Program Design Methods
Final Project Report

BINUS
UNIVERSITY
INTERNATIONAL

Student Information:

Ariel Putra Agustinus / 2440100273

Class:

L1BC

Lecturer(s):

Ida Bagus Kerthyayana Manuaba

Chapter I – Introduction

When trying to come up with an idea for the final project, I could not stick to one idea. I knew I wanted to make some sort of game from the beginning, but I could not decide on what type of game to make. After further brainstorming, I decided to recreate the classic game ‘Asteroids’. Creating the game seemed to be possible for my level of understanding and it seemed like a fun project.

To build my project, I used the python module PyGame. I chose PyGame for the vast amount of documentation and help I can look for if I ever get stuck. This is my first time trying to program a game from scratch and I needed to spend time learning the basics of PyGame. I feel like making this game will be a good challenge to test and sharpen my programming skills since it didn’t seem impossible and that I could do it if I spend some time learning.

Chapter II – Project Specifications

Project Purpose:

To make an interesting project for entertainment and create something fun. To learn python programming and game design.

Project Audience:

People who like to play games, especially classic ones and people who would like to learn about the basics of game design and the algorithms used.

Project Aim:

The aim of this project is to create a functional game that people can play and have fun with. To be able to do the game's intended purpose without constantly crashing or breaking. To create a game that requires minimal computing power.

Project Requirements:

- Fluid player controls
- Asteroids that break into multiple pieces when broken
- Minimal to no amount of computer lag when playing
- A game that is fun to play

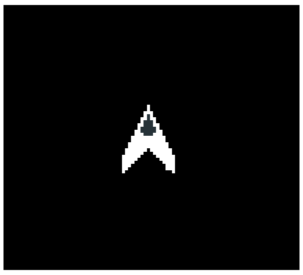
Chapter III – Solution Design

1. External Libraries Used

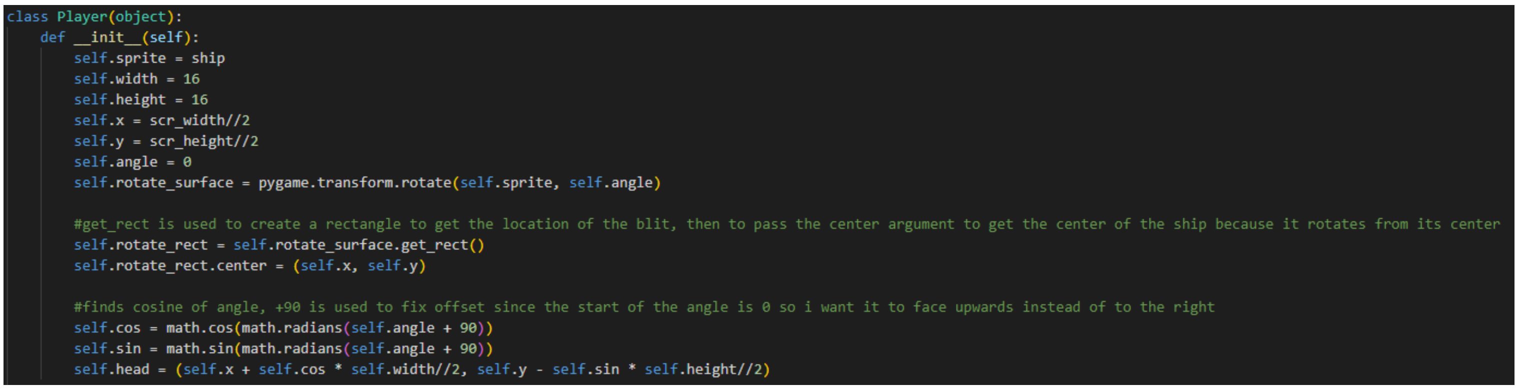
PyGame

PyGame is used as the basis of this entire project. It creates and draws objects onto the game screen. It basically controls everything seen on screen.

2. Player Class



The player can go in 4 directions (left, right, up, down) and can turn a full 360 degrees. The player is viewed from a top-down angle.

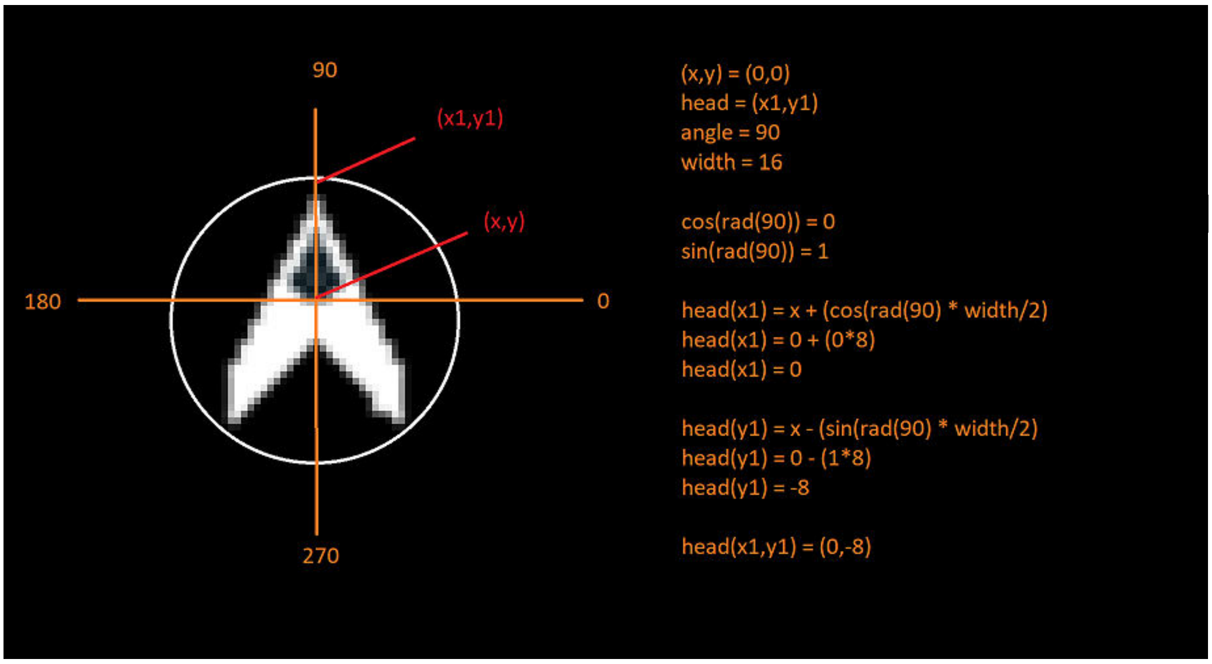


```python
class Player(object):
    def __init__(self):
        self.sprite = ship
        self.width = 16
        self.height = 16
        self.x = scr_width//2
        self.y = scr_height//2
        self.angle = 0
        self.rotate_surface = pygame.transform.rotate(self.sprite, self.angle)

        #get_rect is used to create a rectangle to get the location of the blit, then to pass the center argument to get the center of the ship because it rotates from its center
        self.rotate_rect = self.rotate_surface.get_rect()
        self.rotate_rect.center = (self.x, self.y)

        #finds cosine of angle, +90 is used to fix offset since the start of the angle is 0 so i want it to face upwards instead of to the right
        self.cos = math.cos(math.radians(self.angle + 90))
        self.sin = math.sin(math.radians(self.angle + 90))
        self.head = (self.x + self.cos * self.width//2, self.y - self.sin * self.height//2)
```

