Assignment 3 report - INF-1400

Daniel Antonsen & Håkon Bakker

UiT - Norges Arktiske Universitet

7. april 2022

Introduction

In this assignment, we have made a clone of the classic Mayhem game written by Espen Skoglund. The Game is implemented using concepts from object-oriented programming such as classes, method and inheritance.

Backround

In this assignment, we have used Python 3.9.2Foundation [2021]. Python is a interpreted programming language, and it support object-oriented programming. In addition to this, we have also used Pygame 1.9.5set of members in pygame community [2019].

Classes in object-oriented programming are definitions or description of an object, methods are functions that tells the object how to operate and objects are instances of their classes.

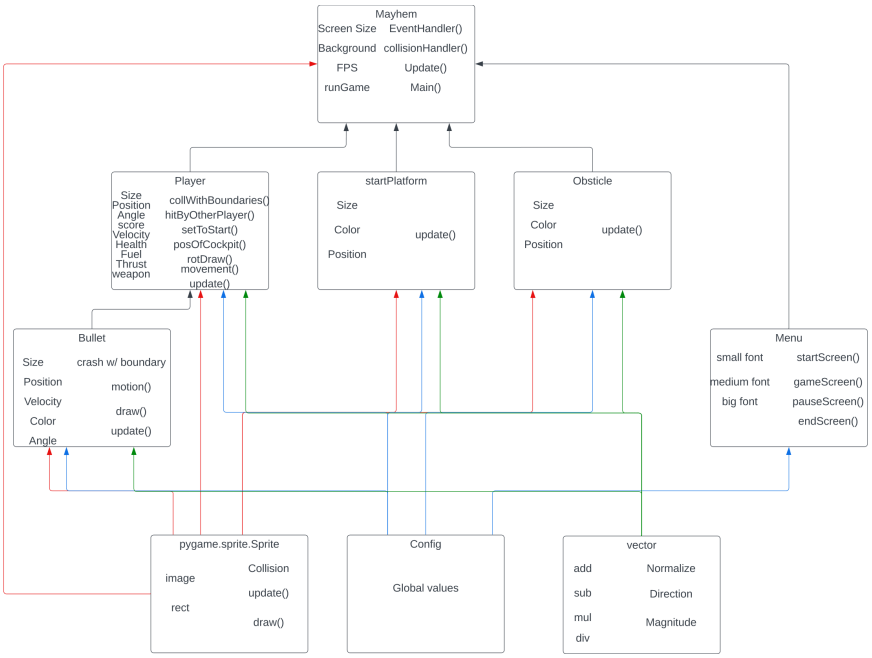
Inheritance is when a class object is derived from a parent class. The child class is a sub-class of the parent class and will have the same interface as their parent class. They will have the same methods and can be extended.

Sprite module contains classes, These are base classes for the objects. Sprites attributes can be to implement images on the screen, get interactions/collisions between different objects and more.

Design

The code consists of eight files where each file contains one class. the ninth module/class is the pygame sprite module used to implement certain attributes. Four classes represents the player class, start platform class and the obstacle class. These four classes are the child class that inherent from the sprite class witch is the parent class. One menu class that gives game information and option to play the game or exit. Config class is used to store global values used in all classes mentioned previously. The vector class is used for vector calculations in the player, start platform, obstacle and bullet class.

The Mayhem class defines the main file, where all the other classes are implemented. This class runs the game, in other words, all events, handlers and updates runs through this class. A UML diagram over the structure of the implementation for the simulation is given in figure 1



Figur 1: UML diagram

Implementations

In the vector module, a simple and basic vector class object has been implemented. It is used to represent the velocities as vector objects. The class contains methods to handle add, sub, mul and truediv, as well as method to calculate Magnitude, Direction and Normalize. It also contains a function to calculate the distance between two vectors, even though this is not used in this assignment. The Vector module is originally written as a module for the previous assignment containing boids.

Player Implementation

Represents the player class object. The image(s) of the player object is implemented by loading a image using pygame.image.load function. The image is then copied in the rotateDraw method before it is called in the update method to be able to be drawn on the screen.

