Ben Lewis-Jones 2020 (N0928518)

Year 1

COMP10082-PROGRAMING STRAND

python project documentation

Table of Contents

[Brief 2](#_Toc58187292)

[What is the game 2](#_Toc58187293)

[How to play 3](#_Toc58187294)

[Features 3](#_Toc58187295)

[Main menu 3](#_Toc58187296)

[Leader board 3](#_Toc58187297)

[Game 3](#_Toc58187298)

[Game over 3](#_Toc58187299)

[Requirements Analysis 3](#_Toc58187300)

[Main menu 3](#_Toc58187301)

[Displaying the content 3](#_Toc58187302)

[Leader board 4](#_Toc58187303)

[Loading data 4](#_Toc58187304)

[Displaying content 4](#_Toc58187305)

[Game 4](#_Toc58187306)

[Displaying content 4](#_Toc58187307)

[Checking for events 4](#_Toc58187308)

[Score 4](#_Toc58187309)

[Collision detection 4](#_Toc58187310)

[Game over 5](#_Toc58187311)

[Making a text box 5](#_Toc58187312)

[Appending data to a file 5](#_Toc58187313)

[Design 6](#_Toc58187314)

[Main code overview 6](#_Toc58187315)

[Importing modules 6](#_Toc58187316)

[Global variables 7](#_Toc58187317)

[Classes 7](#_Toc58187318)

[PlayerClass 7](#_Toc58187319)

[Rock 10](#_Toc58187320)

[Score 11](#_Toc58187321)

[Defining functions 13](#_Toc58187322)

[eventCheckerQuit 13](#_Toc58187323)

[eventChecker 14](#_Toc58187324)

[baseSettingsUpdate 14](#_Toc58187325)

[displayObjects 14](#_Toc58187326)

[newRock 14](#_Toc58187327)

[collideCheck 15](#_Toc58187328)

[reset 15](#_Toc58187329)

[game 15](#_Toc58187330)

[scoreUploader 16](#_Toc58187331)

[UploadData 17](#_Toc58187332)

[downloadData 17](#_Toc58187333)

[leaderboard 18](#_Toc58187334)

[Menu 18](#_Toc58187335)

[Instantiating the classes and calling main menu 18](#_Toc58187336)

[Testing 18](#_Toc58187337)

[Does the program launch? 18](#_Toc58187338)

[When hover over play does the game start? 18](#_Toc58187339)

[When player hit rock does it go to game over? 19](#_Toc58187340)

[When hover over does leaderboard show? 19](#_Toc58187341)

[Does a new high score save? 19](#_Toc58187342)

[What I would change about my game 20](#_Toc58187343)

[Figure 1. Illustration of box collision algorithm 5](#_Toc58187344)

[Figure 2. highly abstracted view of program 6](#_Toc58187345)

[Figure 3. Importing modules code 7](#_Toc58187346)

[Figure 4. Declaring global variables 7](#_Toc58187347)

[Figure 5. Image showing all classes defined 7](#_Toc58187348)

[Figure 6. Player Class methods 8](#_Toc58187349)

[Figure 7. image of the \_\_init\_\_ method for the player Class 8](#_Toc58187350)

[Figure 8. code of the displaySelf method 9](#_Toc58187351)

[Figure 9. code of the boundarys method 9](#_Toc58187352)

[Figure 10. code of the reset method 9](#_Toc58187353)

[Figure 11. code of the y property 9](#_Toc58187354)

[Figure 12. code of the y setter 10](#_Toc58187355)

[Figure 13. rock class methods 10](#_Toc58187356)

[Figure 14. code of \_\_init\_\_ method 10](#_Toc58187357)

[Figure 15. code for the cornerTupleGenerator method 11](#_Toc58187358)

[Figure 16. code for the move method 11](#_Toc58187359)

[Figure 17. code for the displaySelf method 11](#_Toc58187360)

[Figure 18. code for the \_\_init\_\_ method 12](#_Toc58187361)

[Figure 19. increase score method 12](#_Toc58187362)

[Figure 20. displaySelf method 13](#_Toc58187363)

[Figure 21. scoreValue method code 13](#_Toc58187364)

[Figure 22. code for the reset method 13](#_Toc58187365)

[Figure 23. eventCheckerQuit 13](#_Toc58187366)

[Figure 24. eventChecker function 14](#_Toc58187367)

[Figure 25. baseSettingsUpdate function 14](#_Toc58187368)

[Figure 26. displayObjects 14](#_Toc58187369)

[Figure 27. newRock function 15](#_Toc58187370)

[Figure 28. collideCheck function 15](#_Toc58187371)

[Figure 29. reset function 15](#_Toc58187372)

[Figure 30. game function 16](#_Toc58187373)

[Figure 31. textbox in scoreUploader 17](#_Toc58187374)

[Figure 32. uploadData function 17](#_Toc58187375)

[Figure 33. downloadData function 17](#_Toc58187376)

[Figure 34. final block 18](#_Toc58187377)

# Brief

This section will cover what my initial goals were when embarking on this assignment.

