

Assignment 3 report - INF-1400

Amund H. Strøm

April 7, 2022

Introduction

In this assignment we were tasked to implement a version of the classic Mayhem game in python. Mayhem is a 2-player game where you control a spaceship and try to shoot down the other players spaceship. To implement this there was a focus on using the “sprite module” which is a module that belong to the pygame library.

Technical Background

Sprite is a module in the pygame library that makes it easier to handle objects in a game. This module contains different classes and methods to handle tasks like, updating objects, drawing objects, collision between objects, and storing objects inside a group so you can easily add or remove them.

Design pattern is a way for the software developer to solve commonly occurring problems in a code. A pattern cannot be directly converted to code but is rather a template of how to solve a problem. There are many different patterns, and they are a highly flexible, so a pattern is chosen by the requirements of the code.

Polymorphism in programing means that a function/method with the same name can have many different forms. Meaning, that a function with the same name can have different appearances and give different results.

Design

The only design pattern I found in my program was object-oriented, and this was my intent. Object oriented design is the same concept as OOP. The program consists of a total of 12 classes, where there are 2 parent-, 7 child-, and 3 independent classes

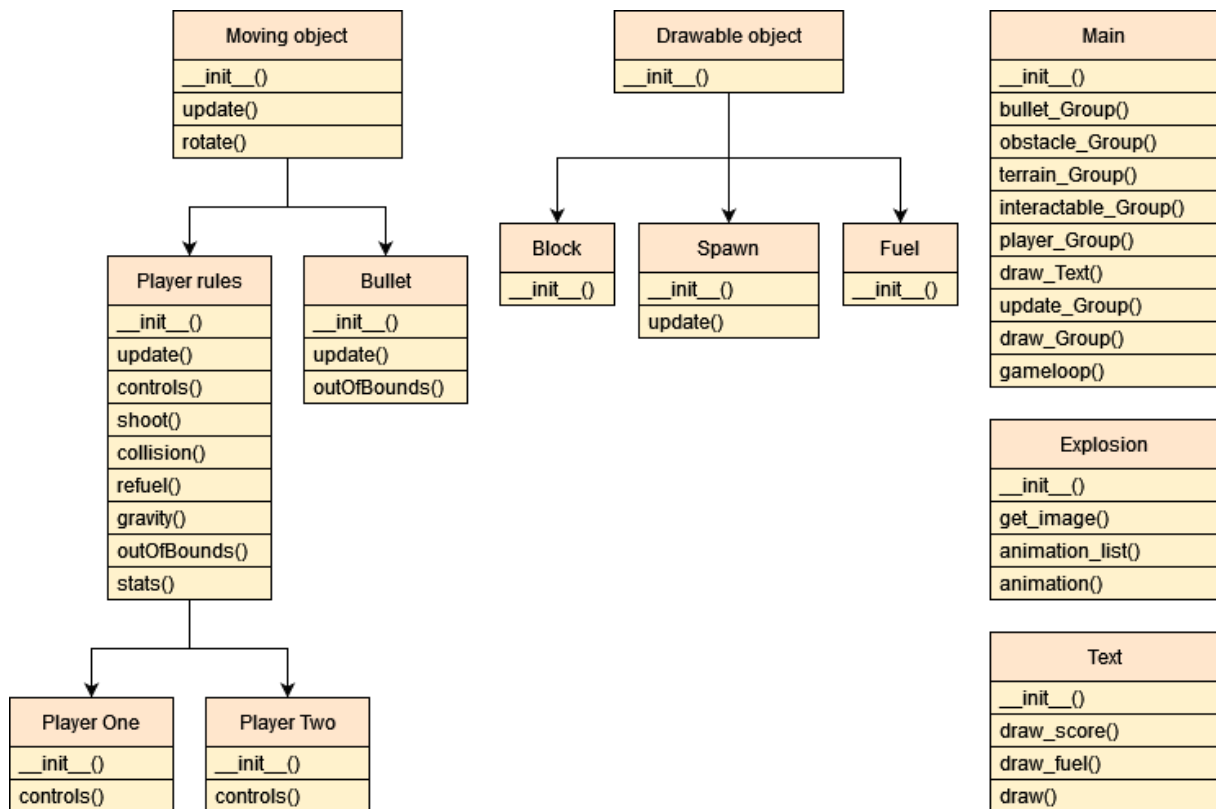


Figure 1: Illustration of class implementation

Implementation

Every class is unique so that there is no repeatable code. This makes the code somewhat difficult to read, but the name of the class describes exactly what it does. The largest and most convoluted class is the player rules class, since the player have many options of what to do inside the game. But again, the methods describe exactly what they are used for. The behavior from the different methods will look something like this:

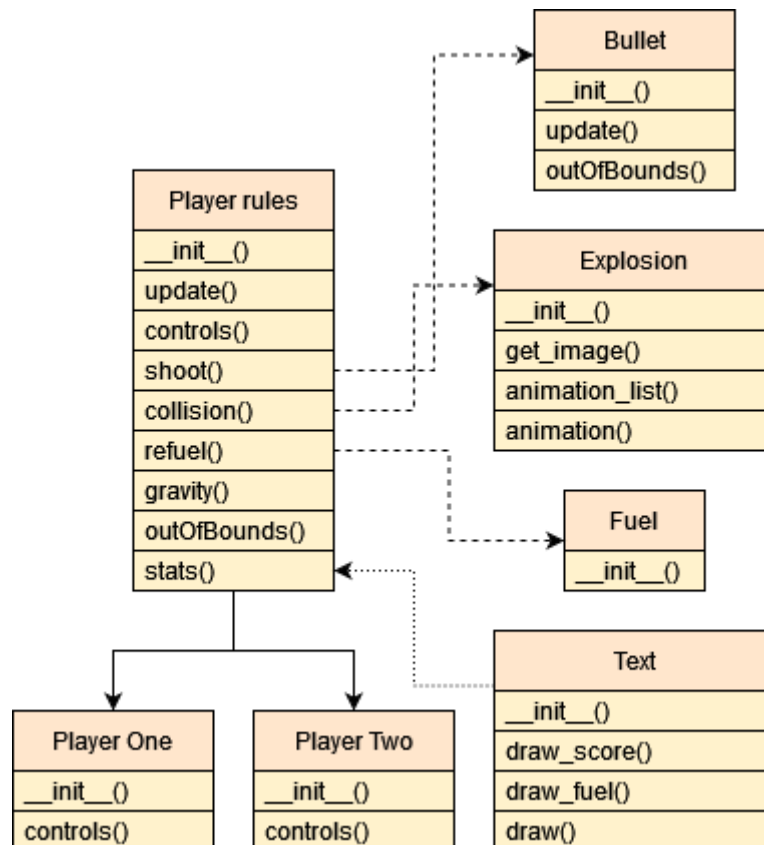


Figure 2: Illustration of player rules behavior

Player rules behavior

- Update, this method is used for updating the class every iteration of the game loop. The method is a built-in tool in the sprite module in pygame, that is easily executed inside the game loop. It most importantly moves the player for every iteration, and initializes other methods that needs to be updated, such as control, collision, refuel, outOfBounds and gravity.
- Controls, this method checks if a button on the keyboard have been pressed, it then checks if that button is 1 out of 4 different buttons, that is unique for each player. These 4 different buttons will either rotate the player to the right or left, move the

player forward or shoot a bullet. If the user shoots a bullet, the shoot method will be called.

- Shoot, this method is only called upon when a player presses the shoot button. The method creates a bullet object using the bullet class and adds it to the bullet sprite group. This makes it so each bullet will be a unique and independent object and therefor we can check the hitbox for each bullet.
- Collision, this method checks for every type of collision in the game. This consist of collision between a bullet and obstacle, player and interactable, player and obstacle, player and player, and player and bullet. If there has been a collision between a player and something else, the collision-method will call upon the explosion class, and there will be an animation of an explosion on the position of the crash.
- Refuel, this method is only used for counting down the timer for respawning the refuel interactable at the same position it was previously on.
- Gravity, this method moves the layer towards the center of the screen with a constant pull.
- outOfBounds, this method moves the player to the other side, if the player travels outside of the screen.
- Stats, this method returns a tuple with the score and fuel variables. It is used for printing the variables on the screen via the Text class. Meaning that the Text class calls upon this method to get the variables.

The Player- One and Two classes inherit form the Player rules class, so they will both have the same attributes and behavior. The only thing that sets them apart is different controls and spawn position.

The Explosion class used in the collision of the player my not have the most obvious implementation. It uses 3 methods to create an animation of an explosion. The explosion is one single image-sheet, meaning that there are 12 different images of an explosion on a single long image. So, we take the image sheet and divide it by 12, we then add those to a list, and then iterate through the list to get an animation.

- Get image, this method returns the correct frame form the image-sheet
- Animation list, this method adds each frame from the image-sheet to a list, using the Get image method, and return a list of images.
- Animation, this method iterates through the list and draws the image on screen.

