**Final Project**

**Algorithm and Programming**

Project Name: “Space Rocks”

Student Name: Abdullah Akmal Sutoyo

Student ID: 2602239320

Class: L1BC

# **Table of Contents**

**Cover Page**

**Contents**

**A. Description**

I. Introduction

II. The function of the program

**B. Design**

I. Parts of the program and requirements

II. Function of each part of the program

**C. Implementation**

I. Classes Diagram

II. Use Case Diagram

III. Activity Diagram

IV. Extensibility

V. Explanation of the functions made and used

**D. Lessons learned**

I. Using Pygame

II. Using the OS module

III. Error handling and debugging

**E. Evaluation**

I. Does the program work properly

II. Future improvements that can be done

III. Reflection

**F. Evidence of a working program**

I. Testing and pictures

# **A. Description**

## **I. Introduction**

After it was announced there was a final project for Algorithm and Programming, I started browsing the internet for simple/intermediate Python projects. I looked for a project that suits my interests and level of understanding in the Python language, and I ended up finding a project that was the perfect fit for me. The project was making an Asteroids game called “Space Rocks” using Pygame.

https://realpython.com/asteroids-game-python

I used the tutorial above to help me with the basic structure of the code, and then added more things so that the user can have a more immersive experience. I used the PyCharm IDE for this project and I will be posting all the code along with this report on my repository which can be accessed through my GitHub account here:

https://github.com/ilovegaming42069/AlgoProFinalProject

## **II. The function of this program**

The function that this program serves is that of an asteroid game that can provide entertainment for the player. The game is played by moving your spaceship, which can shoot lasers to destroy asteroids. The objective of the game is to destroy all the asteroids using the laser as fast as you can. While this game does not have any practical uses in real life, it is a good time killer if you have free time or when you are bored.

# **B. Design**

## **I. Parts of the program and requirements**

Requirements:

* PyGame: Used to design the overall game and visualize it into a window.
* Random: Used to randomize attributes of game objects

## **II. Function of each part of the program**

In this program I used a window with the size of 800x600 and a capped FPS of 60. The program starts out with the starting screen which shows the name of the game and two button inputs to start or exit the game. If the user clicks on the start button, the game will load, and it will immediately start the game.

Now that the user has started the game, the user can now see a spaceship that the user can control in the middle of the screen and asteroids that the user must destroy, floating around the screen. The user can also see a stopwatch in seconds to indicate how many seconds has passed and the shortest time any user has completed the game in that local device. The shortest time will be saved on a txt file. The background music will also change. The following are the controls for Space Rocks:

* UP Arrow: Increase speed
* DOWN Arrow: Decrease speed
* RIGHT Arrow: Rotate right
* LEFT Arrow: Rotate left
* SPACEBAR: Shoot lasers
* ESCAPE: Pause

The user must then try to destroy all the asteroids as fast as possible. The asteroids will all have random velocities between 1 to 3. If a laser from the spaceship collides with the asteroid, it will split into two smaller pieces, and they will split into random directions. Asteroids will no longer split if the asteroid has reached its smallest size. If the user’s spaceship collides with the asteroid, the user fails to complete the game and a “You Lose!” text will appear on the screen. After 3 seconds, it will then go back to the starting screen. If the user pauses the game, all the in-game objects will disappear, and the timer will stop. The user can then either resume the game, go back to the starting screen, or exit the game. If the user has destroyed all asteroids, a “You Win!” text will appear on the screen. If the user plays the game for the first time or if they set a record for the time, a text will also appear indicating that the game has a new lowest time record in that local device. Like when you lose, it will also go back to the starting screen after 3 seconds.

