during the duration of the game. If you try healing more than 5 times, you will waste an action.

To perform a heal action, return a `HealEvent` object.

Scanning

To know anything about your surroundings, you must spend time scanning. Your player keeps track of the room in a list of `ScanData` objects. When you perform the scan action, this list gets updated with the latest information. The `ScanData` objects contain the positions and IDs of zombies. The positions can be helpful in determining where to move, when to attack, or anything else you see fit. The main purpose of the ID is so that you can perform an attack action in the future (see Attacking).

You will have access to your scanned data even if you do not perform a scan in the active frame. However, the data is only updated when you perform a scan. Since zombies can move around freely, the data may become stale if you don't scan frequently enough. Furthermore, scanning only covers $1/4$ of the map, so if you move then your data may also become stale. You may choose to use old scan data, but it may affect your strategy.

To perform a scan action, return a `ScanEvent` object.

Instructions continued on the next page

Attacking

Finally, you may attack any zombie *that you currently have scan data for*. If you try attacking a zombie that is not currently in your scan data list, you will waste an action. The damage dealt to the zombie is computed as follows:

$$\frac{\text{player health}}{e^{\text{distance between player and zombie}}}$$

In other words, the damage is directly proportional to how much health the player has and exponentially decreases by the distance to the zombie. Another reason you may want to scan frequently is to make sure you aren't wasting an action attacking a zombie that has moved far away from where you last scanned its location.

To perform an attack action, return a `AttackEvent` object with the ID of the zombie you want to attack.

Other rules

In addition to being attacked, players and zombies are subject to being worn down. Each frame in which the player or zombie *does not perform a heal action*, they will lose 2% of their maximum health. This decrease in health is dubbed 'aging'

Within a frame, events are processed in the following order.

1. The player and zombies heal.
2. The player and zombies attack one another.
3. The player and zombies move.
4. Scan data for the player and zombies is updated.
5. The player and zombies age.

Remember, in a given frame the player or zombies may only perform one action each. The order in which actions are processed is relevant in situations like the following. In frame N, the player, who is at location (1, 1), attacks a zombie who is also at location (1, 1). In the same frame, the zombie moves to location (1, 2). In this case, the zombie will still receive a direct hit from the player because attacks are resolved before movement.

When the player or zombie reaches 0 health, it is removed from the room.

Instructions continued on the next page

Documentation

The starter files are heavily documented, so you can (and probably should) refer to them. The structure of the code is fairly verbose, and it could probably have been written in 100 lines of undocumented Python, but instead is almost 800 lines. The purpose, however, is to show you one way to write effective OOP code. As I mentioned in lecture, OOP is an investment for the lifetime of the project.

I expect that a significant part of your time on the assignment will be spent reading through the existing code. Once you have an understanding of the code, writing the player's behavior should be simple.

To go through the code, it may be useful to start in the `ZombieCharacter` object in `as3.py`. I've defined a zombie that roams around aimlessly, occasionally healing itself when it has aged too much. Then you can look in the main function of the `main.py` file. You will see the objects used to manage the game logic. After that, you may want to look at the definition of each of the types of events in the `gamelib.py` file (denoted with a large comment). Finally, you can look at the abstract classes defined in the last section of the `gamelib.py` file (also denoted with a large comment). The abstract classes may seem unintuitive and overly complex, but it's a structure that's fairly common in game development. Recall that it is Python convention to prefix names with an underscore if it should be private. Keep that in mind when reading the code and writing your player.

Starter files and running your program

The starter files can be downloaded from Canvas as a file called `as3.zip`. There are four files in the zip: `as3.py`, `gamelib.py`, `main.py`, and `README.txt`. You will write all of your code in `as3.py` where the comments direct you to. You should also fill out the information in `README.txt`.

To run the program, type in the command '`python3 main.py`'. The output shows the events being performed and the state of the room frame by frame, with a 0.5s delay between frames. The delay is just there to make the game seem somewhat animated. There are a few command line options, which can be seen by adding `--help` to the end of your command. When `--no-print-events` is provided, the events are not printed each frame. When `--no-print-map` is provided, the room is not printed each frame. `--sleep=NUM` can be used to specify the time between frames, in seconds. To remove "animation" completely, you can use `--sleep=0`. The flags can be combined arbitrarily.

Grading

Grading for this assignment is unusual because you are given free reign to write the behavior of your player as you please (within the constraints of the game). When I grade the assignment, I will execute multiple variants of the game multiple times and count the number of games in which your player survives to the end. The variants may include zombies with different behavior, different room sizes, etc. The purpose is just to test that you put some thought into the behavior of your player. I will not set up unreasonable situations or make significant changes to the mechanics.

On that note, you may want to test your player's performance in different scenarios by modifying the behavior of the zombies to do something more interesting, even though it isn't required.

Instructions continued on the next page

Submission

You should submit a zip file to Canvas called `as3.zip` with the same structure as the starter files. **The following conditions will result in a 0 on this assignment:**

- `README.txt` is incomplete or incorrectly filled out.
- The structure of the submitted `as3.zip` differs from the original structure.
- The program does not terminate in a reasonable amount of time.

Extensions

There are no direct extensions to this assignment, but you may put in as much thought into your player as you want. The game framework is fairly robust (at least, I hope it is), so you can experiment with a wide variety of complex behaviors - both for your player and the zombies. You may also add more types of characters to the game, which is simple to do (just add a new object that inherits from `ICharacter` and then initialize it in `Map`). It is also straightforward to add more event types (just create a class that has an `executeEvent` method and then process it in the `ZombieHunter` object). Hooray for OOP!

When I grade the assignment, I will replace `gamelib.py` and `main.py` with fresh copies, so make sure you don't leave anything in your code that would break without your own modifications to these files.