INF-1400: Objektorientert programmering

Mandatory Assignment 3

Narongchai Cherdchoo & Daniels Sliks

April 5, 2024

1 Introduction

This report covers the implementation of a clone Mayhem game using object oriented programming (OOP). Mayhem is a simple 2D game where you control a rocket ship by boosting and changing direction, complete objectives while trying to avoiding obstacles whilst battling gravity. The clone will be implemented in python using the Pygame module. Ours will be based of the same concept, but where there is two players that battle each other by trying to shoot each other and not crash in objects, while avoid running out of fuel.

2 Requirements

The following the requirements for the assignment:

- Two spaceships with four controls.
- At least one obstacle in the game world.
- Spaceship collides with walls, obstacles and other spaceships.
- Gravity that acts on spaceships.
- Each player's score is displayed on the screen.
- Each spaceship has a limited amount of fuel and can be refueled.

3 Technical Background

In this project the entire game is represented by classes. Each have methods that enables objects to interact with each other. By structuring our code following OOP principles, we aim to make the code more organized, simple to change, adjustable and flexible for implementing new features.

3.1 Running The Game

The game is executed by running the `main.py` file, which then opens a program window. From there `Player 1` is controlled with '`W`' to boost in the direction of the ship, '`A`' and '`D`' to turning the ship left or right respectively and '`S`' to shoot bullets. `Player 2` controls given in the same order are '`Arrow Up`', '`Arrow Left`', '`Arrow Right`' and '`Arrow Down`'. The goal is to shoot each other to gain points and avoid dying by either crashing or getting shot. Fuel is also lost by boosting and can be regained by landing on the fuel pads (gray platforms).

3.2 Object-Oriented Programming

Is a programming technique where code is structured in a way that generalises various concepts into objects and allows one to define various methods that modify these object in useful ways. Key benefits behind this is inheritance and standardization, where separate objects can share similar traits but be viewed as distinct types, which prevents code from being repeated multiple places and can make code more clear.