Evaluation

This implementation of the classic Mayhem game fulfils all the requirements and technical backgrounds:

Requirements

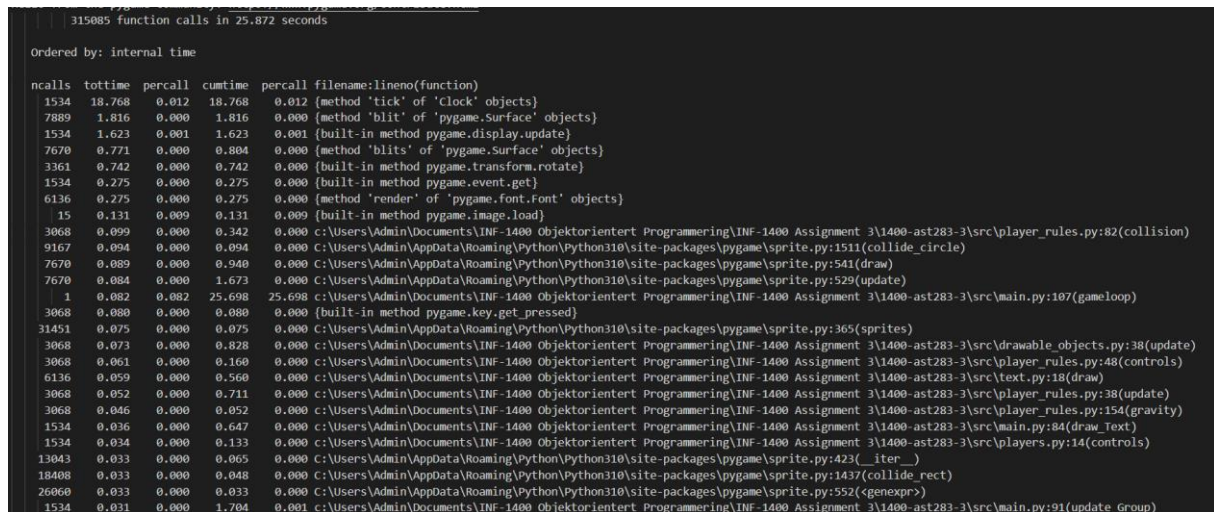
- There are two spaceships with four controls: rotate left and right, thrust and fire.
- Minimum one obstacle in the game world.
- Spaceship can crash with walls, obstacles, and other spaceships.
- Gravity acts on the spaceships.
- Each player has its own score, increasing if a player shoots down another spaceship and decreasing if a player crash.
- Each player has a limited amount of fuel, and it can be refueled by collecting the refuel object in the game world.

Technical background

- Implementation has more than two files and a config file containing global configuration constants.
- The main loop uses timing so that the game is playable on different computers
- The game starts by using the Python's `if __name__ == '__main__':` idiom, and the game starts with a single line.
- All visible objects use the pygame sprite module (except the Explosion class).
- All modules, classes and methods have docstrings.
- Looked for design patterns

Discussion

Using cProfiler and sorting by internal time (total time spent in the given function), we get this result. In my opinion it would be a more correct to sort by “per call” (quotient of total time and number of calls) but from my understanding that was not possible. But if we try and understand this result, we see that the methods that spent the most time were from the pygame library. To improve performance, I could maybe use less of these methods, but I have tried my best to limit my code. If we ignore the pygame methods and look only for the methods that I have created, we see that it spent most of its time in “player rules – collision”, which is as expected, since there are a lot of objects to check for collision. Other than that, I don’t know what else to look for and improve.



