INF-1400: Object-oriented programming
Assignment 3 - Mayhem Clone

Trym Varland

Git user: tva050

UiT - Norges Arktiske Universitet

April 12, 2023

1 Introduction

In the third assignment in INF-1400 course, we gone make a clone of a classic two-player Amiga game called Mayhem. The goal of the game is to shoot down the opponent, and the movement of the spaceships is done by having four controls move right, move left, thrust and fire. In the game world you have some starting pads for the spaceships, obstacles which absorbs bullets/laser-beams and in additional the world has some gravity working on the spaceships. This gives the game, some requirements:

- Two spaceships, each with four controllers.
- Number of obstacles bigger than zero.
- Spaceship can collide with obstacles, walls and other spaceships.
- Each player, have a score displayed on the screen. If the player crash or gets hit by bullet the score decrease, and if the player shoots down the opponent the score increase.
- Each of the players have limited amount of fuel, which decrease every time the player use the thrust. The player refuel by landing on the start pad.

2 Technical Background

2.1 Sprites

Sprites is easy as a tow-dimensional image, which can be manipulated in to large images. That would say that we use sprites for visualizations of characters or objects in a game, these sprites can be moved around, animated to visualize movement or the sprites can be stacked over each other to make more complex images. Forward the sprites can be given properties and behaviors, this can be just a position on the screen, it can move, interact with other sprites, shoot something or have a health. In figure 1, it is shown the sprites for player 1 and player 2 in the Mayhem game.
In python we can install what is called `pygame` which consist of python modules [1], one of these modules is called `pygame.sprite`. Which handles our wanted sprites, it also include a lot of other futures to help us. One of these

futures is the class `pygame.sprite.Group`, that manage and contains multiple sprites [2]. Another future, is the class `pygame.sprite.collide` where you can give different collide functions by adding i.e. `pygame.sprite.spritecollide`, `pygame.sprite.collide_mask` or `pygame.sprite.collide_rect` (we gone come back to what rect is in 2.2) which depends on what we are after. This collide functions in sprites module, checks if the sprites is collide which help us to give assignment to the sprites if they collide/intersect [2].

Figure 1: The sprites used for player 1 and player 2, in the Mayhem game.

2.2 rect

Another great thing in pygame, is the class called `pygame.Rect`. This class represent a rectangle area, where pygame use the rect of the object to store and manipulate the rectangle area [2]. Which is helpful, to detect collisions or for the object position. A `pygame.Rect` object, can be created through specifying values for left, top, width and height. This can also be created if a already existing object that is a rect or have a attribute called rect, which is mostly used in the Mayhem game. Figure 2 shows how the rect will be drawn around one of the spacecrafts.

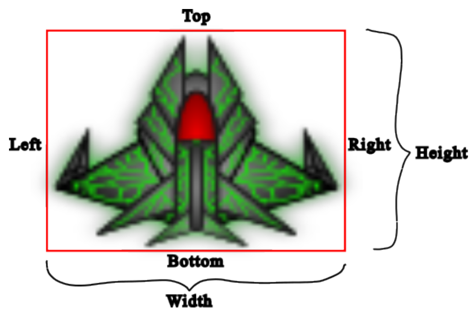

Figure 2: Illustration of how the rect would look like drawn around a spaceship.

2.3 Docstring

Docstrings is a literal string which appears as the first statement in a module, class, function or method definition. The docstring provides us with documentation about the object it is associated with, purpose, usage and behavior. In the code snippet blow, you can see how to use the docstrings in python. In python we can access the docstrings by using `__doc__` attribute of the object it is associated with, that would say if we wanted to access `my_method` inside the class. We would use the `print` method (`print(Myclass.my_method.__doc__)`) or `help` (`help(mymodule.Myclass.my_method)`).

```python
"""
Where the module docstring is. Which could include, a descriptions
    of what you find in the module.
"""
class Myclass:
    """
    Class docstring.
    Includes a description of the class.

    ...
    Attributes
    ----------

    Methods
    ----------
    """
    def my_method(self):
        """
        Method docstring.
        Includes a description of the method.

        Parameters
        ----------
        """
def my_funtion():
    """
    Function docstring.
    Includes a description of the function.

    Paramters
    ----------
    """
```

Listing 1: Example of docstrings

3 Design & Implementation

3.1 Spaceships

The spaceships class, contains a lot of the properties each of the players spaceships have. The class inherit from the `pygame.sprite.Sprite` class, which makes it easier for use to work with sprites. The class takes in a image, position and velocity of the sprites. Which is done so that we can always have control over each single spaceship sprites and give them some initial position, velocity and a given image. In the class we find eight methods `move`, `thrust`, `move_right`, `move_left`, `boundaries`, `draw`, `update` and `reset`. These methods gives the abilities to the spaceships, for them to move around in the boundary of the screen. We also have assigned the class with a group for the laser beam each of the individual spaceships have, and a cool down period which is handled in the update method. If the `reset` method is called, the individual spaceship will respawn on the original location with max amount of fuel.

