University of Tromsø

INF-1400 OOP

Assignment 3 - Mayhem clone

Marius Mæland and Raymon S. Hansen

Department of Computer Science

April 14, 2016

1 Introduction

The assignment is to implement a clone of the old game Mayhem, but in this case we ended up with a space shooter death match type of game, however all of the requirements of the assignment are met.

1.1 Requirements

Requrements for this assignment is to create a multiplayer game where each player can control their spaceship with four buttons, when thrusting the spaceships should move in the direction it is pointing, and are to be rotated right or left to control where to go. The ships should also be able to fire bullets.

Other requirements for the game:

- Two space ships
- Four controls for each player
- Controls: rotate left, rotate right, thrust and fire
- Minimum one obstacle
- Spaceship can crash with other objects on the screen
- Gravity that acts on spaceships
- A score for each player displayed on screen
- Limited amount of fuel to the spaceships
- Refuel station/ refuel pick up

There are also some requirements for the code:

- The code should be split in minimum two files
- The main loop must have timing
- The game shall be started using:

```
if__name__ == '__main__':
```

- All visible objects shall subclass the pygame.sprite.Sprite class
- All modules, classes and methods shall contain docstrings

2 Technical Background

One of the requirements was that all moving objects in the game should sub-class pygames sprite module. The pygame Sprite module is designed to make simple game design easier for developers. It includes built in functions for referring to and working with coordinates, displaying images for a better graphical look as well as detecting collisions and working with objects/sprites in groups.

Sprite groups is practical because it makes the use of polymorphism very easy. The sprite module has a built in draw function which blits each sprites "image" attribute on a surface. We have also used this to simplify per frame update calls on many different objects.

Allthough not a specific requirement, we have implemented and thoroughly enjoyed making animations with sprite sheets. Linking the actual code and the image files has given us valuable insight into how code interacts with "the outside world". Cutting specific sections of a sprite sheet proved difficult at times, as each sheet had a different layout, size, padding, alpha channels etc.

3 Design

We chose to design the code so that each class had its own file to make the code easy to read, and then imported every file in to the game file to gather everything and run it in the game class. Each object that was needed got its own class, like player, explosion, bullet etc..

3.1 Player

The player has all the attributes needed to make a spaceship, along with all of the methods needed to make it move, rotate, shoot, die and so on. To add the flame sprite to the space ship sprite and make it rotate, we had to create a new surface then blit the ship and the flame on the new surface and then rotate the surface, this was partly done to avoid complex calculations regarding the placements of the sprites in relation to each other. The player makes an instance of bullets because the player is the one that knows where the bullets are to spawn on the screen and wich direction to send the bullets.

3.2 Bullet

The bullet have all the attributes and methods for the bullet behavior, it takes in a list of frames to animate.

3.3 Asteroid

Asteroid are animated, it takes in a list of the sprites to do the animation, but animates itself. The asteroid class takes inn width and height to make random scaled asteroids, and has a respawn method that respawns the asteroids at random location over or under the screen, with random x and y speed, but the y speed is set to posivite or negative in relation to witch side of the screen they spawn on, this is to make the asteroids always pass through the screen.

3.4 Explosion

The explosion class is just anitmating the explosion at the location it is called to, the explosion class also takes in width and height to scale it after the objects that explode.

3.5 Animation

This class animates pick up crystals, and the green gravity hole. It has the attributes for the crystals that apply the players random health and fuel on pick up, and have a static ammo amount. The crystals respawns on random location after picked up.

3.6 config

Config is the file with all the global variables, to prevent magic numbers, and makes it easy to just go into the config file to tweak the game. For more info on tweaking see README.

3.7 Game

The Game class is the class that handles all of the things that happend in the game, it creats a screen, loads almost all of the sprites when start up, handles keypress, collision checks, where to create the explosions, sets up player information, creates instances of the objects and draws everything on the screen.

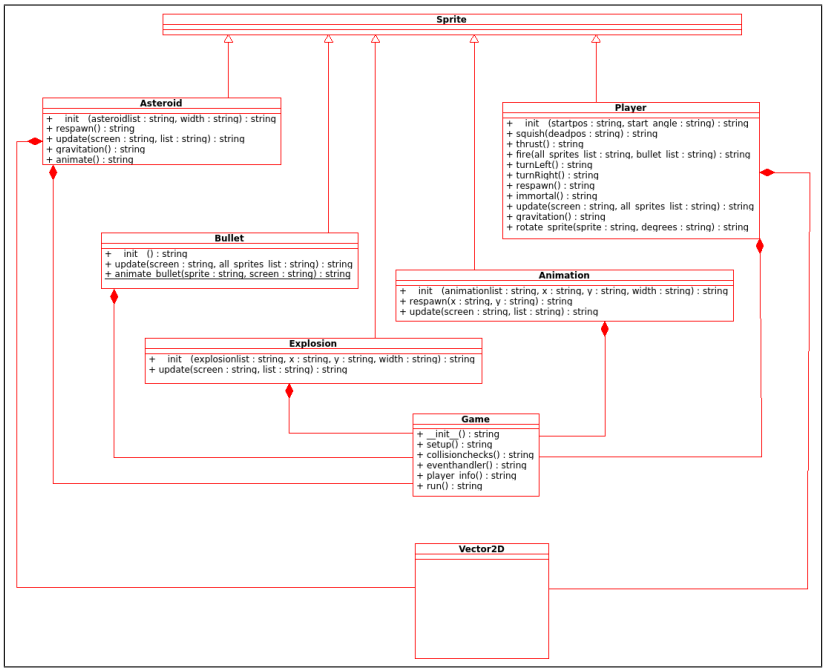

Figure 1: The class diagram

4 Implementation

The assignment was written in python using Sublime Text build 3103 on Linux Mint 17 "Qiana" and Linux Xubuntu 14.04 running kernel version 3.13.0-24 generic GNU/Linux. It also uses the pygame-library.