The movement of the player is determined by the angle of the image and the acceleration from thrust and gravity. This velocity is calculated using the following formula:

$$\text{self.vel.x} += \cos(\text{self.angle}) * \text{self.thrust} * \text{time}$$
$$\text{self.vel.y} += -\sin(\text{self.angle}) * (\text{self.thrust} + \text{Config.GRAVITY}) * \text{time}$$

Pygame starts the coordinates (0,0) at the top left corner and positive y-directions downwards. Therefore there must be a negative sinus for the y-velocity. Position is then updated by adding the velocity to the position.

$$\text{self.rect.x} += \text{self.vel.x}$$
$$\text{self.rect.y} += \text{self.vel.y}$$

Rotation of the player is implemented by using the pygame.transform.rotate where the change of angle speed is determined by a global value called diffAngle from the config class. Using self.image.get_rect() function from pygame the image of the player is positioned at the center with the rotation oriented around its own center. center is found by taking the width and height and divide by two. Now to get the transformed player image to be drawn on screen, the image is blitted on screen using SCREEN.blit().

Player has a method to check if the player collides with boundaries of the screen. This is checked with a if statement. If the x,y is less than 0 or greater then width,height, the player life is reduced by one and the setToStart method is called to position the player back at the start platform.

hitByOthePlayer method checks for collision with other player using if statement. If player is hit, health is reduced by five. another if statement within the first one states that if health is less than zero, then life is reduced by one and setTwoStart method is called.

setToStart method is used to find the starting position and set velocity vector to (0,0), angle i sett to starting angle and the fuel and health is set to its max value.

posOfCockpit method is used to calculate the position of the cockpit in order to get the starting position for bullet class. Since the image of the player is rotating, the coordinates will transform. in order to calculate this, matrix transformation is needed. This matrix is made by implementing a 2 by 2 numpy array like this:

$$\begin{bmatrix} \cos(\text{self.angle}) & -\sin(\text{self.angle}) \\ \sin(\text{self.angle}) & \cos(\text{self.angle}) \end{bmatrix}$$

After this the center top of the player image is found by taking the width of the image and devide by two, and leaving the y-axis unchanged, giving the coordinates (width/2, 0). Multipling the matrix with the top center coordinates gives the rotational top center position of the image. By finding the true center of the image and adding the rotational top center we get the position of the cockpit for every angle.

Update method calls three methods from the player class, collWithBoundaries, rotateDraw and movement. these are the three methods needed in the update method that is needed when update is called in the Mayhem class.

Bullets Implementation

Represents the bullet class object, which is fired from the player object. The object is implemented by image in the form of a pygame.Surface() and drawing a circle using the pygame.draw.circle() on the surface. The circle is draw with center at the image's center and half the width and height as radius. The color is set to white. The pygame.sprite.Sprite module requires a rectangular object to represent the object, thus we use the self.image.get_rect() command to obtain a rectangular area to represent the bullet as a sprite object. The rect object is what represent the sprites position on the screen, by using self.rect.x and self.rect.y. The velocity of the bullet is implemented as a vector using the Vector module, and is set to vector(0, 0) initially. To limit the speed of the bullet we have definied a maxSpeed variable. And define the angle θ to determine the direction the bullet fired out of the player object.

The class has a method draw() which blit the image on the screen with self.SCREEN.blit(self.image, self.rect). It has the method crashWithBoundaries() which determines the action if the bullet hits the screen boundaries. It is defined checking if bullet collides with left or rigth part of the screen, if so then we use the self.kill() method which comes from the pygame.sprite.Sprite module. The method removes the object from sprite, but the method does not change the sprite and thus it can be continued to be used. This is also used for the top and bottom part of the screen.

The object has the method motion(time) which has the input time. It determines updated the motion of the bullet; that is update the position using the velocity. The velocity is defined by

$$\begin{aligned} self.vel.x+ &= self.maxSpeed * \cos(self.angle) * time \\ self.vel.y+ &= self.maxSpeed * (-\sin(self.angle)) * time \end{aligned}$$

Here the negative sign is used before the y-component since pygame uses positive y-axis downwards and the point (0, 0) is defined as the top left corner. We then uses the velocity to update the position by

$$\begin{aligned} self.rect.x+ &= int(self.vel.x) \\ self.rect.y+ &= int(self.vel.y) \end{aligned}$$

Where the int() function takes a float or other type of object and transforms it to an integer since pygame objects position is represented as integers on the screen.

The class also contains a method update() that calls all the methods that need to be called in the object. It adds to the update() method already contained in the pygame.sprite.Sprite module.