3.2 Laser beam

In the laser beam class we inherits from the `pygame.sprite.Sprite` class, we take in a image, position and angle. The position and angle comes from the spaceship, so that the laser beam will be shoot out from the right place on the screen and in the correct angle which will be the direction of the laser beam. The class contains three method `draw`, `move` and `update`. Here the draw method draws the laser beam image, move method moves the laser beam in the direction of the spaceship and the update method updates the move and draw methods so that it will follow the spaceships as they move.

3.3 Platforms

The platform class inherit from the `pygame.sprite.Sprite` class in the sprite module, and it takes in a position x and y. This class is mainly used for us to initialize, the sprite that would say draw the plat from sprite and place it on a location on the screen.

3.4 Obstacles

Obstacle class also inherits from the `pygame.sprite.Sprite`, and takes in a image and a position. The obstacle class have so and say, the exactly properties as the platform class. But here we place obstacles sprites around on the screen with a given picture and a given location on the screen.

3.5 Button

Button class takes in an image and position, so that we a able to choose different button images and their position on the the screen. The class contains two methods `draw` and `check_for_input`, draw method just draw the wanted button image on the screen. In the check for input method, we check if the mouse is over the button and if its pressed than it returns true else false.

3.6 Screens

In the screen class, is where the main menu and the info screen is created. The main menu method, shows the main menu which is the screen we first meet when starting the game. Here you see two button play and info, by using the button class we can check if any of the buttons is pressed. If the info button is pressed, we go to the info screen which is created by the info method. The info screen just shows, the keys to controls the spaceships and you can go back by pressing the back button which again use the button class. Now that we are back to the main menu, we press the play button and the game starts.

3.7 Game

The game class is the main class, which makes everything happen. Every sprites gets put in to their corresponding sprite group. The class contains fourteen methods in total, where 5 methods i called in the main game loop. The methods called in the game loop is `event_handler`, `shoot`, `collision`, `update` and `draw`. The `event_handler` method, gives the players the ability to move the spaceships by using the keys on the keyboard. For player one the control keys are 'w' to thrust, 'd' to move right and 'a' to move left. Player two use to up arrow for thrust, right arrow to move right and left arrow to move left. `Shoot` method handles the shoot keys for each of the spaceships, and also adds laser beams to the laser beam group to their representative spaceship. `Collision` method, handles all the collisions with the sprites in the game. That will say collisions between spaceships, platforms, obstacles and laser beams. `Update` method, is updating all the sprite groups and the laser beams that are shoot. The last method `draw`, draws all the sprites to the screen, it also draws the fuel bar, the score and the background of the game.

