INF-1400

Assignment 3

Magnus Dahl-Hansen

March 2020

1 Introduction

In this report I describe my implementation of the two-player game "*Mayhem",* which is written in *Python* using the *Pygame* library. In the game each player controls a spaceship with a goal of shooting down the other player. The players have four available controls: rotate left, rotate right, thrust forwards and fire bullets, thus the world has gravity and an obstacle that the players need to avoid.

1.1 Requirements

The game and the code implementation were required to contain these features:

-Two players with four controls each. The controls had to rotate the players left or right, move them forward in the direction they're facing and fire a projectile.
-The game world had to contain gravity and minimum one obstacle
-A score system displayed on the screen, which increased or decreased depending on if the players collide or gets shot by another player.
-The spaceships had to be powered by fuel which could be refilled when they land on a fuel platform.

In addition to the requirements for the game world, the code implementation had to satisfy these requirements:

-The implementation had to contain at least two files, with one containing global configuration constants for the game world and the objects inside it.
-The main loop had to contain timing to make it playable on different computers
-The program was to be started using Pythons "if __name__==" __main__" "idiom
-All visible object had to be sprites which should be contained, updated and drawn via sprite groups.
-All modules, classes and methods were to be commented on using docstrings.

2 Technical Background

When starting on this assignment one should be familiar with classes, inheritance and sprites:

Classes are blueprints/scripts which contains a collection of variables that gives a form and a behaviour to an object.
Inheritance: is a tool that enables a programmer to create a class that takes all the functionality from another existing class, and therefore makes it possible for a class to inherit attributes and methods from another class.
Sprites are 2D computer graphic objects that can be used via the Pygame library. Using sprites makes it easier to work with objects in python since they have, for example, built in collision detection, draw and update functions.

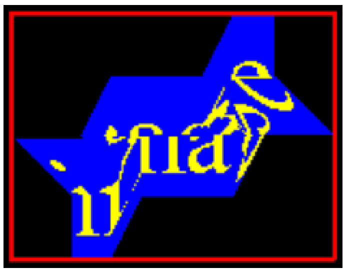
3 Design/implementation

To create a good overview of the implementation decided to have six python files, some containing classes defining the objects in the game, one containing the main program-loop and one for the global configuration values. In the main file you can find the program-loop and all the instances of the classes which are put into sprite groups, updated and drawn. The program is run via the Python idiom: "if __name__ == "__main__". Included in the main file are all the collision detection methods which uses the "sprite collide" function from Pygame. I decided to create the collision methods in the main file because I had easy access to all the objects in the game in this file.

The players are defined in two separate files, mainly to maintain a good overview over which keys that controls each player and which player an instance of a bullet belongs to. Player one is controlled with the "left" and "right" key for rotation, "up" key for moving forward and "m" key for shooting projectiles. Player two is controlled with the "a", "d" for rotation, "w" key for moving forward and the "c" key for shooting projectiles. When an instance of the bullet class is created as a projectile it is placed inside a sprite group belonging to one of the players.

When I implemented these classes I met two problems. First, I used pygames image rotate function to rotate the sprites image, but the image got distorted and smudged. I figured this happened because the rotate function generates a small inaccuracy and lowers the resolution of an image each time it is rotated (according to a source from stackoverflow.com). When the function rotates an already rotated image the sum of many rotation ends up distorting the image, as shown below.

Original image: Image after using pygame's rotate function:



This problem is solved by keeping a variable for the original image and only rotating a copy of this image once. When an image is rotated once, a new cop of the original image gets rotated with a bigger angle only allowing the function to generate one small inaccuracy instead of many.

The player had to move forward in the direction it was facing, which brings me to the next problem. I found a formula for calculating a vector from an angle value retrieved from https://python-forum.io/Thread-pygame-Moving-an-object-at-angles. The vector is calculated by converting an angle value into a radian value, which is then converted to a sine and a cosine value that makes up a vector pointing in the direction the player is facing. The angle value increases or decreases when the player presses a key for rotation. When the vector is calculated the players x and y values are then moved in the direction of this vector when a thrust key is pressed.