## What is the game

The game will involve a spaceship (the player) on the left of the screen. The main objective of the game is not to crash your spaceship. The challenge is when asteroids are coming in from the right and to the left; the player must avoid these asteroids to survive. The longer the player survives, the greater their score.

## How to play

The player must move up and down to avoid the asteroids; this will be achieved by using the arrow keys. The player's speed will be fixed; however, the asteroid's velocity will not. The asteroid velocity will be random but will tend to increase as the score increases; this means that more skilled players will face as much of a challenge as a novice.

## Features

I will make the game as a series of pages, this will include a: main menu, leader board, game and game over screen.

### Main menu

This will be the simplest of the pages. This page will allow the player to either go and play the game, or look at the leader board. The player will be able to choose between the two by using their mouse and clicking on the mode they want to play.

### Leader board

This page will show the top 3 players accompanied with their scores; it will also have a main menu button.

### Game

This will launch the game as described above, once game over is reached, it will automatically bring up the game over screen.

### Game over

This page will show the player their score; it will also ask the player for their name. Once entered the game will then save their name and score to a bitstream file.

# Requirements Analysis

This section will take a more focused look on the requirements of each of the pages, and how I might go about solving them.

## Main menu

As described above this page will give the user the choice of a. starting the game or b. looking at the leader board. Breaking this challenge down we must first display the content, create buttons, look out for when the mouse is hovering over them and clicks on them, and then finally the relevant page should then be loaded.

### Displaying the content

To do this we must be able to create a window of relevant size, generate surfaces and render them onto the page. To accomplish this, I will use a module called pygame; pygame makes it a lot easier to create a window, render content, as well as event checking. I will be using pygame throughout the program.

## Leader board

This page will display the top three players and their scores. The first challenge is to get the data that for the players scores. The second is to display the content. Third create a main menu button.

### Loading data

I want to be able to save arrays and not simply just the name and score, I could just loop through the text file and recreate an array that way, but that would be inefficient. This means that the conventional way of just saving strings to a text file will not be satisfactory. Therefore I must save it to a bit stream file like .pkl or .dat. To accomplish this I will be using the pickle module, this module can save and load data to or from an .pkl or .pickle file.

### Displaying content

As I said above this will be done with pygame, I will not go into any more detail here since it is essentially the same. One thing I must be aware of is if there is not enough data to fill the content. For example, only two people have played the game, that means that there will only be two high scores. So, when the player goes to look at the leader board, it may error since there is no content for third place.

## Game

This page is easily the most complex of the pages. The challenges that I must overcome are: displaying content and delaying the next frame, check for numerous events, displaying all the objects, handle the score, check for collisions and then link onto the game over page.

### Displaying content

Here I must display all the objects onto the screen, and for each frame display their updated location. I must also note that each frame must be delayed, because if I didn’t, on a faster computer the game would run faster, and therefore be harder as well as unfair.

### Checking for events

The program must check for events, when I am referring to events, I mean actions that the user makes for example pressing a key or moving the mouse. So, say the user presses the up arrow, we would expect the spaceship to move upwards. Another event the program must look out for is the quit event, this occurs when the user presses the exit button on the window, this of course must close the program.

### Score

I will create the score as an object, this is because I would like different methods associated with it like, to display itself onto the screen. It also makes it easier to read the code and modify it at a later date.

### Collision detection

This is where the program will detect if the player has collided with a meteor.

There are two methods of going about this problem:

1. Create a mathematical representation of all the objects and if any share the same space then they are colliding. This is the most accurate of the methods, however it is only really good with simple objects say a circle or a square, since having a complex shape it would be unrealistic to create such vectors in code and computationally straining (makes the computer lag).
2. Create a ‘ruff’ mathematical representation of the object and if they are in the same space they are colliding. This is less accurate and less computationally straining, this is the method most AAA video game studios use.

My algorithm is based on the second of the two. It has two requirements; one object is larger than the other, and you know the dimensions of the object.