To control the player, the up and down arrow keys are used to move the player forwards and backwards while the left and right arrow keys turn the player on the center axis.

The player class has a width and height value of 16x16 which is the sprite size. The location of the player starts at the middle of the screen at start up. It also has an angle value which can be updated to tilt the player.

I use the pygame.transform.rotate() function to control the rotation of the ship. In order to make the ship rotate from the center axis, I use get_rect() and center() to change the center coordinates of the sprite from the top left to the middle.

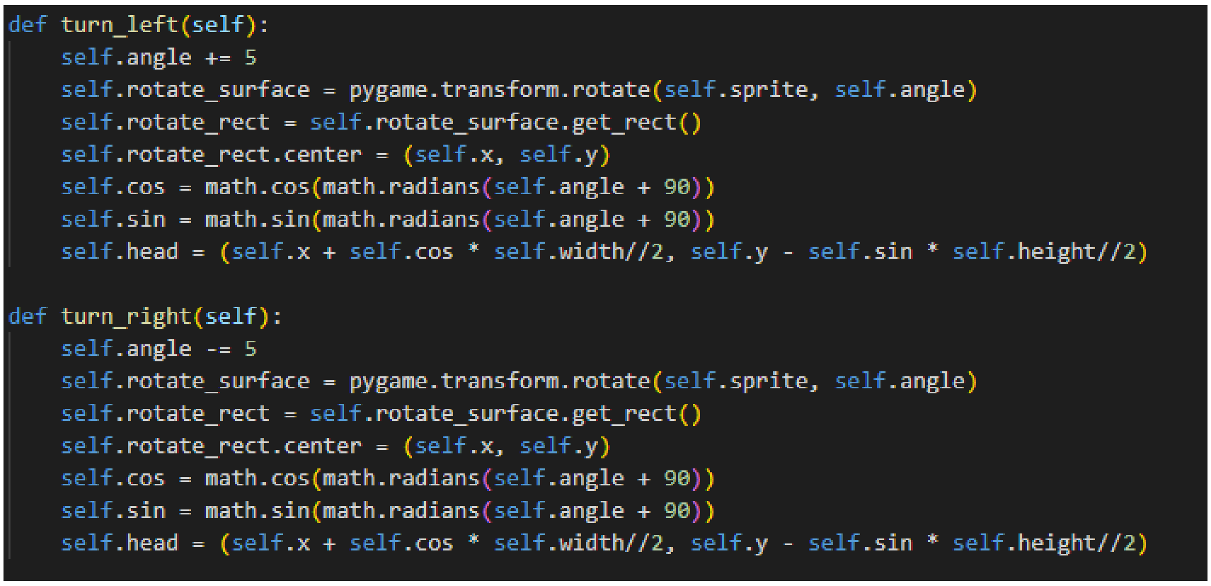


```
(x,y) = (0,0)
head = (x1,y1)
angle = 90
width = 16

cos(rad(90)) = 0
sin(rad(90)) = 1

head(x1) = x + (cos(rad(90) * width/2)
head(x1) = 0 + (0*8)
head(x1) = 0

head(y1) = x - (sin(rad(90) * width/2)
head(y1) = 0 - (1*8)
head(y1) = -8

head(x1,y1) = (0,-8)
```

The head value is used to determine the point that the bullet shoots from. It calculates the position of the front of the ship and it updates every time the ship rotates so the bullet will always come out in front.

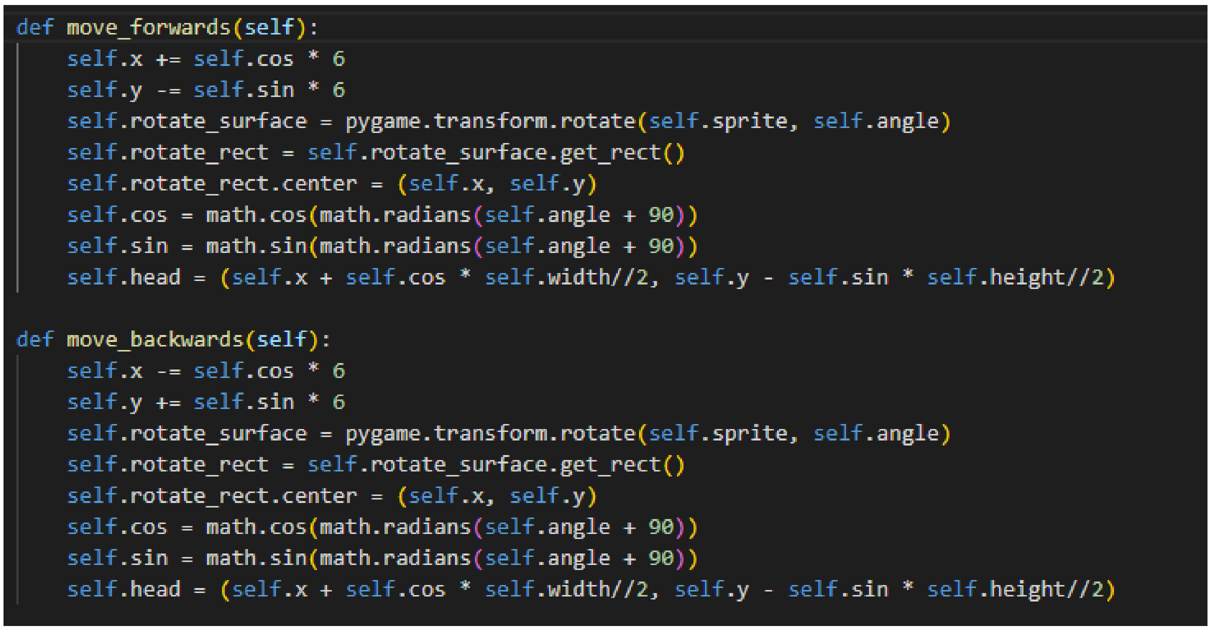
To get the x value of the head, we add the center value (the player's x value) with the width of the player/2 (so it starts from the middle point) multiplied by the cosine of the angle the player is facing.