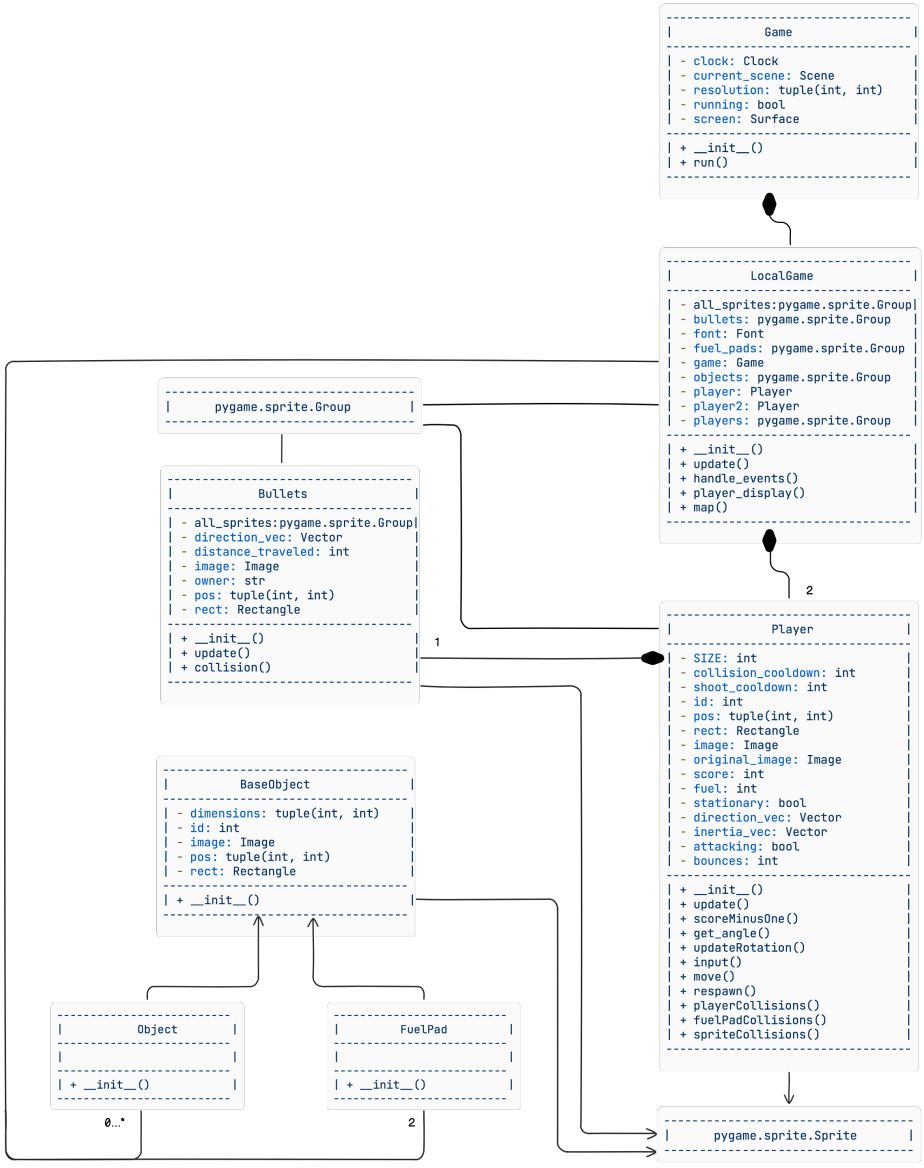

Figure 1: UML of The Program

4 Design & Implementation

The game is represented by a single game object containing the `pygame` loop. From there another object is stored that represents the current scene being displayed, its main responsibilities is to represent the current match by keeping track, updating and drawing the sprites within it. With this setup, multiple scenes could be defined and be switched between, like for example a main menu screen and different game modes. In this case only a local game object is defined, with the placement and definitions of objects and players predefined at an initial state. All sprites in the game inherit from `pygame.sprite.Sprite` class which are stored in multiple `pygame.sprite.Group` objects allowing for easy updating and rendering of each sprite.

4.1 Player

The player class is responsible for representing each player. It responds to input from the user, has limited fuel that decreases when boosting, which can only be regained on a fuel pad. The player has simple implementation for acceleration when moving, as well as air resistance and gravity for much smoother movement. For the most part collision will cause the player to respawn and lose a point, except for when landing on a fuel pad with a straight enough angle and slow speed. When a player shoots, a Bullet object is created with it as the owner, this way points can be awarded when hitting another player.

4.1.1 Movement

Player rotation is achieved by using pygame's vector class. A vector object is stored within a player (direction vector) which can turned left or right by a set amount of degrees using vector methods. This vector representing the direction a player is facing can be translated from vector angles to normal angles. Which then can be used to rotate the player's image in the direction matching the vector.

Player movement is all represented by single vector (velocity vector) which is modified based on some simple logic. Those being: player input, air resistance and gravity. When a player boosts the velocity vector is slightly moved towards the direction vector. The direction vector is scaled to be the length of the max speed such that the velocity vector can never get past this limit. Along side, each frame the vector is also reduced in length, simulating air resistance and causes the player to eventually slow down when no longer boosting. The velocity vectors `y` value is also constantly being reduced a set amount simulating gravity acceleration. A terminal velocity is defined that prevents the velocity vector from ever gaining a length greater than it.

Player Location is simply updated by adding the `x` and `y` values of the velocity vector to the corresponding position values to the player, which is used later by the draw method.

Statistics Each players contains attributes for their score and the amount of fuel that updates constantly during the runtime, we can use these information to print the player stats information on the screen as a simple display.

4.1.2 Player Collision

Since every sprite in the game inherits from pygame's sprite class, they can be added to another class called Group. This then allows for the use of `spritecollide()` function to get a list of

sprites the a given sprite collides within the group. From there logic can be added based of what type object it collided with.

General Collision causes the player to respawn back to their beginning spot and lose a point. If a player collides with a bullet it doesn't own, it not only respawns the player and removes a point, but also adds a point to who ever owned the bullet.

Fuel Pad collision is a bit special, here after checking that a player has a sufficiently slow speed and is oriented within a certain range it doesn't crash the player and it can land. When landing it will bounce three times, each time losing speed until it get frozen on top the of fuel pad. This state is the ship being stationary and will start refilling a players fuel back up.

4.2 Bullets

A bullet is created by a player. The player is set as its owner and it will copy the its position and direction vector from the player and travel in that direction with a constant speed until it collides with something. When it does collide, it then deletes itself for optimization reasons. To ensure the players can only shoot one bullet per 30 frames, a cooldown is in place that is counted down each frame. Although a bit redundant, bullets do have a max distance they can travel before also deleting themselves in case they might escape the screen.

4.3 Objects

The objects are very simple in that they only consist of a color, position and dimension. With this they can be manually defined where every one would like to place one. The game only has two types of objects, the normal object that when a player touches they die, and fuel pads where players are able to land on.

4.4 Rendering

The rending for the game is in essence the idea of painting over new pixels on the screen each frame. So at the start of the frame the screen is fully colored in with the background color and then the `LocalGame` object as various groups that contain all different sprites and simply each group is called with the `draw()` method at the end after the `update()` methods have been run.

5 Discussion

Pygame has tremendously simplified the process of creating this game. It's Sprite, Group and Vector classes were tremendously helpful with their methods. The game is rather simplistic in design, but does meet all criteria of the assignment. The challenging part of this project was starting from nothing and needing to set up multiple things at the same time designed to work with each other smoothly. So time was spend figuring out how different functionality of the game would be designed to interact internally. Further, working in a team has enhanced our ability to maintain the code structure to be readable and flexible to modify. By dividing task and responsibility, we have decreased our work load and increased productivity. Later on in the project we were surprised by how well and efficiently it was to implement new changes as a result of our code base being well structured and organized. This collaborative not only improved the quality of the code, but also deepened each others understanding by sharing our.