3.8 UML diagram

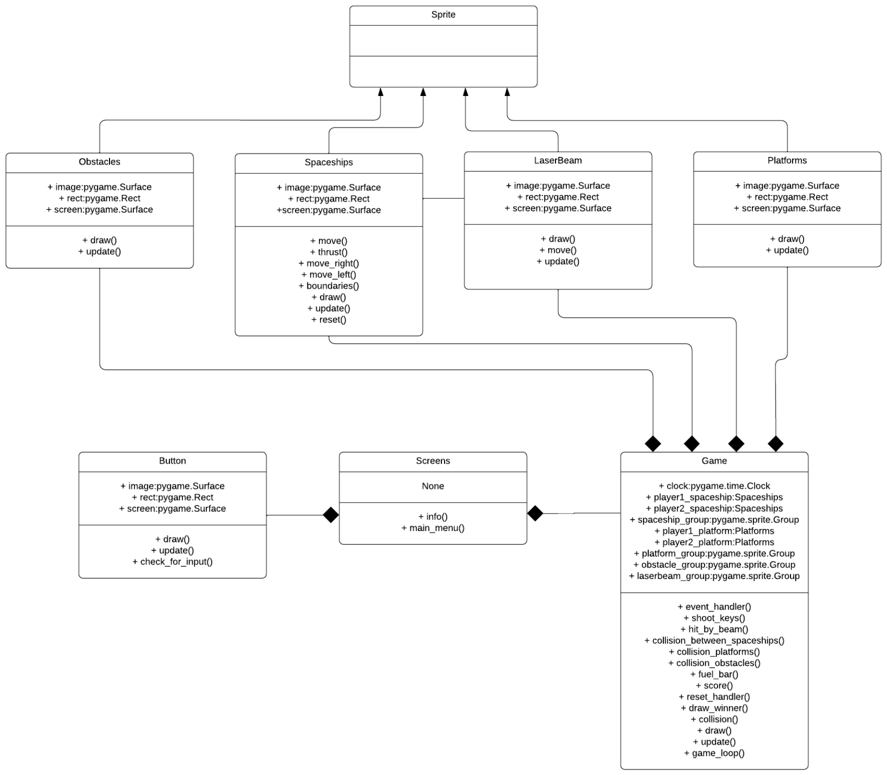

Figure 3: UML diagram for the Mayhem Clone game.

4 Experiments and Results

4.1 Code profile

Under we have the code profile for the Mayhem program, where all values in profile will vary with how long the program is running. Where builtins and limimport, is removed. In the code profile 'ncalls' is number of something called on, 'tottime' is the total time used by a function, 'percall' is the total time with number of calls and 'cumtime' is the time spent in given function. So we can see that we was in the main menu screen for 41.844 seconds, in the info screen we was in for 40.584s and the game was played for 38.294s. For us to improve the performance of the code, we will think of how to get cumtime down. By looking at the code and looking at the code profile we can see any clear improvements of the code, but if we had rewritten the code so that we would have own class for movements for all the sprites, we would maybe see some improvement in the code. No bottlenecks in the code is to be found, by looking at the code profile.

```
   ncalls  tottime  percall  cumtime  percall filename:lineno(
    function)
        1    0.000    0.000   41.844   41.844 <string>:1(<module>)
        2    0.000    0.000    0.000    0.000 __init__.py:89(
    find_spec)
     7463    0.007    0.000    0.009    0.000 button.py:34(__init__
    )
     7463    0.003    0.000    0.137    0.000 button.py:52(draw)
        5    0.000    0.000    0.000    0.000 button.py:63(
    check_for_input)
        1    0.000    0.000    0.001    0.001 freetype.py:1(<module
    >)
     2232    0.065    0.000    0.289    0.000 game.py:108(
    event_handler)
     4464    0.013    0.000    0.054    0.000 game.py:141(shootkeys
    )
     2232    0.016    0.000    0.070    0.000 game.py:162(shoot)
     2232    0.006    0.000    0.013    0.000 game.py:166(
    hit_by_beam)
     2232    0.004    0.000    0.009    0.000 game.py:179(
    collison_between_spaceships)
     2232    0.027    0.000    0.054    0.000 game.py:193(
    collision_platform)
     2232    0.027    0.000    0.159    0.000 game.py:213(
    collision_obstacles)
     2232    0.011    0.000    0.034    0.000 game.py:234(fuel_bar)
     2232    0.024    0.000    0.196    0.000 game.py:245(score)
       67    0.001    0.000    0.203    0.003 game.py:258(
    draw_winner)
       67    0.000    0.000    0.001    0.000 game.py:276(
    reset_handler)
     2232    0.011    0.000    0.246    0.000 game.py:294(collision
    )
     2232    0.011    0.000    0.606    0.000 game.py:308(update)
     2232    0.025    0.000    2.689    0.001 game.py:323(draw)
        1    0.063    0.063   38.294   38.294 game.py:344(game_loop
    )
        1    0.000    0.000    0.027    0.027 game.py:78(__init__)
       16    0.000    0.000    0.008    0.001 laserbeam.py:38(
    __init__)
     2500    0.002    0.000    0.005    0.000 laserbeam.py:65(draw)
     2500    0.008    0.000    0.011    0.000 laserbeam.py:76(move)
     2500    0.003    0.000    0.019    0.000 laserbeam.py:87(
    update)
        5    0.000    0.000    0.023    0.005 obstacles.py:32(
    __init__)
    11160    0.009    0.000    0.286    0.000 obstacles.py:53(draw)
    11160    0.007    0.000    0.293    0.000 obstacles.py:62(
    update)
        2    0.000    0.000    0.003    0.002 platforms.py:35(
    __init__)
     4464    0.004    0.000    0.105    0.000 platforms.py:55(draw)
     4464    0.004    0.000    0.109    0.000 platforms.py:69(
    update)
        1    0.008    0.008   40.584   40.584 screens.py:36(info)
      2/1    0.027    0.013   41.844   41.844 screens.py:72(
    main_manu)
      389    0.014    0.000    0.027    0.000 spaceships.py:109(
    move_right)
      362    0.011    0.000    0.022    0.000 spaceships.py:123(
    move_left)
     4464    0.008    0.000    0.008    0.000 spaceships.py:137(
    boundaries)
     4464    0.005    0.000    0.091    0.000 spaceships.py:155(
    draw)