The same goes for they y value of the head but instead of adding to it, we subtract it because the y value increase the lower you are on the screen.



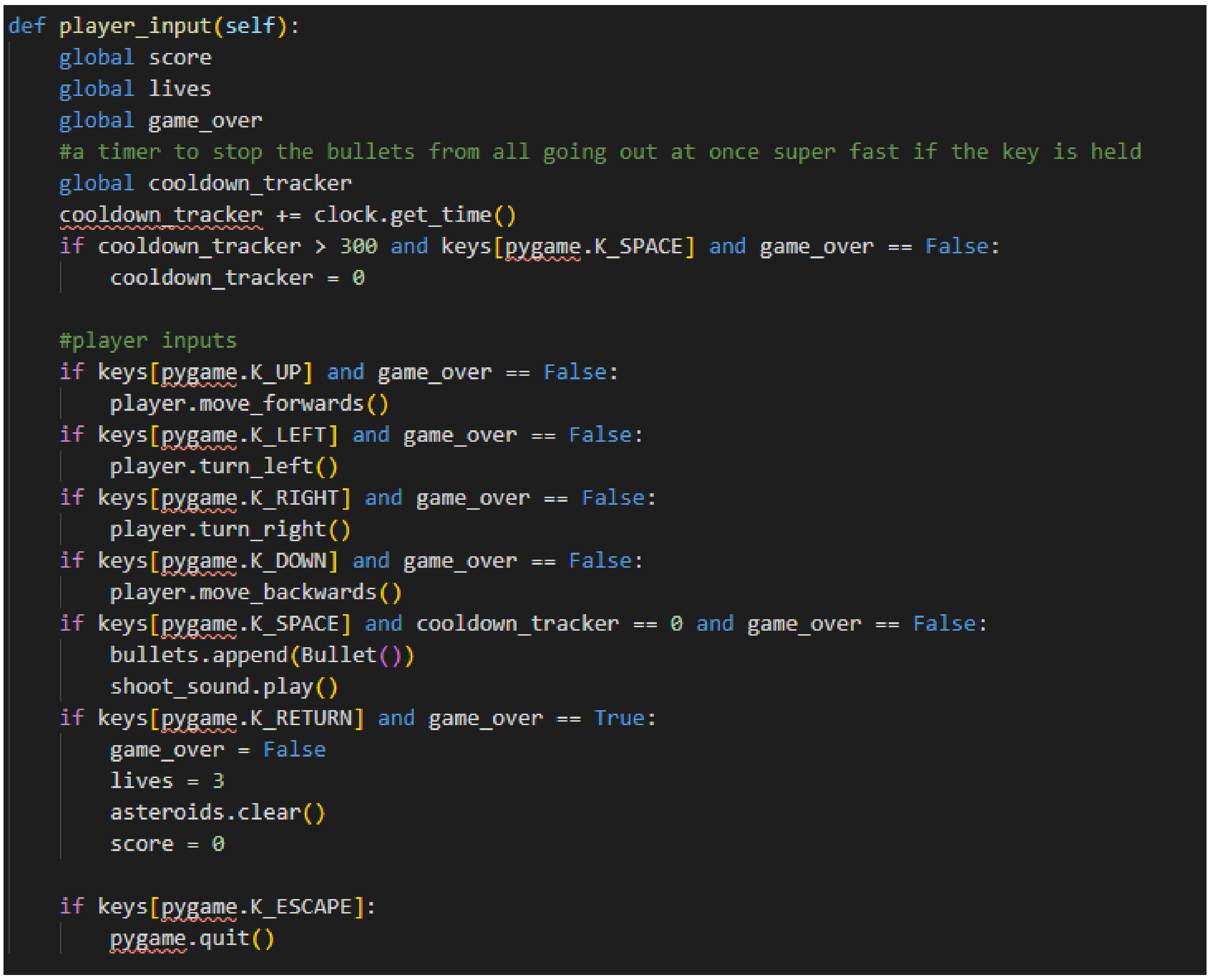
```python
def turn_left(self):
    self.angle += 5
    self.rotate_surface = pygame.transform.rotate(self.sprite, self.angle)
    self.rotate_rect = self.rotate_surface.get_rect()
    self.rotate_rect.center = (self.x, self.y)
    self.cos = math.cos(math.radians(self.angle + 90))
    self.sin = math.sin(math.radians(self.angle + 90))
    self.head = (self.x + self.cos * self.width//2, self.y - self.sin * self.height//2)

def turn_right(self):
    self.angle -= 5
    self.rotate_surface = pygame.transform.rotate(self.sprite, self.angle)
    self.rotate_rect = self.rotate_surface.get_rect()
    self.rotate_rect.center = (self.x, self.y)
    self.cos = math.cos(math.radians(self.angle + 90))
    self.sin = math.sin(math.radians(self.angle + 90))
    self.head = (self.x + self.cos * self.width//2, self.y - self.sin * self.height//2)
```

To turn left and right, I use the aptly named function turn_left() and turn_right(). All these functions do is add the angle in increments of 5 and updates the other values accordingly when it is called.