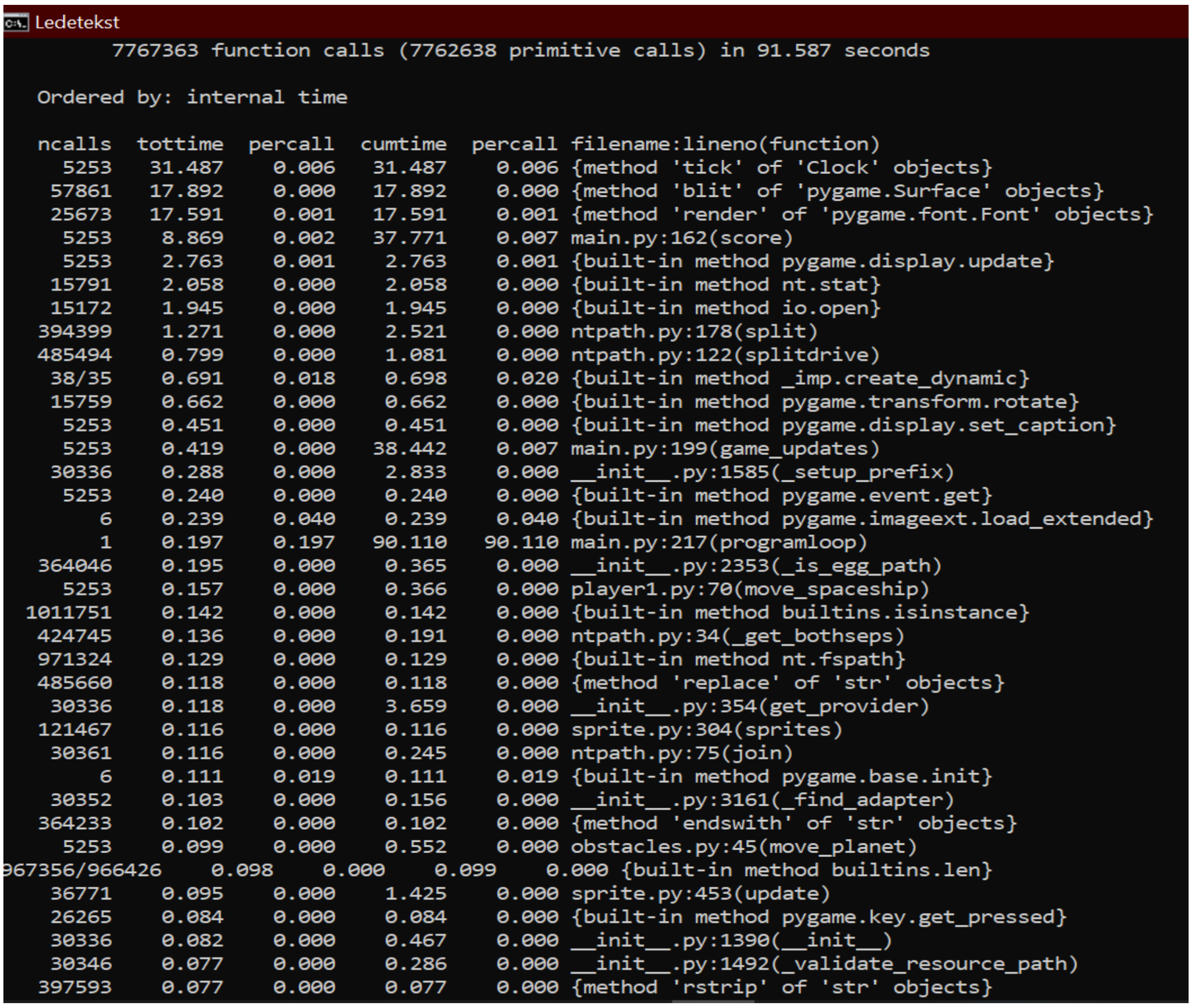
The bullet class is implemented in a fairly simple way. It is a sprite with methods for moving and updating it. An instance of the bullet is created inside each of the player classes when either the "m" or "c" button is pressed and then put inside a sprite group belonging to either player one or player two depending on who created the bullet. The moment one of these keys are pressed the bullet inherits the direction vector from its respective class and its x and y coordinates are moved in this direction until it is removed from its sprite group when it hits an object or collides with the walls.

I decided to implement one obstacle in the game. This is a sprite that moves in a specific direction with rotation, though I decided to set the rotation speed to zero since the bounding box gets bigger when the image rotates, and the players will then hit an invisible bounding box. The obstacle is moved by changing its x and y coordinated and is rotated using pygames rotate image function. In the same file there is also a class for the fuel icon. This sprite will refuel the players when they collide with it using the “sprite collide” function. When an instance of this class is created it requires an x and y position value so that it can be placed in different locations.

4 Discussion

As I mentioned in the design/implementation part I decided to put all the collision detection methods inside the main file to make it easier to access the sprite groups and objects. I would probably not do this the next time because the code looks messy and it would make more sense to connect each collision detection to a class it belongs to. Another minor flaw when I’m talking about collisions is the invisible hitbox surrounding the obstacle. The players will collide with an invisible box surrounding the obstacle, and when obstacle rotates this hit box will become larger. Because if this I set its rotation value to zero in the configuration file to make collisions with it more predictable. I think Pygame has a function for collisions with circles, though I did not get into this solution since it was not a requirement.