5.1 Evaluation

Figure 2 showcases the result from running cProfile during the runtime of our game, specifically focusing on the update function. By observing the diagram, we can see that the collision detection method was significantly high. This isn't surprising and is most likely due to iteration of needing each bullet and player to check collision with all sprites in the game, giving a time complexity of $O(n)$. Theoretically, to improve the performance of the game we could implement a more complex system by having a grid like data structure, where it would have to check less sprites in total. But also given the rather small size of amount total sprites in the game, simply storing all sprites in a single big group and using pygames `spritecollide()` function works adequately and keeps a consistent frame rate either way, making any optimizations redundant with our current scale.

The input is also rather high too, but again, as expected since each frame the players `input()` method is called with a bool list of all keyboard keys pressed used to find out what movement to execute for the player and updating vectors accordingly, hence the `updateRotation()` method is called so many times.

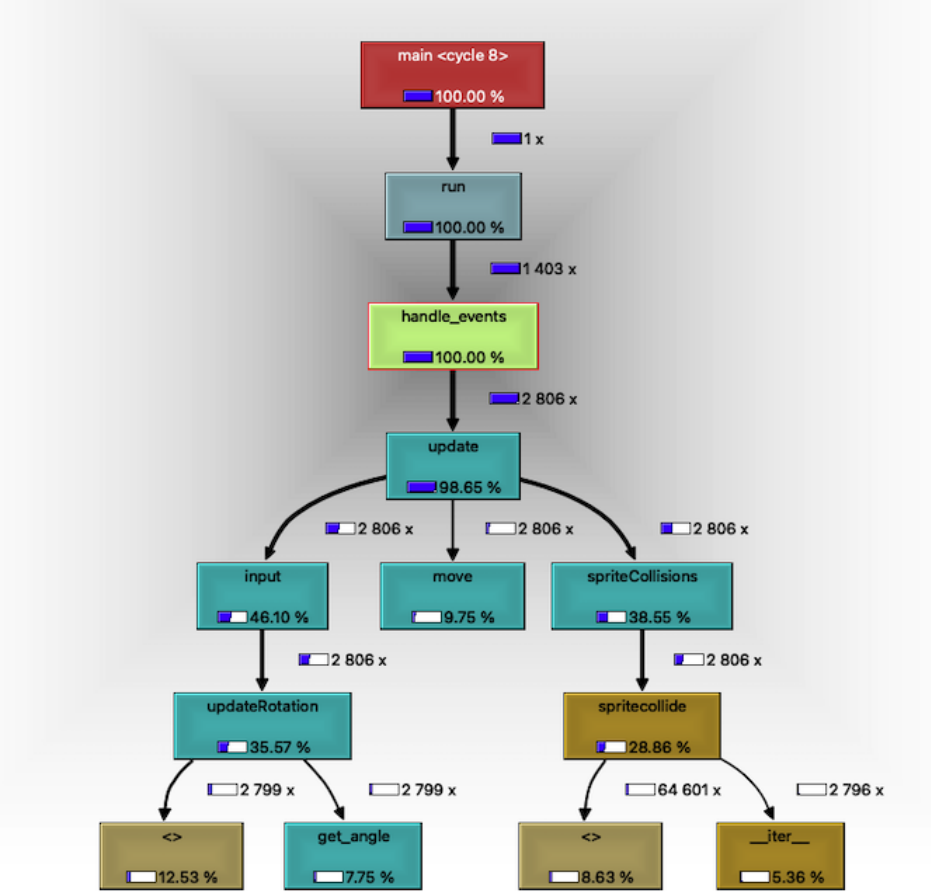

Figure 2: cProfile

6 Conclusion

The assignment was to implement a mayhem clone with specific features as requirements. We have completed the assignment successfully and satisfy all the specified requirements. The game performs rather well and from our tests runs perfectly smooth with a stable frame rate. It has been a rather good experience for both of us by cooperating in this assignment and has gained us some first hand experience on what it's like to develop a project in a team. Our communication was good, we constantly discussed ideas and suggestions. Originally, we had much greater ambitions of what we wanted to make, but had to cut back due to it not being as simple and fast as expected. The work load for both of us was balanced and each did their part really well.

References

[1] Pygame v2.6.0. (n.d.). *Pygame docs*. Pygame, `https://www.pygame.org/docs/`

[2] Wikipedia. (Feb 2024). *Object-oriented programming*. Wikimedia Foundation, `https://en.wikipedia.org//wiki/Object-oriented_programming`