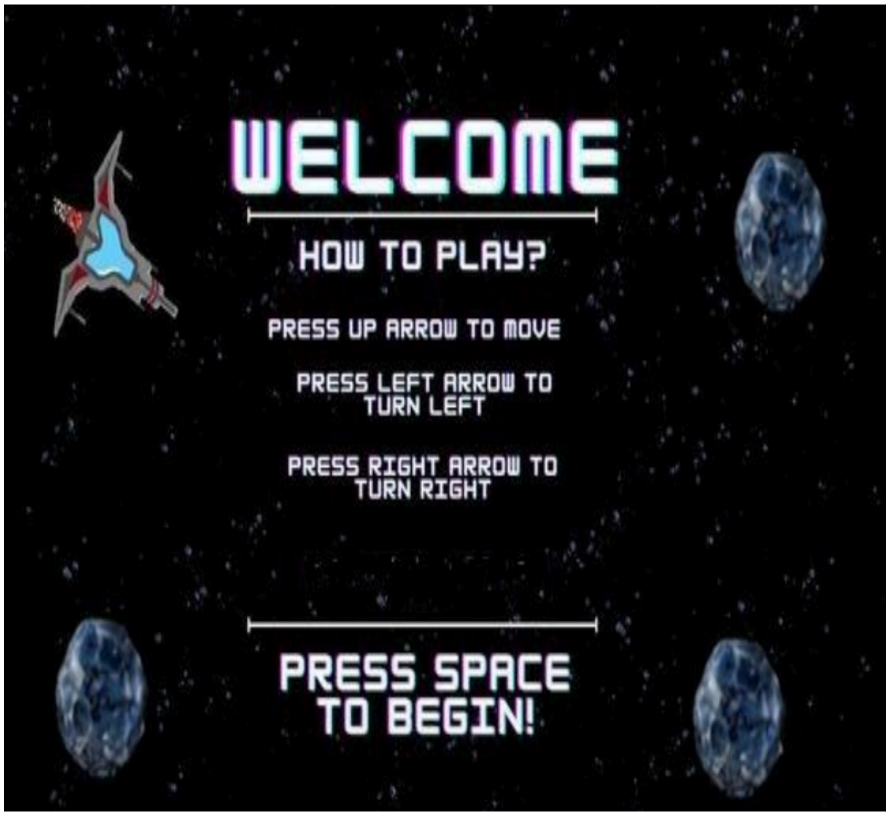
Introduction:
This report registers the development of our 2D game called 'Asteroid Attack' using simplegui library.The report also displays the understanding of key concepts learnt through the course.It also demonstrates the execution of various concepts including collision detection , GUI and many more.

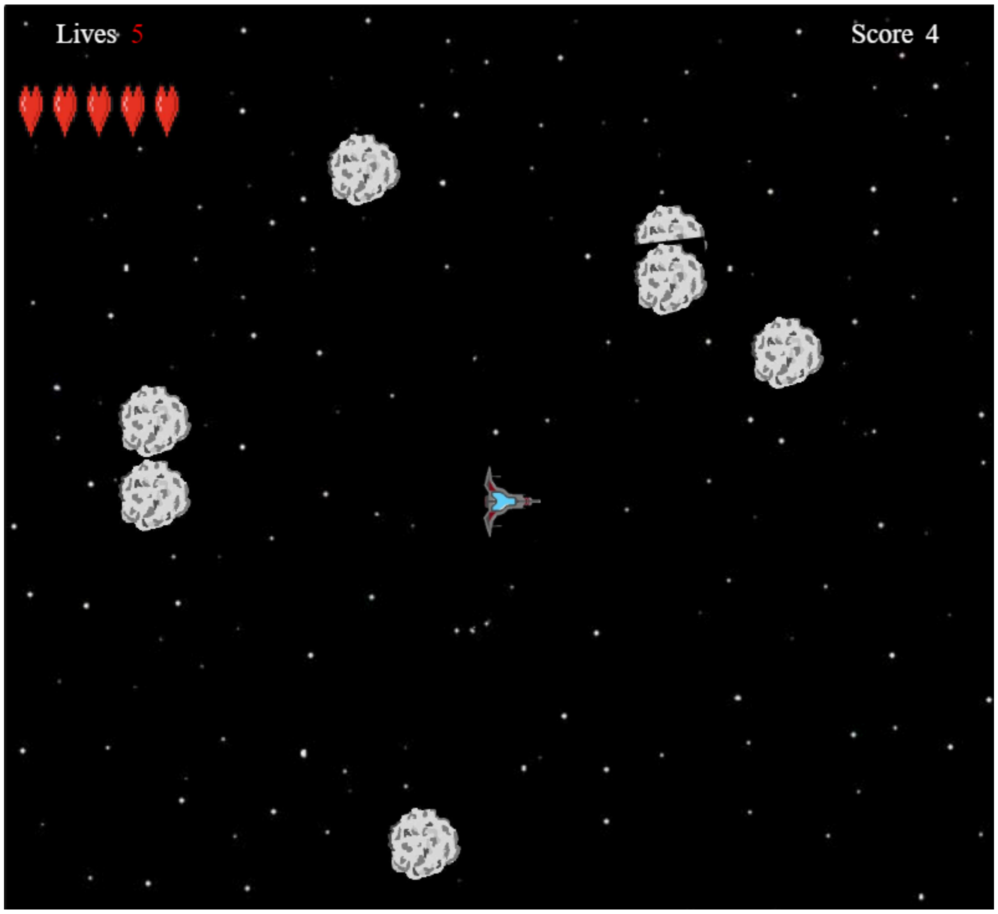
Game Description:
The story of the game 'Asteroid Attack' is within the name itself.There is no such storyline however the objective is to survive as long as possible and prevent the asteroids hitting your spaceship.You as a player must control the spaceship to dodge the moving asteroids if you fail to do so,will make you lose 1 life.Player starts with 5 lives so use them wisely! Player also needs to stay alert as staying stationary won't help as the asteroids will be coming from random positions.