A picture containing shape

Description automatically generated

Figure 1. Illustration of box collision algorithm

Figure 1, illustrates the basis behind my algorithm. The rule is, if the smaller object and larger object have collided, then the corner of the smaller object must be within the larger object.

## Game over

This page will ask the user for their name and then save their name and score to a file. The challenges associated with this page are: make a text box, display content, append data to a file.

### Making a text box

I will do this by checking for if each key pressed, if the key pressed has a Unicode value then append that text to a surface. If the backspace button is pressed delete last value. If return is pressed user has completed their name, then move onto the next task.

### Appending data to a file

For this I will use the pickle module as mentioned above to load and store data to a bitstream. To append the data, I will first load the array stored in the file, then append the new data, then store it. If there is no array the program will error, so I will use a try/except statement, so if there is no data in the file or it contains nonrelevant data, it will be over written and store the new data.

# Design

I am going to abstract the program into understandable chunks using flow charts, I will then explain it, and finally show you the code in question. I will focus in on sections as further down.

## Main code overview

Diagram

Description automatically generated

Figure 2. highly abstracted view of program

This is the general structure of my code. The main bulk of the code is located within the functions or classes.

## Importing modules

Text

Description automatically generated

Figure 3. Importing modules code

In figure 3, I import pygame, random and pickle. I also initialise pygame, this must be done or pygame’s functionality will be very limited. Pygame will allow me to display content and check of events. Random will allow me to generate random numbers, I will use this a lot in the rock class (more on that later). Pickle is what I use to access data from the bitstream file.

## Global variables

Text

Description automatically generated

Figure 4. Declaring global variables

Figure 4 shows, the global variables I have made. I define the screenWidth and screenHeight, as variables so I can use them later, when positioning objects. This means that the window will be responsive if I change these variables. Screen variable just generates the screen. The rockList list, will hold the rock objects, this is so I can loop through them instead of hard coding them; this essentially makes the amount of rocks limit less.

## Classes

A picture containing text, indoor, screenshot

Description automatically generated

Figure 5. Image showing all classes defined

I am not going to display all the code on this walkthrough; however, I will show key points and describe what each method does.

### PlayerClass

The player class inherits the object class, I have done this so I can use descriptor properties.

Graphical user interface, application

Description automatically generated

Figure 6. Player Class methods

Figure 6 shows the methods I had made for the player class.

#### \_\_init\_\_

Text

Description automatically generated

Figure 7. image of the \_\_init\_\_ method for the player Class

This is a method that is run when the class is instantiated. In there I have made any variables that may be associated with the player class.

I would quickly like to note; the \_\_ prefix makes it so that I won’t accidently modify attributes in the main code, and that the self. prefix makes it so that, that specific attribute can only be accessed within that object.

The height and width attributes describe the height and width of the image, these will be used later in collision detection.

The \_\_x and \_\_y values describe the current location of the player.

The \_\_image attribute loads the image of the player class, ready to be used at a later date.

#### displaySelf

When this method is called, it will display the player in the next frame, with the players new x-axis and y-axis values.

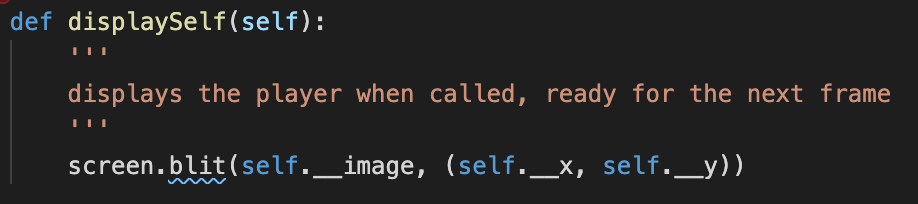


Figure 8. code of the displaySelf method

#### Boundarys

When this method is called, it will return a tuple, with what the y-axis is on the left and right of the object, and what the x-axis is on the top and bottom of the object.

Text

Description automatically generated

Figure 9. code of the boundarys method

#### reset

When this method is called, it will reset the y-axis value, to what it was when the object was instantiated.

Text

Description automatically generated

Figure 10. code of the reset method

#### Y property

This is essentially a getter, for the y-axis. This is achieved with the descriptor property.

Text

Description automatically generated

Figure 11. code of the y property

#### Y setter

This is a setter for y, y’s value will be incremented by the value of newY, if the player will go out of bounds then y’s new value will be rejected.

Text

Description automatically generated

Figure 12. code of the y setter

### Rock

This is the class for the meteors that I mentioned at the top of this document.