Obsticle/platform Implementation

Represents the obstacle/platform class object. The class is implemented by creating an image in the form of pygame.Surface() with a width and height, which in the case of the obsticle is the same thus we create a square obsticle object. The color of the obsticle/platform is set by using the self.image.fill() function which fill the surface with the input color. The pygame.sprite.Sprite module requires a rectangular object to represent the hitbox of the object, this is done by using the self.image.get_rect() command. The position of this rectangular area is what determines the position of the pygame surface on the backgorund screen.

The drawing of the image on the background is done with the update() method, which adds to update() method described in the pygame.sprite.Sprite object class.

Menu Implementation

Represents the different menus used in the course of the game. The class is implemented by creating three different fonts for the text on the screen; one big, one medium and small in size. The big is defined in 'Verdana' font with a size of 55 by using the pygame.font.SysFont() method, the medium is defined in 'Helvetica' with a size of 30 also using pygame.font.SysFont() method. And the small is defined in 'Helvetica' with a size of 20, this also using the pygame.font.SysFont() method. It is defined three variables

The class has the method startScreen() which represents the start screen when the game is booted up. It contains a while-loop so the start screen is continually run as long as the self.stScreen variable is True, which it is by default. The text for the name of the game and controls for each player, and also the actions for start and exit game, is blit in hte screen using self.SCREEN.blit() method. The text is first rendered using the font.render() method. The for-loop looks for events which occurs by using the pygame.event.get() method, if pygame.QUIT action is detected, then the game closes with pygame.quit() and the system shutsdown with sys.exit() method. If it register a buttondown action; if the button is SPACEBAR then the variable self.stScreen = False and the game starts. If the button is ESCAPE then the player exits the game with pygame.quit() and sys.exit(). In the loop the display is updated using the pygame.display.update() method.

The class has the method gameScreen() which represents the game screen when game is running. It uses some font.render() method to render text on the screen. The text for both players health, score, lives and fuel is blit on the screen using self.SCREEN.blit() method. As long as the gameplay is running the method is called.

The class has a method pauseScreen() which represent the pause screen when the gamplay is paused. It has a while-loop so as long as the self.pause varable is True, the loop is running. We set the same background for all menus. We render text for pause text, actions such as continue the game or exiting the game with the some font.render() method. A for-loop which looks for actions, if it register

pygame.QUIT then the player exits the game with pygame.quit() and sys.exit(). If the SPACEBAR is pressed then the variable self.paues = False and the game continues. The display is updated using pygame.display.update() method.

The class has a method endScreen to represent the screen when the gameplay is over. It shows different text depending on which of the two players that wins, and displays the winner's score. The player has the option to restart the game or to exit the game. If SPACEBAR is pressed, then the game restarts and if ESCAPE is pressed or the action pygame.QUIT is registered, then the game is exited using pygame.quit() and sys.exit().

Main Implementation

Main module contains the Mayhem class witch handles all events and run the game. Mayhem contains the attributes that set the display, background image, clock (pygame.time.Clock()) and the fps.

The EventHandler method contains all non-collision events. using a for loop for all events in pygame.event.get(), it checks if event type is pygame.QUIT, then game quits. Using pygame.KEYDOWN the loop checks if escape key is pressed, if pressed pause method fro meny class is called. Another for loop is used to controll the player spaceships. Using pygame.key.get_pressed() the action of the key pressed will continue through the game loop. If the key pressed is L̈EFT̈he player wil rotate counter clockwise with speed equal to Config.diffAngle, if key pressed is R̈IGHTït will rotate clockwise with same speed. If T̈HRUSTänd fuel is greater than zero, the player will accelerate with acceleration from Config.acceleration, and the fuel will reduse by Config.diffFuel times time. If key is not "ẗhe thrust/acceleration is set to zero. If key is F̈IREp̈osOfCockpit method i called as position. Then weapon sprite.group() adds bullet with position, angle and screen as input.

collisionHandler handles all collision events in the game. Using a for loop of all spaceships/players in spaceshiplist collision for spaceships and all other sprites can be checked and determined. Pygame.sprite.spritecollide is used between spaceship and platformsprites and set to False. False means that the spaceship will not be removed from sprite.group(). Now using a if statement for when the collide, the velocities in x and y direction is set to zero, fueling is set to True and the angle is set to 90°in order to angle the player face up. Fueling is determined by another to if statements in the for loop. If it is fueling and fuel is less than max fuel:

$$\text{spaceship.fuel} -= \text{Config.diffFuel} * \text{time}$$

Now if fuel is greater than or equal to max fuel, fuel is set to max fuel. If the y position of the player is greater then the y position of the platform + 2, then fueling is set to false.