315085 function calls in 25.872 seconds

Ordered by: internal time

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
1534	18.768	0.012	18.768	0.012	{method 'tick' of 'Clock' objects}
7889	1.816	0.000	1.816	0.000	{method 'blit' of 'pygame.Surface' objects}
1534	1.623	0.001	1.623	0.001	{built-in method pygame.display.update}
7670	0.771	0.000	0.804	0.000	{method 'blits' of 'pygame.Surface' objects}
3361	0.742	0.000	0.742	0.000	{built-in method pygame.transform.rotate}
1534	0.275	0.000	0.275	0.000	{built-in method pygame.event.get}
6136	0.275	0.000	0.275	0.000	{method 'render' of 'pygame.font.Font' objects}
15	0.131	0.009	0.131	0.009	{built-in method pygame.image.load}
3068	0.099	0.000	0.342	0.000	c:\Users\Admin\Documents\INF-1400 Objektorientert Programming\INF-1400 Assignment 3\1400-ast283-3\src\player_rules.py:82(collision)
9167	0.094	0.000	0.094	0.000	C:\Users\Admin\AppData\Roaming\Python\Python310\site-packages\pygame\sprite.py:1511(collide_circle)
7670	0.089	0.000	0.940	0.000	C:\Users\Admin\AppData\Roaming\Python\Python310\site-packages\pygame\sprite.py:541(draw)
7670	0.084	0.000	1.673	0.000	C:\Users\Admin\AppData\Roaming\Python\Python310\site-packages\pygame\sprite.py:529(update)
1	0.082	0.082	25.698	25.698	c:\Users\Admin\Documents\INF-1400 Objektorientert Programming\INF-1400 Assignment 3\1400-ast283-3\src\main.py:107(gameloop)
3068	0.080	0.000	0.080	0.000	{built-in method pygame.key.get_pressed}
31451	0.075	0.000	0.075	0.000	C:\Users\Admin\AppData\Roaming\Python\Python310\site-packages\pygame\sprite.py:365(sprites)
3068	0.073	0.000	0.828	0.000	c:\Users\Admin\Documents\INF-1400 Objektorientert Programming\INF-1400 Assignment 3\1400-ast283-3\src\drawable_objects.py:38(update)
3068	0.061	0.000	0.160	0.000	c:\Users\Admin\Documents\INF-1400 Objektorientert Programming\INF-1400 Assignment 3\1400-ast283-3\src\player_rules.py:48(controls)
6136	0.059	0.000	0.560	0.000	c:\Users\Admin\Documents\INF-1400 Objektorientert Programming\INF-1400 Assignment 3\1400-ast283-3\src\text.py:18(draw)
3068	0.052	0.000	0.711	0.000	c:\Users\Admin\Documents\INF-1400 Objektorientert Programming\INF-1400 Assignment 3\1400-ast283-3\src\player_rules.py:38(update)
3068	0.046	0.000	0.052	0.000	c:\Users\Admin\Documents\INF-1400 Objektorientert Programming\INF-1400 Assignment 3\1400-ast283-3\src\player_rules.py:154(gravity)
1534	0.036	0.000	0.647	0.000	c:\Users\Admin\Documents\INF-1400 Objektorientert Programming\INF-1400 Assignment 3\1400-ast283-3\src\main.py:84(draw_Text)
1534	0.034	0.000	0.133	0.000	c:\Users\Admin\Documents\INF-1400 Objektorientert Programming\INF-1400 Assignment 3\1400-ast283-3\src\players.py:14(controls)
13043	0.033	0.000	0.065	0.000	C:\Users\Admin\AppData\Roaming\Python\Python310\site-packages\pygame\sprite.py:423(_iter_)
18408	0.033	0.000	0.048	0.000	C:\Users\Admin\AppData\Roaming\Python\Python310\site-packages\pygame\sprite.py:1437(collide_rect)
26060	0.033	0.000	0.033	0.000	C:\Users\Admin\AppData\Roaming\Python\Python310\site-packages\pygame\sprite.py:552(<genexpr>)
1534	0.031	0.000	1.704	0.001	c:\Users\Admin\Documents\INF-1400 Objektorientert Programming\INF-1400 Assignment 3\1400-ast283-3\src\main.py:91(update_group)

One other improvement could maybe be to use the sprite module in the Explosion class and call upon it in a different way when a player crashes. I am not particularly happy of how this class works and how it animates the explosion, but I could not figure out how to do it in another way.

Conclusion

My implementation is based on object-oriented design and consists of 12 classes. There is not much repeatable code, and the classes have a unique propose and behavior. The implementation also fulfils all the requirements, with some improvement in the Explosion class.

References

Code References

Bleed. (2014, November 11). *OpenGameArt.org*. Hentet fra Simple Explosion - Bleed's Game Art:
<https://opengameart.org/content/simple-explosion-bleeds-game-art>

darkrose. (2012, December 29). *OpenGameArt.org*. Hentet fra Vortex Background:
<https://opengameart.org/content/vortex-background>

FunwithPixels. (2017, November 20). *OpenGameArt.org*. Hentet fra Generic Planet Green Gas Giant:
<https://opengameart.org/content/generic-planet-green-gas-giant>

Pygame. (2022, Mars 10). *Pygame*. Hentet fra Pygame: <https://www.pygame.org/>

qubodup. (2009, August 24). *OpenGameArt.org*. Hentet fra Space Blocks:
<https://opengameart.org/content/space-blocks>

Russ, C. w. (2021, April 30). *Youtube.com*. Hentet fra PyGame Beginner Tutorial in Python - Loading Spritesheets: https://www.youtube.com/watch?v=M6e3_8LHc7A

Russ, C. w. (2021, May 13). *Youtube.com*. Hentet fra PyGame Beginner Tutorial in Python - Sprite Animation: <https://www.youtube.com/watch?v=nXOVcOBqFwM>

scenna. (2013, April 15). *OpenGameArt.org*. Hentet fra Circle:
<https://opengameart.org/content/circle>

Skorpio. (2013, August 9). *OpenGameArt.org*. Hentet fra Spaceship:
<https://opengameart.org/content/spaceship-8>