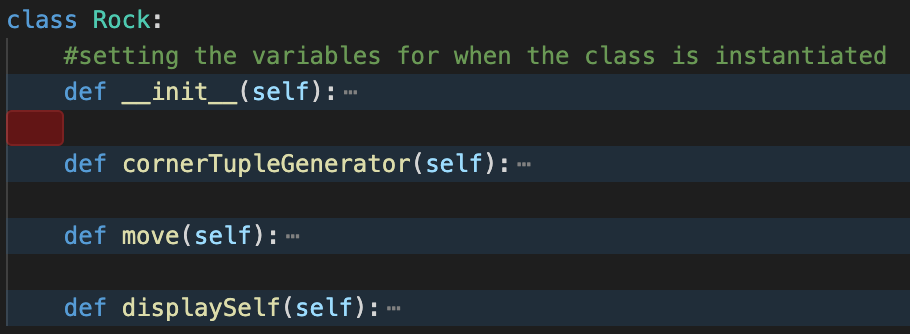


Figure 13. rock class methods

#### \_\_init\_\_

Again this is where I have set all my variables for the class.

Text

Description automatically generated

Figure 14. code of \_\_init\_\_ method

The height and width attributes set the dimensions for the object, that will be used in collision detection.

\_\_image loads the image for the rock.

xLocationRock and yLocationRock, both describe the objects current location. Note here, that the values assigned to them here, are the ‘spawn’ locations of the rocks. This means that the rock will spawn all the way to the right, and anywhere along the y-axis.

xVelocityRock and yVelocityRock, both describe the velocity of the rock and how far it should move per a frame, these values random however they will increase as the score increases.

#### conerTupleGenerator

This method when called will generate a tuple of the objects corners coordinates, and return it.

Text

Description automatically generated

Figure 15. code for the cornerTupleGenerator method

#### move

This method when called will move the object by the predefined velocities.

Text

Description automatically generated

Figure 16. code for the move method

#### displaySelf

This method when called will display the object in the next frame, with the new coordinates.

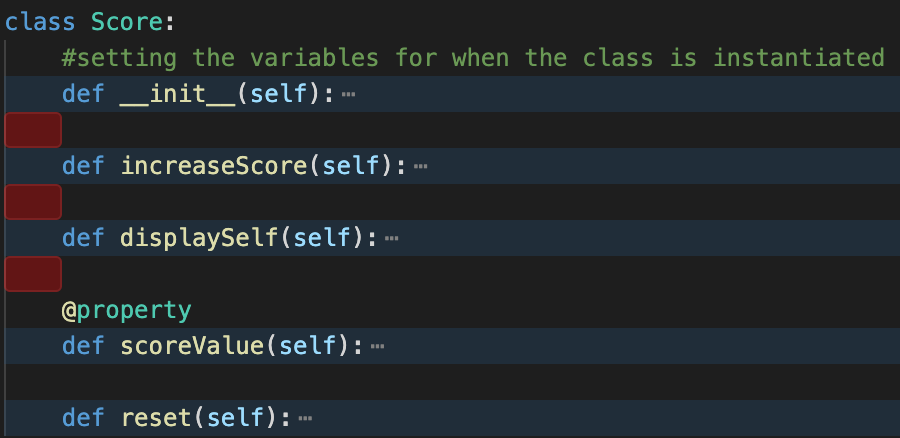
Text

Description automatically generated

Figure 17. code for the displaySelf method

### Score

This is the class for the scoring system.



#### \_\_init\_\_

Text

Description automatically generated

Figure 18. code for the \_\_init\_\_ method

Score holds the current value of the score. When the game starts its value is 0.

xLocationScore and yLocationScore, define where the score will appear on the screen, while playing the game.

font attribute holds the font type and size of the text.

#### increaseScore

This increments the score by 5 each time it is called.

Text

Description automatically generated

Figure 19. increase score method

#### displaySelf

Displays the new score ready for the next frame.

Text

Description automatically generated

Figure 20. displaySelf method

#### scoreValue

Acts as a getter and returns the current value of score.

Graphical user interface

Description automatically generated

Figure 21. scoreValue method code

#### reset

Sets the value of score back to 0.

Text

Description automatically generated

Figure 22. code for the reset method

## Defining functions

### eventCheckerQuit

This function when run will check to see if the user has pressed the quit button, the cross in the corner of a window, if it has been pressed then the game will quit.