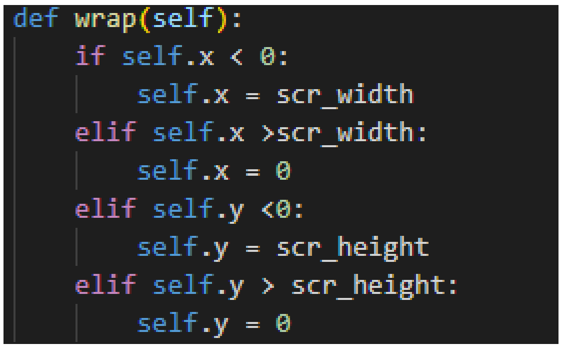
```python
def move_forwards(self):
    self.x += self.cos * 6
    self.y -= self.sin * 6
    self.rotate_surface = pygame.transform.rotate(self.sprite, self.angle)
    self.rotate_rect = self.rotate_surface.get_rect()
    self.rotate_rect.center = (self.x, self.y)
    self.cos = math.cos(math.radians(self.angle + 90))
    self.sin = math.sin(math.radians(self.angle + 90))
    self.head = (self.x + self.cos * self.width//2, self.y - self.sin * self.height//2)

def move_backwards(self):
    self.x -= self.cos * 6
    self.y += self.sin * 6
    self.rotate_surface = pygame.transform.rotate(self.sprite, self.angle)
    self.rotate_rect = self.rotate_surface.get_rect()
    self.rotate_rect.center = (self.x, self.y)
    self.cos = math.cos(math.radians(self.angle + 90))
    self.sin = math.sin(math.radians(self.angle + 90))
    self.head = (self.x + self.cos * self.width//2, self.y - self.sin * self.height//2)
```

To move forwards and backwards, it is essentially the same as turning left and right, the difference being instead of changing the value of the angle, this function changes the x and y position of the player by multiplying the cosine and sine values of the player by a set velocity such as 6.



```
def player_input(self):
    global score
    global lives
    global game_over
    #a timer to stop the bullets from all going out at once super fast if the key is held
    global cooldown_tracker
    cooldown_tracker += clock.get_time()
    if cooldown_tracker > 300 and keys[pygame.K_SPACE] and game_over == False:
        cooldown_tracker = 0

    #player inputs
    if keys[pygame.K_UP] and game_over == False:
        player.move_forwards()
    if keys[pygame.K_LEFT] and game_over == False:
        player.turn_left()
    if keys[pygame.K_RIGHT] and game_over == False:
        player.turn_right()
    if keys[pygame.K_DOWN] and game_over == False:
        player.move_backwards()
    if keys[pygame.K_SPACE] and cooldown_tracker == 0 and game_over == False:
        bullets.append(Bullet())
        shoot_sound.play()
    if keys[pygame.K_RETURN] and game_over == True:
        game_over = False
        lives = 3
        asteroids.clear()
        score = 0

    if keys[pygame.K_ESCAPE]:
        pygame.quit()
```

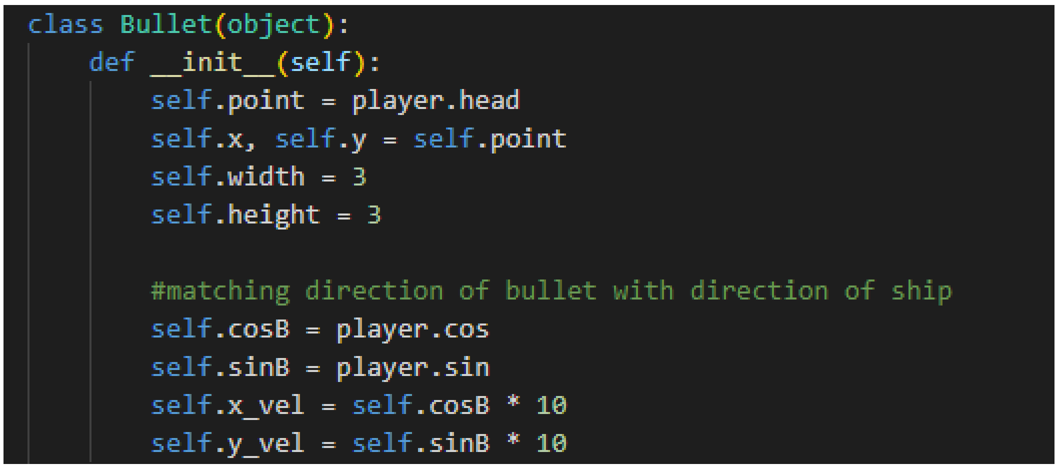
The player_input() function is used to track all the inputs the player makes. The variable cooldown_tracker is used to add a delay/cooldown for every time the player holds down the space key to shoot. Without it, bullets would fly out of the ship rapidly at once and the game would be a too easy.



```
def wrap(self):
    if self.x < 0:
        self.x = scr_width
    elif self.x >scr_width:
        self.x = 0
    elif self.y <0:
        self.y = scr_height
    elif self.y > scr_height:
        self.y = 0
```

The wrap() function is used to make the player wrap around the screen instead of exiting it. So if the player were to go up, they would reappear at the bottom. Same goes for left and right movement as well.

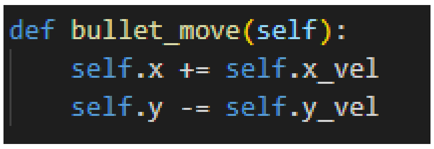
3. Bullet Class



```python
class Bullet(object):
    def __init__(self):
        self.point = player.head
        self.x, self.y = self.point
        self.width = 3
        self.height = 3

        #matching direction of bullet with direction of ship
        self.cosB = player.cos
        self.sinB = player.sin
        self.x_vel = self.cosB * 10
        self.y_vel = self.sinB * 10
```

When initiating the bullet class, we assign its coordinates to be the coordinates of player.head. This is done so the bullets initial position when shot is always at the front of from all directions

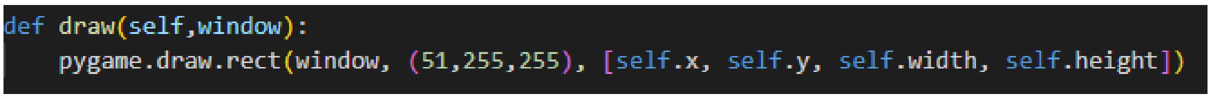
We match the bullets cosine and sine values to the player to make the bullets shoot out in the



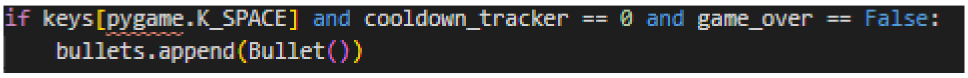
```python
def bullet_move(self):
    self.x += self.x_vel
    self.y -= self.y_vel
```

direction the player is facing. The velocity of the bullet is just the cosine and sine values multiplied by a velocity; in this case it is 10.