# **C. Implementation**

## **I. Classes Diagram**

|  |
| --- |
| **SpaceRocks** |
| * finish * playing * duration * screen * timer * font * background * boom * clock * message * asteroids * bullets * spaceship |
| + main\_loop()  + \_init\_pygame()  + menu()  + paused()  + \_handle\_input()  + \_process\_game\_logic()  + \_draw()  + \_get\_game\_objects(self) |

**Figure 4:** Class diagram of the spacerocks class.

|  |
| --- |
| **GameObject** |
| * position * sprite * radius * velocity * click |
| + draw(surface)  + move(surface)  + collides\_with(other\_obj) |

**Figure 5:** UML Class diagram of the chesspieces class.

|  |
| --- |
| **Spaceship** |
| * (inherit position, sprite, and velocity) * create\_bullet\_callback * laser\_sound * direction |
| + (inherited methods from GameObject class)  + rotate(clockwise)  + draw(surface) (override from GameObject)  + accelerate()  + decelerate()  + shoot() |

**Figure 6**: UML Class diagram for individual pieces.

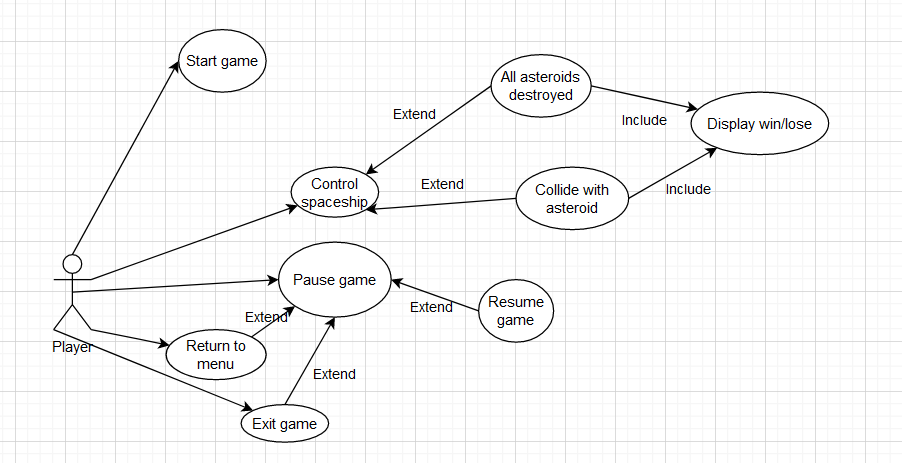
|  |
| --- |
| **Asteroid** |
| * (inherit position, sprite, and velocity) * create\_asteroid\_callback * size |
| + (inherited methods from GameObject class)  + split() |

**Figure 6**: UML Class diagram for individual pieces.

|  |
| --- |
| **Bullet** |
| * (inherit position, sprite, and velocity) |
| + (inherited methods from GameObject class)  + move(surface) (override from GameObject) |

**Figure 6**: UML Class diagram for individual pieces.

## **II. Use-case Diagram**



## **III. Activity Diagram**

Diagram

Description automatically generated

## **IV. Extensibility**

1. Comment lines are used to further explain the functions of each function and method and the purpose of each of them to help other programmers.
2. Variables use meaningful identifiers to help other coders to understand easier.
3. Class diagram and flowchart to make it easier for others to understand and to easily identify features.

## **V. Explanation of all the functions made and used**

### *game.py*

This file contains the class SpaceRocks, and in the class there are three local variables.

MIN\_ASTEROID\_DISTANCE: This variable sets the minimal distance of the spawning position of the asteroid and the spaceship (Integer: 250)

NUMBER\_ASTEROID: This variable determines the number of asteroids that will be spawned (Integer: 3)

FPS: This variable determines the FPS of the game. It is also used for the timer in the game. (Integer: 60)

* \_\_init\_\_(self):
  + Attributes of the class
  + Attribute self.finish, which is to determine if the round has finished or not. (Boolean: False)
  + Attribute self.playing, which is to determine if the game is playing

(Boolean: True)

* + Attribute self.duration, which is used to count down 3 seconds after game ends. Set to 180 because of FPS variable being set to 60, since 180 / 60 = 3 (Integer: 180)
  + Attribute self.screen, which is used to set the window size. On this project, it is set to 800\*600
  + Attribute self.timer, which will count up every 1 second, starts at 0

(Integer: 0)

* + Attribute self.font, which is used to display the text that will appear in the game
  + Attribute self.background, which is used to display the text that will appear in the game
  + Attribute self.boom, which contains the sound effect when a bullet collides with the asteroid. (Volume set to 0.1)
  + Attribute self.clock, which is used to cap the FPS
  + Attribute self.message, which is used to determine what text will be displayed

(String: **Empty**)