Text

Description automatically generated

Figure 23. eventCheckerQuit

It works by looping through all the events that have been made, if the user presses the quit button then the QUIT event will be found and the game will quit.

### eventChecker

This function will also loop through all the events, if the user presses or holds the up arrow the spaceship, will go up and if held the spaceship will continue to go up, the reverse will happen with the down arrow.

Text

Description automatically generated

Figure 24. eventChecker function

### baseSettingsUpdate

This function will take a input of a time delay, it will set the background colour and also set the title of the window.

Text

Description automatically generated

Figure 25. baseSettingsUpdate function

### displayObjects

This function will call upon all the displaySelf methods from the objects.

Text

Description automatically generated

Figure 26. displayObjects

### newRock

Has a random chance of spawning a new rock.

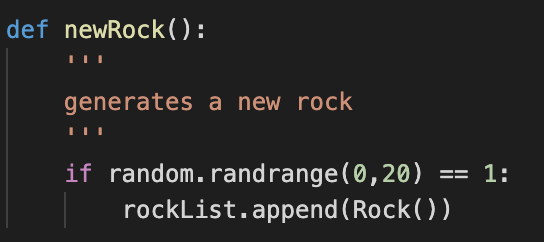


Figure 27. newRock function

### collideCheck

Text

Description automatically generated

Figure 28. collideCheck function

Calls in the global variable running, this is so that if there is a collision the game will end on the next iteration.

The for loop checks each rock, to see if any of its corners are within the players bounds. If it is then running is set to false.

### reset

Text

Description automatically generated

Figure 29. reset function

This function is run at the start of the game function, and resets all variables that have been changed, back to what they were when the game started.

### game

This function runs the other functions we have just spoken about.

Text

Description automatically generated

Figure 30. game function

First, I declare the global variable running and set its value to True. I understand that this is bad practice however, I find it helps with the readability of the code. The program then resets all values, it then runs all the functions we defined earlier. Note that I update the display at the end, and also the game function returns the score.

### scoreUploader

This is another page, this shows the users score, and will ask them to enter their name into a text box, the data will then be passed into the uploadData function, which will then upload the data. Most of the code is similar to other functions so I am not going to show it, but I will show the code for the text box.

Text

Description automatically generated

Figure 31. textbox in scoreUploader

This essentially checks for events, if it is a key down event then the letter will be appended to the textEntered variable, this variable will then be displayed on the screen. If enter is pressed, then the uploadData function is run. If backspace is pressed a letter is removed off the end of the variable.

### UploadData

Text

Description automatically generated

Figure 32. uploadData function

Loads data from a bitstream file, into a variable called data. It will then append data to be added, and then saves data back into the file. If there is no data or the data is not compatible, then the file is erased and new data is stored.

### downloadData

Text

Description automatically generated

Figure 33. downloadData function

This function load the data from the bitstream file, then orders it based on score. Then returns the ordered list.

### leaderboard

This function displays the leaderboard page, it shows the top three scores and associated names. It also has a button to go back to the homepage. If there is no data for the top 3 scores then the program will display ‘No sortedLeaderboard’ in that positions place.

This function possesses many of the same attributes as other pages.

If the mouse hovers over the main menu button it will take you back to the main menu.

### Menu

This function displays the main menu, it shows a game button and a leaderboard button, if the user hover over one, they will be taken to the game or the leader board respectively.

## Instantiating the classes and calling main menu

Finally I instantiate the classes into objects, and call main menu.

Text

Description automatically generated

Figure 34. final block

# Testing

## Does the program launch?

Graphical user interface, website

Description automatically generated

## When hover over play does the game start?

Graphical user interface, application

Description automatically generated

## When player hit rock does it go to game over?

A picture containing application

Description automatically generated

## When hover over does leaderboard show?

Graphical user interface, application

Description automatically generated

## Does a new high score save?

 Graphical user interface, application

Description automatically generated

# What I would change about my game

A large issue was the fact that I couldn’t get the mouse button down function to work. I believe this to be because, I use a mouse with only one button, which may cause some issues. If I could have gotten it to I would have used it on the buttons, on the main menu and leaderboard pages.

I would also implement a more advanced collision detection system; at the moment it is possible for the ship to appear to not collide with the rocks and still be game over. I would try and make something where it looks at the transparency of each pixel, and if two pixels that are not transparent collide, it would then call game over.

One final thing I would add is more a diverse range of rocks, so for example small and large rocks, this would be quite easy to implement. However, to do this under my current collision detection system would be a bad idea, since there would be large gaps where the hit box is and there is no visible image.