Collision between bullets and obstacle is checked by using pygame.sprite.spritecollide and set to True. this will remove the bullet that collides with obstacle from sprite.group(). Checking for collision between a player and obstacle is done in the

same way, but set to false. If they collide player is set to start and lives is reduced by one. Collision between players is set to false and players are reset at start position and life reduced by one. Next the game checks if a player is hit by another players bullet. If hit and player hit is not fueling, player hit is set to True and hitByOtherPlayer method is called. Lastly collision between bullets and platform. This collision is handled the same as for collision between bullet and obstacle.

Update method is used to call update method for player, platform and obstacle class. and then using pygame.display.update() function to update the screen/game.

Main method represents the main game loop. Time is defined to be:

$$\text{time} = \text{self.clock.tick(self.FPS)}/100$$

While loop if runGame is True, the background is drawn on the screen using self.SCREEN.blit(). Here the menu screens, event handler and collision handler i called. A for loop is used to check when the game ends. This is done by checking if the score is equal to 2 or if one player is out of lives. If one of this is true the endScreen is called displaying the winner and the score. From this menu there is a choice between restarting the game or to quit.

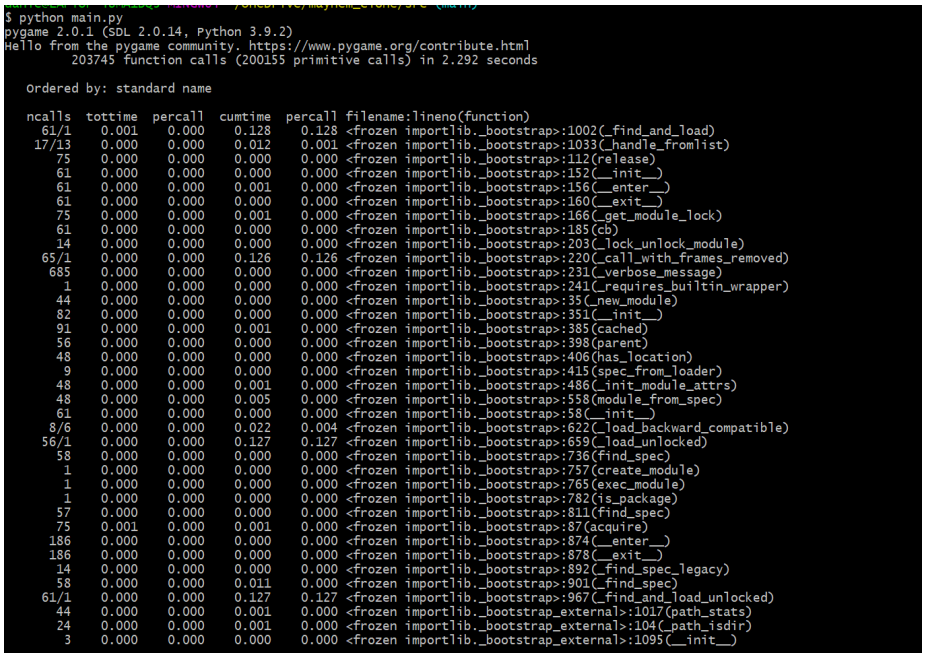
-

Evaluations

In this assignment there is a lot of requirements, that have all been fulfilled. We have implemented two spaceships which has four controls: rotation left, rotation right, thrust and fire. We have implemented one rectangular obstacle at the center of the screen. We have implemented collisions between walls, obstacle and between the two spaceships. Gravity acts on the spaceship and not on the bullets, but that was not required. Each player has a score that is displayed on the screen and that score is increased if the player shoots down another player, and decreases if the player crash with the other player, walls or obstacles. Each player has limited with fuel and needs to land on one of the platforms to refuel. We have not implemented scrolling window, but as stated in the assignment this is not a requirement.