* + Attribute self.asteroids, which is a list containing all the asteroids in the game that is currently displayed in the game
  + Attribute self.bullet, which is a list containing all the bullets that is currently displayed in the game
  + Attribute self.spaceship, which contains the Spaceship class and the bullet list
* main\_loop(self):
  + Main loop of the game (calls menu(), \_handle\_input(), \_process\_game\_logic(), \_draw())
  + Contains the background music for the starting menu and the game itself (Volume set to 0.3)
* \_init\_pygame(self):
  + Contains the name of the game to be displayed on the border
  + Attribute self.best\_time calls the function get\_time() from utils.py
* menu(self):
* Beginning section of the game
* Load and blit the title of the game above a box, and other images indicating where to press for players to play or exit the game
* Gets your mouse position, and depending on where player left clicks, the game will either start or exit
* paused(self):
* Pause section of the game
* Load and blit other images indicating where to press for players to resume, return to menu or exit the game
* Gets your mouse position, and depending on where player left clicks, the game will either resume, return to menu or exit
* \_handle\_input(self):
* Handles the input of the game
* Gets keyboard event, and depending where the players press down on the keyboard, the game will do different things
* \_process\_game\_logic(self):
* Handles the game logic, such as all the game object movements, what happens when spaceship collides with asteroid, and what happens when bullet collides with asteroid
* Displays you win/you lose after the round ends
* \_draw(self):
  + Blits all game objects into the game
  + Blits the timer and the best time score
* \_get\_game\_objects(self):
  + Returns all game objects
* Notes:
  + Pygame is imported
  + Functions like get\_random\_position, load\_sprite, print\_text, print\_time, get\_time are imported from utils.py
  + Classes like Spaceship and Asteroid are imported from models.py

### *utils.py*

This file contains functions that is used on both main.py and models.py.

* load\_sprite(name, with\_alpha=True):
* make a path format so that I could load images easily
* wrap\_position(position, surface):.
* returns a vector that is the new wrapped position of images
* get\_random\_position(surface):
* returns a vector that is a random position for the asteroids
* get\_random\_velocity(min\_speed,max\_speed)
* returns a vector that is the random speed and the direction of the asteroid
* load\_sound(name):
* make a path format so that I could load sound effects easily
* print\_text(surface, text, font, color=Color("tomato")):
* Used to print You Win or You Lose after the round ends
* print\_time(surface, text, font, position, color=Color("tomato")):
* Used to print the timer and the best time during the game
* Notes:
  + random is imported
  + Functions like Color, Sound, Vector2, and load are all imported from pygame

### *models.py*

This file contains all the game object classes, and for all the classes there is a global variable.

UP: Default position for all game objects, since we want it to face up, the value inside is Vector2(0, -1).

Class GameObject:

* \_\_init\_\_( self, position, sprite, velocity):
* Attribute self.position to determine the position of the game object
* Attribute self.sprite to load the image
* Attribute self.radius to get the radius of the image
* Attribute self.velocity to determine the speed of the game object
* Attribute self.click to keep count on how many times the user pressed the accelerate button
* draw(self, surface):
* Blits game objects
* move(self, surface):
* Changes the position of game objects
* collides\_with(self, surface):
* Returns a Boolean to determine whether an object is colliding with one another

Class Spaceship: This class is a child class of GameObject. This class has 3 local variables:

MANEUVERABILITY: Determines the manoeuvrability rate (set to 5)

ACCELERATION: Determines the speed of the spaceship when user accelerates (set to 0.25)

BULLET\_SPEED: Determine the bullet speed (set to 10)

* \_\_init\_\_( self, position, create\_bullet\_callback):
* Inherit attributes from GameObject
* Attribute self.create\_bullet\_callback to add bullets into spaceship
* Attribute self.laser\_sound to load the sound effect of the bullet
* Attribute self.direction to determine the direction of the spaceship, set to default
* draw(self, surface):
* Blits the object after getting the new position and angle
* rotate(self, clockwise=True):
  + Rotates the spaceship, rotation rate will be determined MANEUVERABILITY variable
* accelerate(self):
* Accelerates the spaceship everytime user accelerates, self.click will add by 1 everytime the user accelerates. Acceleration rate is determined by ACCELERATION variable
* decelerate(self):
* Decelerates the spaceship everytime user decelerates and self.click > 0, self.click will be substracted by 1 everytime the user decelerate. Deceleration rate is also determined by ACCELERATION variable
* shoot(self):
* Shoots the bullet, with the bullet speed determined by the BULLET\_SPEED variable. It also plays the sound effect

Class Asteroid: This class is a child class of GameObject.