     4464    0.012    0.000    0.137    0.000 spaceships.py:165(
    update)
        7    0.000    0.000    0.000    0.000 spaceships.py:182(
    reset)
        2    0.000    0.000    0.001    0.000 spaceships.py:48(
    __init__)
     4464    0.018    0.000    0.025    0.000 spaceships.py:83(move
    )
     1425    0.024    0.000    0.026    0.000 spaceships.py:95(
    thrust)
       25    0.000    0.000    0.000    0.000 sprite.py:113(
    __init__)
     8928    0.008    0.000    0.013    0.000 sprite.py:1437(
    collide_rect)
       25    0.000    0.000    0.000    0.000 sprite.py:154(
    add_internal)
       10    0.000    0.000    0.000    0.000 sprite.py:162(
    remove_internal)
    24552    0.045    0.000    0.061    0.000 sprite.py:1633(
    collide_mask)
    26784    0.032    0.000    0.045    0.000 sprite.py:1660(
    spritecollide)
     4464    0.025    0.000    0.070    0.000 sprite.py:1713(
    groupcollide)
        6    0.000    0.000    0.000    0.000 sprite.py:184(kill)
        5    0.000    0.000    0.000    0.000 sprite.py:361(
    __init__)
    58034    0.035    0.000    0.035    0.000 sprite.py:365(sprites
    )
       25    0.000    0.000    0.000    0.000 sprite.py:378(
    add_internal)
       16    0.000    0.000    0.000    0.000 sprite.py:391(
    remove_internal)
       25    0.000    0.000    0.000    0.000 sprite.py:402(
    has_internal)
     8928    0.011    0.000    0.027    0.000 sprite.py:423(
    __iter__)
       30    0.000    0.000    0.000    0.000 sprite.py:429(add)
    11160    0.032    0.000    0.596    0.000 sprite.py:529(update)
    11160    0.048    0.000    0.554    0.000 sprite.py:541(draw)
    33748    0.015    0.000    0.015    0.000 sprite.py:552(<
    genexpr>)
        2    0.000    0.000    0.000    0.000 sprite.py:587(empty)
        5    0.000    0.000    0.000    0.000 sprite.py:638(
    __init__)
        1    0.000    0.000    0.000    0.000 {method '__exit__' of
     '_io._IOBase' objects}
        4    0.000    0.000    0.000    0.000 {method '__exit__' of
      '_thread.lock' objects}
        21    0.000    0.000    0.000    0.000 {method 'append' of '
    list' objects}
    49645    3.186    0.000    3.186    0.000 {method 'blit' of '
    pygame.Surface' objects}
    11160    0.466    0.000    0.482    0.000 {method 'blits' of '
    pygame.Surface' objects}
         6    0.000    0.000    0.000    0.000 {method 'clear' of '
    dict' objects}
    23964    0.008    0.000    0.008    0.000 {method 'colliderect'
     of 'pygame.Rect' objects}
        25    0.000    0.000    0.000    0.000 {method '
    convert_alpha' of 'pygame.Surface' objects}
```

Listing 2: code profile given by Cprofile.

5 Conclusion

The goal of the task was to make a clone of the Mayhem game, where we had to meet some requirements. We was able to meet all of them, we created two spaceship each with four controllers, five obstacles on the screen, both the spaceships are able to collide with each other, obstacles and the walls. Where collisions with each other and obstacles resets the players spaceship, to the start platform. Also each of the players have displayed score on the screen, while the game is running. The players lose points if they crash or gets hit by a leaser beam, and if the player shoots down the opponent the score increase. The players spaceships have a limited amount of fuel, which decrease every time the thrust is used and to refuel they need to land on the starting platform.
We see that the program meets all the, but the code structure of program could be better by having a own moving class which handles all the movements in the game for example. But in overall we meet the requirements and the controls for the spaceships feels good.

6 Appendix

References

[1] About - pygame wiki. https://www.pygame.org/wiki/about, April 2023. [Online; accessed 10. Apr. 2023].

[2] pygame.sprite — pygame v2.4.0 documentation. https://www.pygame.org/docs/ref/sprite.html#pygame.sprite.Group, March 2023. [Online; accessed 10. Apr. 2023].