Moving the bullet is just a matter of adding the velocity to the x and y values of the bullet.



```python
def draw(self,window):
    pygame.draw.rect(window, (51,255,255), [self.x, self.y, self.width, self.height])
```



```python
if keys[pygame.K_SPACE] and cooldown_tracker == 0 and game_over == False:
    bullets.append(Bullet())
```

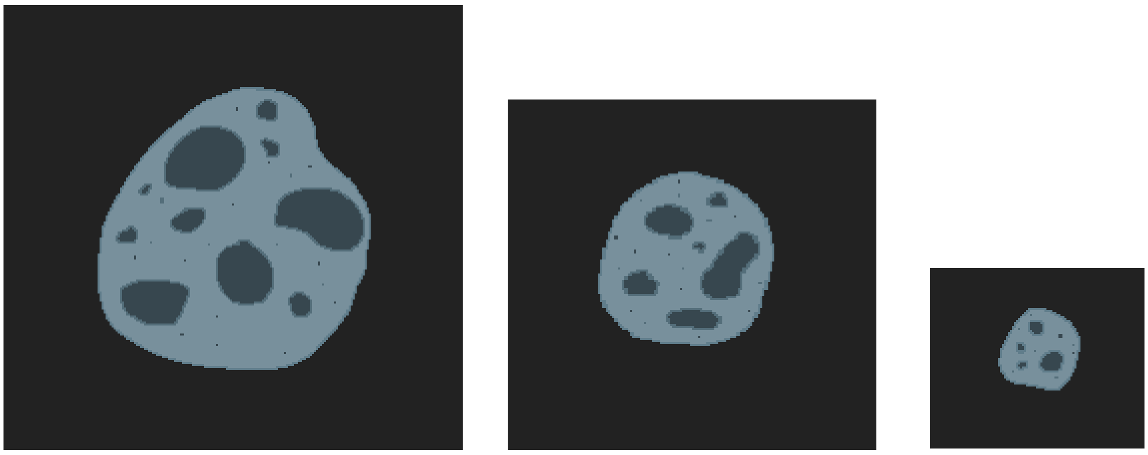
The bullet is a 3x3 rectangle in the cyan color.

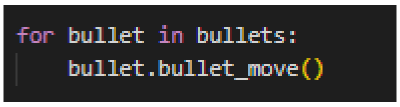
To spawn the bullet, I create an empty list called bullets and every time the player presses spacebar, it would append Bullet() to the list. In the redrawGameWindow() function, it draws an asteroid for every iteration in the list. Then, in the main loop there is a for loop that calls for the bullet_move() function for every iteration in the list.

3. Asteroid Class

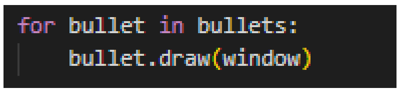
While playing the game, asteroids spawn at random points outside of the screen at a set interval and move in a random trajectory towards the middle of the screen.

There are 3 different sizes of asteroids. There is one at 50x50, 100x100, and 150x150 pixels.

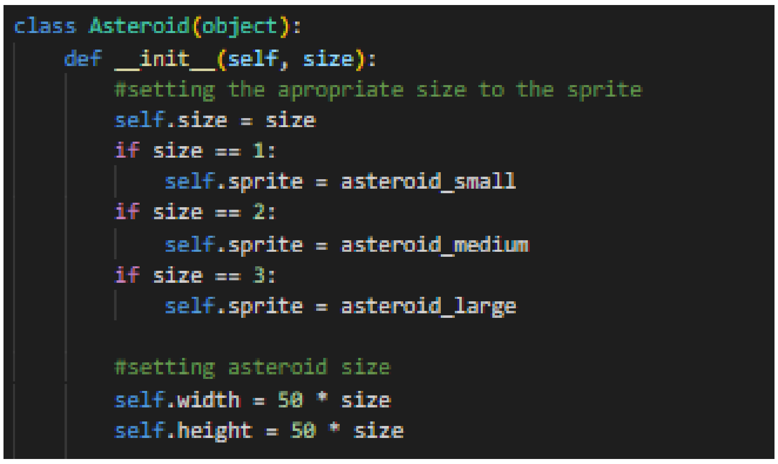




```
for bullet in bullets:
    bullet.bullet_move()
```

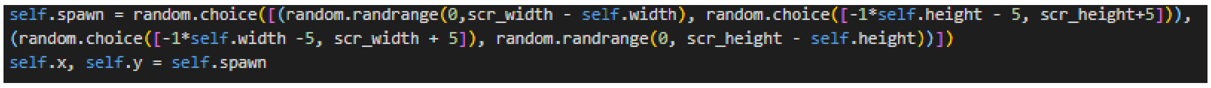


```
for bullet in bullets:
    bullet.draw(window)
```



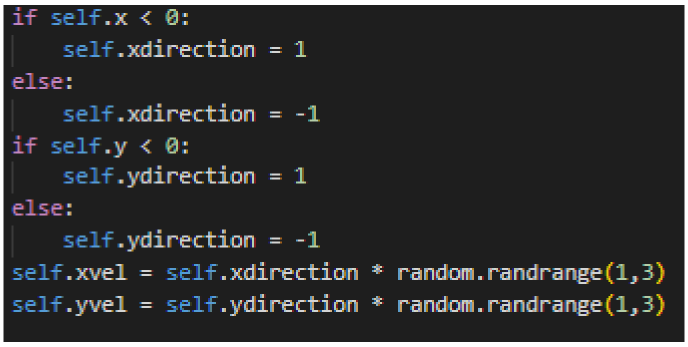
```python
class Asteroid(object):
    def __init__(self, size):
        #setting the apropriate size to the sprite
        self.size = size
        if size == 1:
            self.sprite = asteroid_small
        if size == 2:
            self.sprite = asteroid_medium
        if size == 3:
            self.sprite = asteroid_large

        #setting asteroid size
        self.width = 50 * size
        self.height = 50 * size
```