Player can enter and leave from each and every side of the map allowing them to dodge the asteroids although user shouldn't rely on just escaping from the sides as it may lead them to hit an on-coming asteroid.The score is calculated by the amount of time you survive each time user passes a checkpoint they are alerted and checkpoints are incremented by 50.

User Manual:
To run the game,you must firstly open your preferred browser and open https://py3.codeskulptor.org/ .
Once codeskulptor is opened, copy and paste the game code in codeskulptor or if provided open the direct link to the game.Once the code is loaded,just press 'Run' to start the game and a game window should show up on your screen.

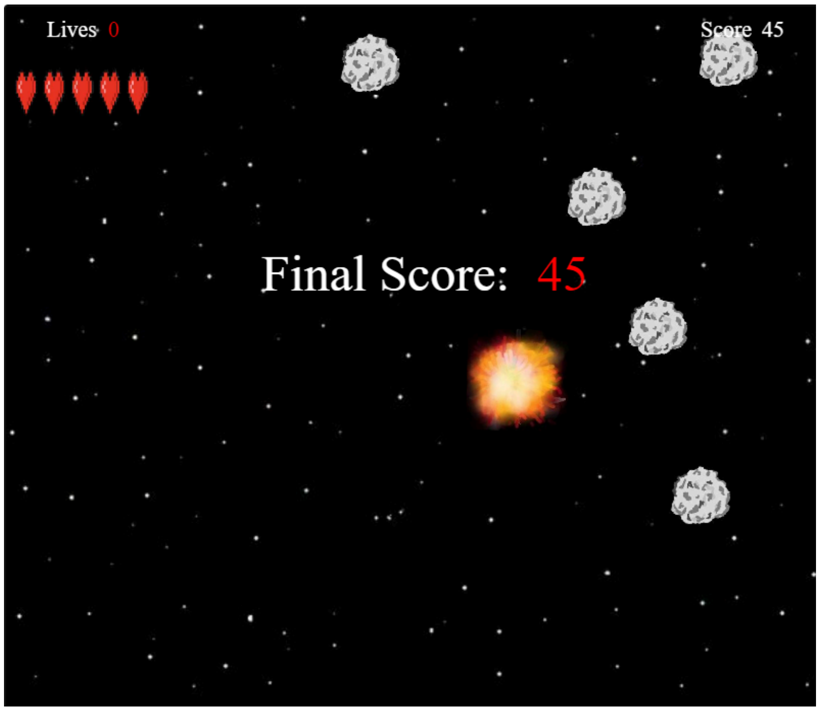


User is presented with a welcome screen as shown above which gives all the instructions to the game as well as the instructions to ‘press’ to begin the game. When you enter the welcome screen a sound effect is played to make you feel engaged with the game.