We have implemented the code with modules each containing only one class object and one of the modules is called "config.py" file containing global configuration constants. The main game loop also have timing, and thus it is playable on different computers. The game is called using one line in a $if\_\_name\_\_ ==''\_\_main\_\_''$ block, all other code is implemented inside classes. All objects drawn on the screen is a subclass of pygame.sprite.Sprite module and is put into sprite groups using pygame.sprite.Group(). Updating and drawing is used with update() and draw(). All modules contains docstrings and all methods contains docstrings. The code is well commented, thus making it easy to understand in our opinion. Design pattern is not a requirement and we have

not implemented design patterns with the knowledge of doing so, if it is found in our submission.



```
$ python main.py
pygame 2.0.1 (SDL 2.0.14, Python 3.9.2)
Hello from the pygame community. https://www.pygame.org/contribute.html
         203745 function calls (200155 primitive calls) in 2.292 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
     61/1    0.001    0.000    0.128    0.128 <frozen importlib._bootstrap>:1002(_find_and_load)
    17/13    0.000    0.000    0.012    0.001 <frozen importlib._bootstrap>:1033(_handle_fromlist)
       75    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:112(release)
       61    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:152(__init__)
       61    0.000    0.000    0.001    0.000 <frozen importlib._bootstrap>:156(__enter__)
       61    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:160(__exit__)
       75    0.000    0.000    0.001    0.000 <frozen importlib._bootstrap>:166(_get_module_lock)
       61    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:185(cb)
       14    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:203(_lock_unlock_module)
     65/1    0.000    0.000    0.126    0.126 <frozen importlib._bootstrap>:220(_call_with_frames_removed)
      685    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:231(_verbose_message)
        1    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:241(_requires_builtin_wrapper)
       44    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:35(_new_module)
       82    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:351(__init__)
       91    0.000    0.000    0.001    0.000 <frozen importlib._bootstrap>:385(cached)
       56    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:398(parent)
       48    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:406(has_location)
        9    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:415(spec_from_loader)
       48    0.000    0.000    0.001    0.000 <frozen importlib._bootstrap>:486(_init_module_attrs)
       48    0.000    0.000    0.005    0.000 <frozen importlib._bootstrap>:558(module_from_spec)
       61    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:58(__init__)
      8/6    0.000    0.000    0.022    0.004 <frozen importlib._bootstrap>:622(_load_backward_compatible)
     56/1    0.000    0.000    0.127    0.127 <frozen importlib._bootstrap>:659(_load_unlocked)
       58    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:736(find_spec)
        1    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:757(create_module)
        1    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:765(exec_module)
        1    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:782(is_package)
       57    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:811(find_spec)
       75    0.001    0.000    0.001    0.000 <frozen importlib._bootstrap>:87(acquire)
      186    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:874(__enter__)
      186    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:878(__exit__)
       14    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:892(_find_spec_legacy)
       58    0.000    0.000    0.011    0.000 <frozen importlib._bootstrap>:901(_find_spec)
     61/1    0.000    0.000    0.127    0.127 <frozen importlib._bootstrap>:967(_find_and_load_unlocked)
       44    0.000    0.000    0.001    0.000 <frozen importlib._bootstrap_external>:1017(path_stats)
       24    0.000    0.000    0.001    0.000 <frozen importlib._bootstrap_external>:104(_path_isdir)
        3    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap_external>:1095(__init__)
```

Figur 2: Screenshot of the cProfiler output

Based on the cProfiler, we have no significant bottlenecks in our code and it seems like the implementation has a good performance.

Discussion

As stated in the Evaluation section, all requirements are met, but not without its issues. The motion of the spaceships is a bit off, in some cases it can have zero velocity or loose all velocity in the y-direction but keep velocity in x-direction. By reworking the code and making a better function to calculate velocity this issue can be avoided, but due to time restrictions our solution was what we went with. There is also a small issue where if the player velocity is to fast, the bullets fired will lagg behind. This can be a quick fix by either setting bullet velocity faster or by limiting the max velocity for the player. Choosing higher bullet velocity will require more frames per seconds because the bullets moves so fast that the computer does not register that the sprites is overlapping and the bullets tunnel through the other objects.

Conclusion

In this assignment, we have made a clone of the classic Mayhem game written by Espen Skoglund a former student with UiT. We have implemented the game using the concepts of object-oriented programming; that is classes, methods and inheritance.

The assignment is written i Python, which is an interpreted language with the help of Pygame library.

All requirements are met, there are some bugs with the movements, but overall the code runs fine.

Referanser

Python Software Foundation. Python language reference, version 3.9.2, 2021. URL `http://www.python.org/`.

A set of members in pygame community. Pygame community, version 1.9.5, 2019. URL `http://www.pygame.org/`.