The asteroid class takes 1 argument, size that determines which asteroid is spawned.



```python
self.spawn = random.choice([(random.randrange(0,scr_width - self.width), random.choice([-1*self.height - 5, scr_height+5])),
(random.choice([-1*self.width -5, scr_width + 5]), random.randrange(0, scr_height - self.height))])
self.x, self.y = self.spawn
```

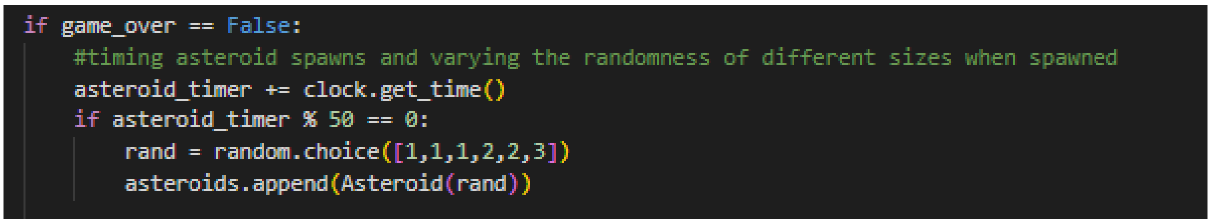
The asteroids spawn at a random location outside of the screen. To randomize the spawn location, I use the random module. The randomizer will return a location the screen that is either outside of the screen on the x axis or the y axis.



```python
if self.x < 0:
    self.xdirection = 1
else:
    self.xdirection = -1
if self.y < 0:
    self.ydirection = 1
else:
    self.ydirection = -1
self.xvel = self.xdirection * random.randrange(1,3)
self.yvel = self.ydirection * random.randrange(1,3)
```

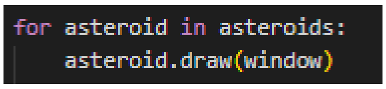
To determine the direction the asteroid goes after it has been spawned, it goes through a simple check to find out where it is outside the screen, whether it is x positive or negative and whether it is y positive or negative.

The velocity of the asteroid is also randomized to give it a little bit of variety.



```
if game_over == False:
    #timing asteroid spawns and varying the randomness of different sizes when spawned
    asteroid_timer += clock.get_time()
    if asteroid_timer % 50 == 0:
        rand = random.choice([1,1,1,2,2,3])
        asteroids.append(Asteroid(rand))
```

To spawn the asteroids, I created a list called asteroids and a variable called asteroid_timer. The use of asteroid_timer is for timing the asteroid spawn using the pygame.time feature. For every time the timer is divisible by 50, it would spawn an asteroid. The size randomness of the asteroid is set using the random.choice() function that chooses a random value in a list. There is a bigger chance of spawning a small asteroid compared to a big one. It would then

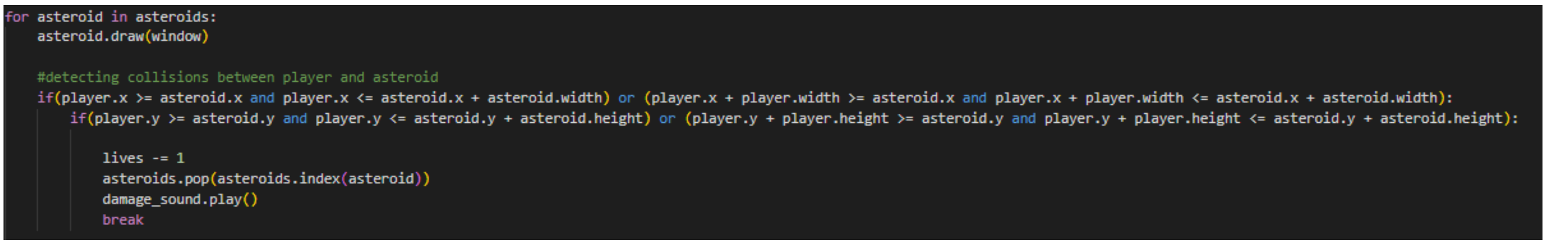


```
for asteroid in asteroids:
    asteroid.draw(window)
```

append it to the list asteroids with Asteroid(rand).

The asteroid would then be drawn on the screen with a loop in the redrawGameWindow() function.