The various images and sprite sheets have all been edited using GIMP(GNU Image Manipulation Program).

Profiling tools pydoc and cprofiler.

5 Discussion

During tha writing of this assignment we have endeavoured to keep the architecture simple and easily understandable. In addition, we wanted it to be easy to expand with new ideas and elements. One of the reasons for working in a team was to experience how to manage one project with more writers. This forced us to carefully considder how encapsulation is handled. It must be possible for one person to work on some part of the code and the other works on another part, without worrying about making changes that ruin both writers contributions. As a result, the code has a fairly solid design on the whole. The game-class manages all the other classes as well as their interactions with each other(mainly when they crash!).

The explosion-class is a good example of solid and reusable code. It takes in alist of the frames to animate the explsion(this can then be any animation you like)as well as the point where you want it and the size. This made it very simple toanimate explsions when asteroids blow up, players die etc. The puff of dust when asteroids bump into eachother, is also an instance of the explosion-class.

A similar concept is used for the power ups as they are all instances of the animation-class They each get sent a different coloured list of frames to animate the crystals as well as placement and size.

A possible improvement on the explosion and animation would be to have a more general animation parent-class from which one could make power ups, explosions and so on inherit. Loading images and cutting subsurfaces from a sprite-sheet are heavy operations, therefore we have tried to do most of the loading in the game-class setup-method. The preloaded lists are then sent to each class where they are needed. The actual cutting of the animation-sheets are mostly done using nested for-loops to traverse the image. We made some attempts at abstracting this process out as it is done several times. This proved difficult however because of great variasions in the layout of the sprite-sheets. As a result, the setup-method has it's own sheetcutter method, there's aslo a general img list and some for-loops. This is a bit messy and should have been abstracted out or the sprite-sheets could be changed to fit within a chosen standard.

```
Thu Apr 14 14:18:28 2016    cProfile

         16112337 function calls in 643.904 seconds

   Ordered by: call count
   List reduced from 101 to 10 due to restriction <10>

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
  2011182    0.554    0.000    0.554    0.000 {method 'colliderect' of 'pygame.Rect' objects}
  2011182    1.845    0.000    2.399    0.000 sprite.py:1287(collide_rect)
  1492992    2.168    0.000    2.168    0.000 sprite.py:303(sprites)
  1446336    1.930    0.000    4.135    0.000 sprite.py:339(__iter__)
```

```
  1446336    0.444    0.000    0.444    0.000 {built-in method iter}
  1230017    1.079    0.000    1.079    0.000 precode.py:21(__init__)
   797797    0.324    0.000    0.324    0.000 {built-in method sqrt}
   797797    1.959    0.000    2.283    0.000 precode.py:70(magnitude)
   618347  344.058    0.001  344.058    0.001 {method 'blit' of 'pygame.Surface' objects}
   352654    0.701    0.000    1.001    0.000 precode.py:32(__add__)

Thu Apr 14 14:18:28 2016    cProfile

         16112337 function calls in 643.904 seconds

   Ordered by: internal time
   List reduced from 101 to 10 due to restriction <10>

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
   618347  344.058    0.001  344.058    0.001 {method 'blit' of 'pygame.Surface' objects}
    23328  102.980    0.004  102.980    0.004 {built-in method flip}
   113309  101.839    0.001  101.839    0.001 {built-in method rotate}
   153064   25.605    0.000   25.605    0.000 {built-in method from_surface}
   263887   13.013    0.000   13.013    0.000 {built-in method scale}
    37664   10.847    0.000   10.847    0.000 {method 'copy' of 'pygame.Surface' objects}
    36189    4.425    0.000  134.391    0.004 player.py:202(rotate_sprite)
    23328    4.039    0.000   14.607    0.001 game.py:161(collisionchecks)
   139968    2.228    0.000   16.848    0.000 asteroid.py:31(update)
  1492992    2.168    0.000    2.168    0.000 sprite.py:303(sprites)
```

The two lists above shows the top ten functioncalls sorted by number of calls and total time respectively. Not surprisingly, the built in sprite.collide rect takes up the highest number of calls. This is due to the enormous amount of collision-checks, done each frame. This also confirms that doing the rect check BEFORE maks, pays off.

In terms of time, almost half the total time of the testrun, is spent blitting. This is also expected as there are a large number of images to blit each frame. Other built in functions like rotate, mask from surface etc also range high. This leads us to think that the update-function in the player class perhaps could be optimized.

6 Conclusion

We had a great time creating this game, we met some problems adding vector calculations after everything was created. The sprite sheet cutting function seemed to be difficult to create generally to work in all our the different cases, therefore there is a cupple functions/methods to cut sprite sheets. A fun project that was enjoyed while making the game. The game looks nice and is fun to play.

References

[1] Philips Dusty *Python 3 Object Oriented Programming*. Packt Publishing, Second Edition, 2015.

[2] http://www.pygame.org/docs/

[3] http://millionthvector.blogspot.no/p/free-sprites_12.html

[4] http://opengameart.org/sites/default/files/styles/watermarked/public/asteroid_01_no_moblur.png

[5] http://commondatastorage.googleapis.com/codeskulptor-assets/explosion.hasgraphics.png

[6] http://www.spriters-resource.com/pc_computer/touhou123/sheet/33595/

[7] http://www.spriters-resource.com/pc_computer/touhou123/sheet/33594/

[8] http://www.spriters-resource.com/pc_computer/touhou123/sheet/27027/