A minor flaw in the implementation is that some of the object’s positions are adjusted manually to make the game look better, for example placing the players next to the fuel icons. This could be a problem if one were to change the size of the game-screen because the sprites could appear in weird locations either on or outside the screen.
Performance:
Når jeg brukte cProfiler for å teste performancen til programmet fikk jeg følgende resultat:



```
Ledetekst
         7767363 function calls (7762638 primitive calls) in 91.587 seconds

   Ordered by: internal time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
     5253   31.487    0.006   31.487    0.006 {method 'tick' of 'Clock' objects}
    57861   17.892    0.000   17.892    0.000 {method 'blit' of 'pygame.Surface' objects}
    25673   17.591    0.001   17.591    0.001 {method 'render' of 'pygame.font.Font' objects}
     5253    8.869    0.002   37.771    0.007 main.py:162(score)
     5253    2.763    0.001    2.763    0.001 {built-in method pygame.display.update}
    15791    2.058    0.000    2.058    0.000 {built-in method nt.stat}
    15172    1.945    0.000    1.945    0.000 {built-in method io.open}
   394399    1.271    0.000    2.521    0.000 ntpath.py:178(split)
   485494    0.799    0.000    1.081    0.000 ntpath.py:122(splitdrive)
    38/35    0.691    0.018    0.698    0.020 {built-in method _imp.create_dynamic}
    15759    0.662    0.000    0.662    0.000 {built-in method pygame.transform.rotate}
     5253    0.451    0.000    0.451    0.000 {built-in method pygame.display.set_caption}
     5253    0.419    0.000   38.442    0.007 main.py:199(game_updates)
    30336    0.288    0.000    2.833    0.000 __init__.py:1585(_setup_prefix)
     5253    0.240    0.000    0.240    0.000 {built-in method pygame.event.get}
        6    0.239    0.040    0.239    0.040 {built-in method pygame.imageext.load_extended}
        1    0.197    0.197   90.110   90.110 main.py:217(programloop)
   364046    0.195    0.000    0.365    0.000 __init__.py:2353(_is_egg_path)
     5253    0.157    0.000    0.366    0.000 player1.py:70(move_spaceship)
  1011751    0.142    0.000    0.142    0.000 {built-in method builtins.isinstance}
   424745    0.136    0.000    0.191    0.000 ntpath.py:34(_get_bothseps)
   971324    0.129    0.000    0.129    0.000 {built-in method nt.fspath}
   485660    0.118    0.000    0.118    0.000 {method 'replace' of 'str' objects}
    30336    0.118    0.000    3.659    0.000 __init__.py:354(get_provider)
   121467    0.116    0.000    0.116    0.000 sprite.py:304(sprites)
    30361    0.116    0.000    0.245    0.000 ntpath.py:75(join)
        6    0.111    0.019    0.111    0.019 {built-in method pygame.base.init}
    30352    0.103    0.000    0.156    0.000 __init__.py:3161(_find_adapter)
   364233    0.102    0.000    0.102    0.000 {method 'endswith' of 'str' objects}
     5253    0.099    0.000    0.552    0.000 obstacles.py:45(move_planet)
967356/966426    0.098    0.000    0.099    0.000 {built-in method builtins.len}
    36771    0.095    0.000    1.425    0.000 sprite.py:453(update)
    26265    0.084    0.000    0.084    0.000 {built-in method pygame.key.get_pressed}
    30336    0.082    0.000    0.467    0.000 __init__.py:1390(__init__)
    30346    0.077    0.000    0.286    0.000 __init__.py:1492(_validate_resource_path)
   397593    0.077    0.000    0.077    0.000 {method 'rstrip' of 'str' objects}
```

The results from the cProfile screenshot tells me that the tick, blit, render, game update and main program loop methods uses the most time. When it comes to the tick function which keeps track of time I don’t think there are many ways to optimize it since it simply has to be called many times to keep track of time.

The blit and render functions are used for computer graphics and are generally slow. I’m not certain why this is, though I think that graphics handling is more demanding for the processor. “game updates” and “programloop” uses more time than other methods since they are responsible for updating, processing all methods and running the entire program.

5 Conclusion

This assignments task was to recreate a game called Mayhem supporting some required functions. The assignment was solved using the python programming language, Pygame library and with good use of classes. I think my code could have had a better overview so for the next assignment I will try to design my code in a better way and make better use of inheritance.

6 References

- https://stackoverflow.com/questions/4183208/how-do-i-rotate-an-image-around-its-center-using-pygame
- https://python-forum.io/Thread-pygame-Moving-an-object-at-angles

- https://stackoverflow.com/questions/18839039/how-to-wait-some-time-in-pygame