4. Collisions

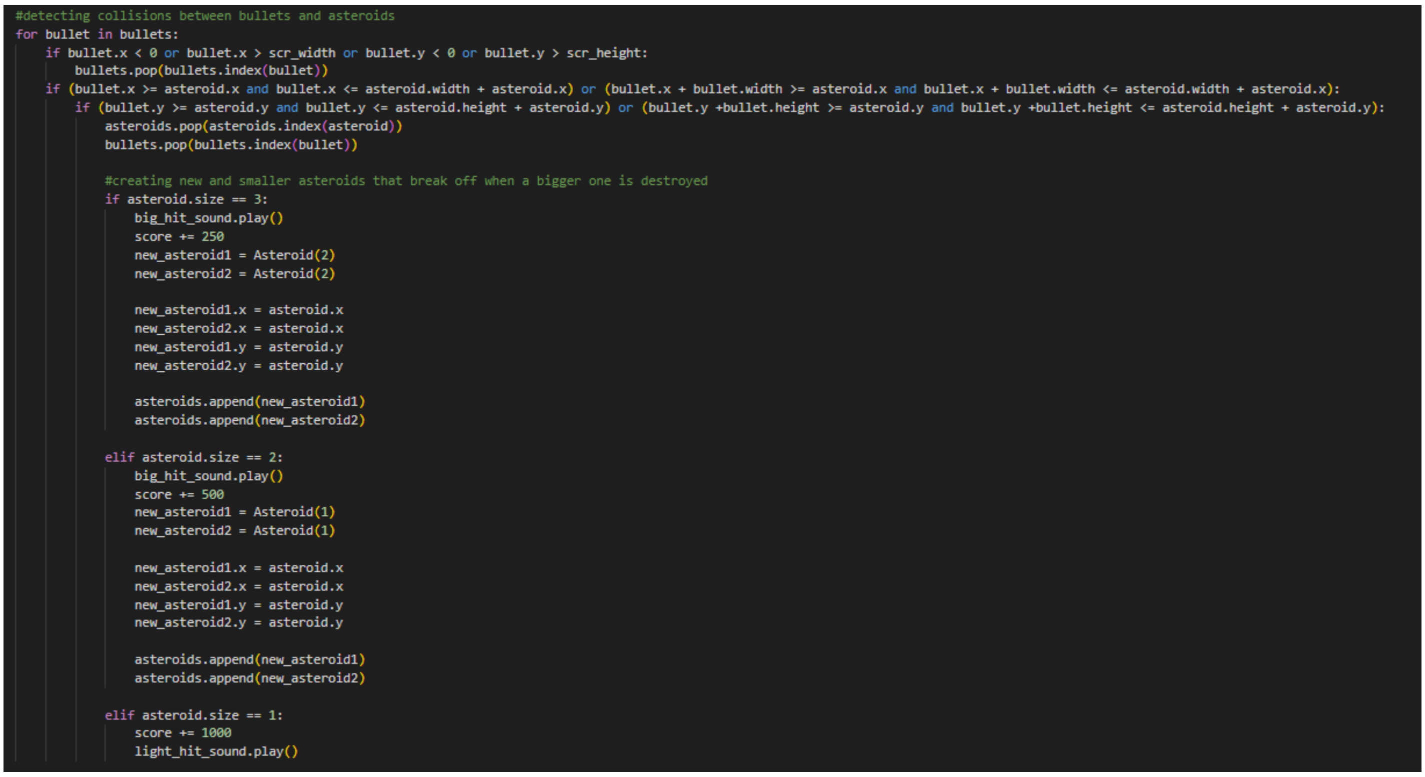


```
for asteroid in asteroids:
    asteroid.draw(window)

    #detecting collisions between player and asteroid
    if(player.x >= asteroid.x and player.x <= asteroid.x + asteroid.width) or (player.x + player.width >= asteroid.x and player.x + player.width <= asteroid.x + asteroid.width):
        if(player.y >= asteroid.y and player.y <= asteroid.y + asteroid.height) or (player.y + player.height >= asteroid.y and player.y + player.height <= asteroid.y + asteroid.height):

            lives -= 1
            asteroids.pop(asteroids.index(asteroid))
            damage_sound.play()
            break
```

a. Player and Asteroids

To detect collisions between the player and the asteroid, the program checks whether the player's x and y coordinates are inside the coordinates of the asteroid plus its dimension since the coordinates of the asteroid are on the top left side.

If the player is struck by an asteroid, it would reduce the player's lives count by 1 and pop the asteroid out of the asteroids list.



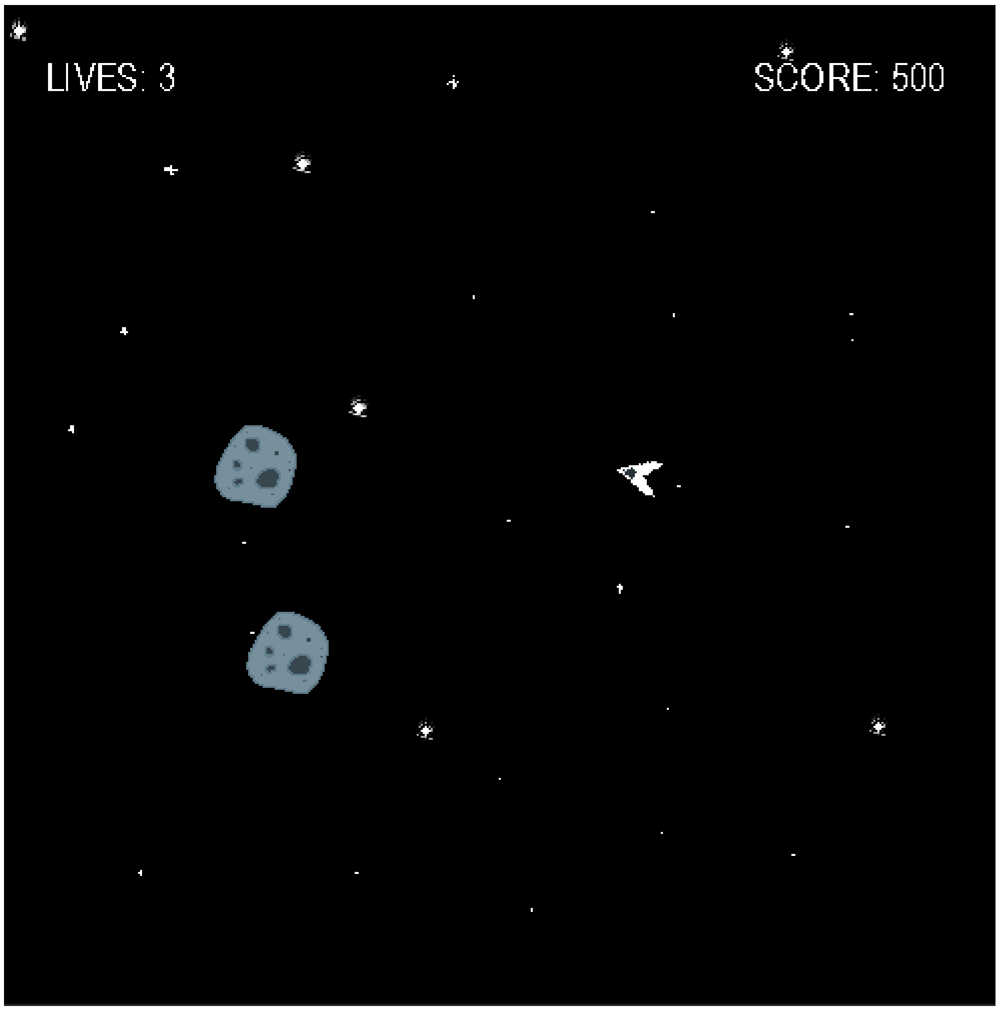
```python
#detecting collisions between bullets and asteroids
for bullet in bullets:
    if bullet.x < 0 or bullet.x > scr_width or bullet.y < 0 or bullet.y > scr_height:
        bullets.pop(bullets.index(bullet))
    if (bullet.x >= asteroid.x and bullet.x <= asteroid.width + asteroid.x) or (bullet.x + bullet.width >= asteroid.x and bullet.x + bullet.width <= asteroid.width + asteroid.x):
        if (bullet.y >= asteroid.y and bullet.y <= asteroid.height + asteroid.y) or (bullet.y +bullet.height >= asteroid.y and bullet.y +bullet.height <= asteroid.height + asteroid.y):
            asteroids.pop(asteroids.index(asteroid))
            bullets.pop(bullets.index(bullet))

            #creating new and smaller asteroids that break off when a bigger one is destroyed
            if asteroid.size == 3:
                big_hit_sound.play()
                score += 250
                new_asteroid1 = Asteroid(2)
                new_asteroid2 = Asteroid(2)

                new_asteroid1.x = asteroid.x
                new_asteroid2.x = asteroid.x
                new_asteroid1.y = asteroid.y
                new_asteroid2.y = asteroid.y

                asteroids.append(new_asteroid1)
                asteroids.append(new_asteroid2)

            elif asteroid.size == 2:
                big_hit_sound.play()
                score += 500
                new_asteroid1 = Asteroid(1)
                new_asteroid2 = Asteroid(1)

                new_asteroid1.x = asteroid.x
                new_asteroid2.x = asteroid.x
                new_asteroid1.y = asteroid.y
                new_asteroid2.y = asteroid.y

                asteroids.append(new_asteroid1)
                asteroids.append(new_asteroid2)

            elif asteroid.size == 1:
                score += 1000
                light_hit_sound.play()
```