* \_\_init\_\_( self, position, create\_asteroid\_callback, size=3):
* Inherit attributes from GameObject
* Attribute self.create\_asteroid\_callback to add the list of asteroids
* Attribute self.size to determine the size of the asteroids (set to 3)
* Scaling the asteroid sizes, this is because we want the size to change after the asteroid splits
* split(self):
  + Split function, if self.size > 1, it will split the asteroid into 2, and both asteroids will be one size smaller than before.

Class Bullet: This class is also a child class of GameObject

* \_\_init\_\_( self, position, velocity):
* Inherit attributes from GameObject
* move(self, surface):
  + Changes the position of the bullet

# **D. Lessons learned**

## **I. Using Pygame**

Pygame is the module I used to make this game. I first got introduced to Pygame when Mr Jude gave us an assignment for the Week 9 of the forum week. Using Pygame at first was very simple and seamless and it made me realize just how fun making something from scratch really is. I learnt how to load sprites and place them in particular positions of the window and also how to manage actions such as the escape button, the arrow buttons, and spacebar. I also learned how to use vectors and rotozoom using Pygame. Last but not least, I also learned how to use background music and sound effects using Pygame.

## **II. Using random**

I used the random module to randomize the attributes of the game object, such as the positions of the asteroids and the velocity of the asteroids. This isn’t particularly hard, because in my opinion random is a very basic module and anyone with a basic understanding of Python can use it.

## **III. Error Handling and Debugging**

One of the most frustrating parts when becoming a coder is having to deal with debugging your own program. I feel the same way too, as having to solve a problem that you caused can be quite confusing. I have devised my own way to figure out what went wrong in my code to lead to something not going the way I want to. First, I will check in what line is the error in. Then I will try checking if I have a typo in my code. Maybe I accidentally missed a bracket or a comma when writing it. If there is nothing wrong, then there is something wrong with the logic of the code and I will need to check what kind of error is the code having. Usually when running a problematic code, the terminal will give you a description of the error, and usually I will try to fix based on what the problem is based on what the terminal said. If I can’t fix it, I will just copy what the terminal said and paste it on Google to find out more on that type of error. If it is too complicated to fix, I will just try switching the logic of my code.

# **E. Evaluation**

## **I. Does the program work properly?**

The Space Rocks program is essentially just a code I copied from the internet, and then I added more stuff to it (deceleration, pause menu, start menu, bgm, etc.). The program works really well as a game and has basically no errors that can occur that hasn’t been dealt with. However, the controls are a bit weird and clunky, and I hope in the future I can find a good way to balance the control system. Overall, I would say that this program is good in terms that it reaches its goal as a program.

## **II. Future Improvements that can be done**

The improvements I would like to add to this game would be a difficulty system which allows users to choose the difficulty of the game (each difficulty will have a different best time score), an improvement on the pause menu (music pauses, users can edit the graphics and volume of the sound effects and bgm) and a resizable screen. The other improvements aren’t necessarily that hard, but making a resizable screen is hard, as I have to resize the in game objects every time the screen resizes.

## **III. Reflection**

With this project I now know my limitations as a programmer and I wish that I could improve on this limitations. I am satisfied with how my program ended up being, even though I know that there are many improvements that can still be done with the existing program. I will continue to work on this program after submitting this report and I will keep improving it so that it can be something that I am proud of. After all this is one my first Python projects and I will remember the lessons, I learnt when making this all throughout my life.

# **F. Evidence of working program**

Graphical user interface

Description automatically generated

Menu/beginning section

Graphical user interface, website

Description automatically generated

Pause section

A screenshot of a galaxy

Description automatically generated with low confidenceA picture containing text

Description automatically generatedIn game

You won, but no best time

Website

Description automatically generatedYou won and you got new best time

A picture containing text, outdoor object

Description automatically generatedYou lost