b. Bullets and Asteroids

Detecting a collision between a bullet and an asteroid uses the same method as before. It checks whether the position of the x and y are inside the x and y position of the asteroid plus its dimensions.

If a bullet hits an asteroid, it adds to the players score and break off into 2 smaller sized asteroids. To do this, the program appends 2 new asteroids with the same x and y values of the old, destroyed asteroid.

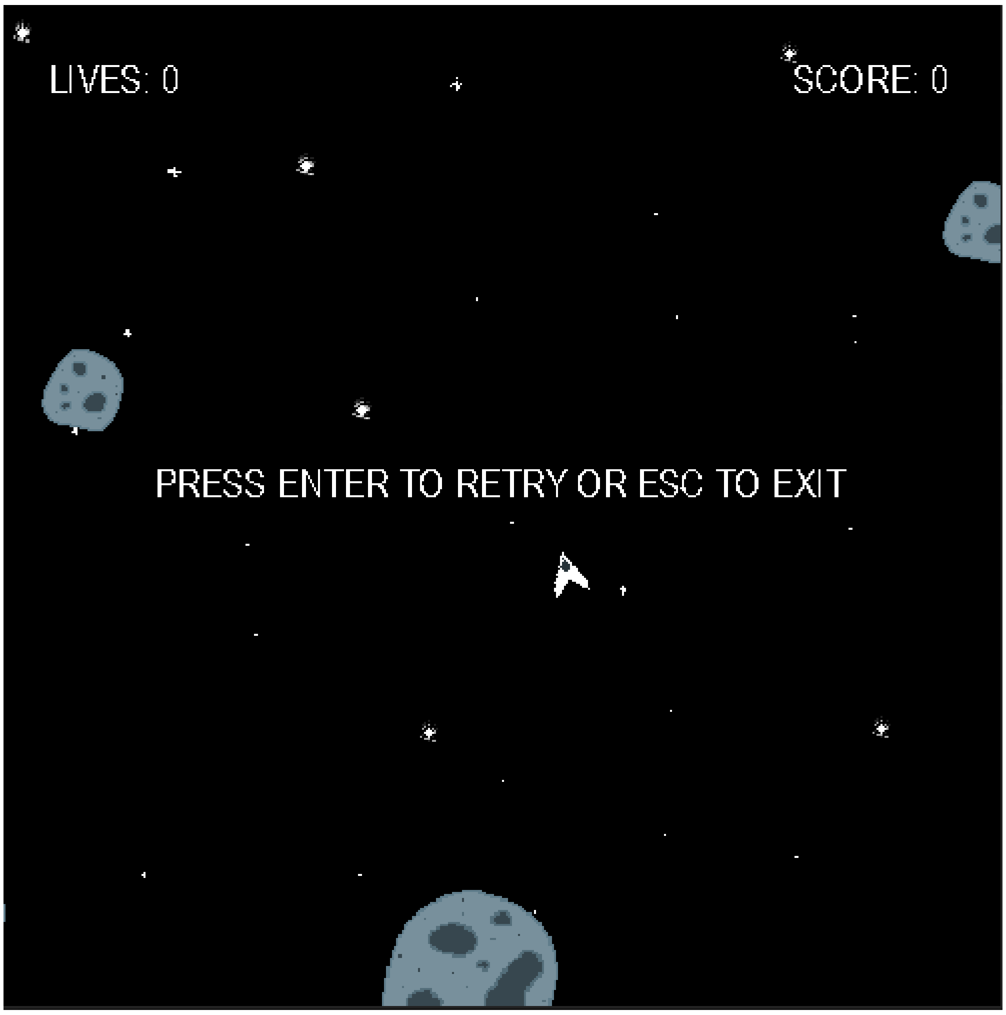
5. Gameplay



The gameplay is very simple. Players can move forwards and backwards while also being able to turn left and right a full 360 degrees. Asteroids will spawn randomly and the player will have to survive as long as they can to keep playing.

For every asteroid the player breaks, the player gets a score added to their total score. 250 for big asteroids, 500 for medium asteroids, and 1000 for small asteroids.

The player is given 3 lives and they will lose a life if they get struck by an asteroid. Lose 3 lives and the game is over.



When the game is over, the screen will be frozen and players are given a choice to either retry or quit the game.