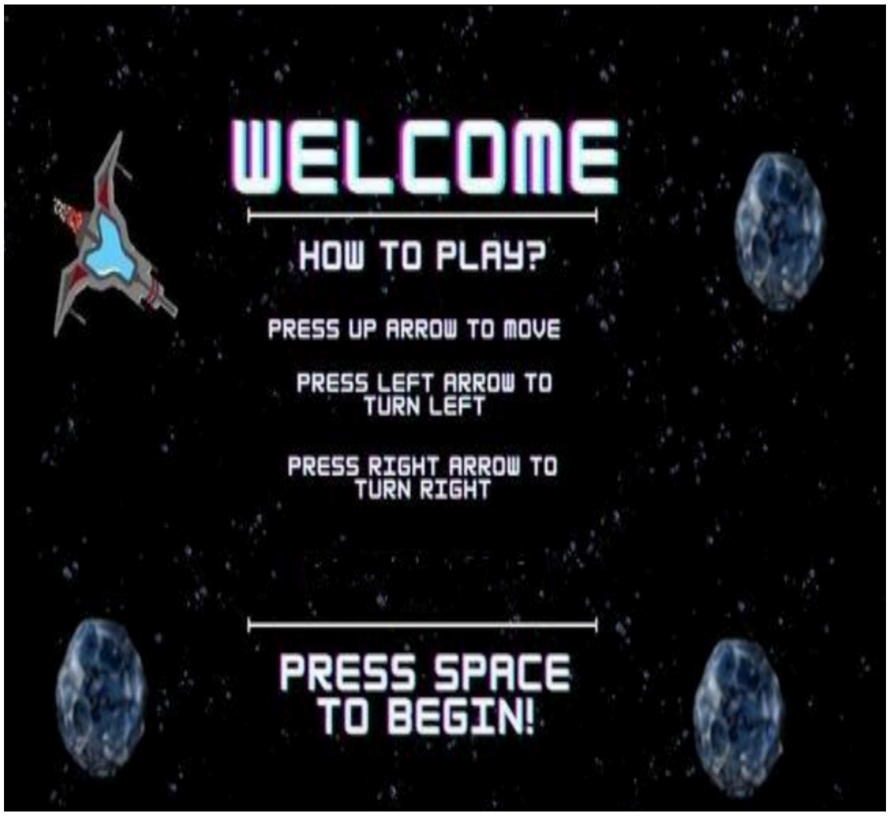


The second image is of when you start the game a background music is played when the game is running to keep the user thrilled . The spaceship is what you're controlling and you're surrounded by asteroids that are moving all over the map.At the top left corner you can see how many lives you have remaining and a graphical representation of the lives to help the user understand it visually.
You lose a life every time you hit an asteroid.On the top right corner you can see a score counter which keeps count of your score so the player knows what the score is.

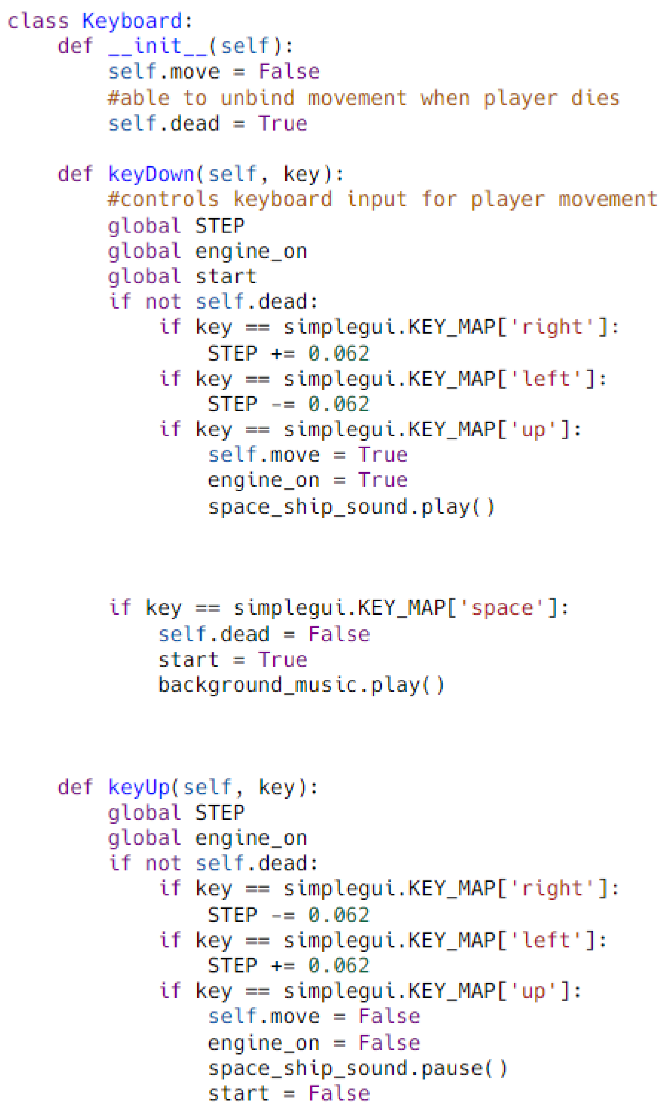
To start the game the you must press space bar.To control your player, you must use arrow keys up,left and right to control the spaceship.If user runs out of lives,a final score message is displayed and you're taken back to the welcome screen.



As previously mentioned above a final score is displayed and the spaceship is destroyed.The user will then see the same welcome screen.

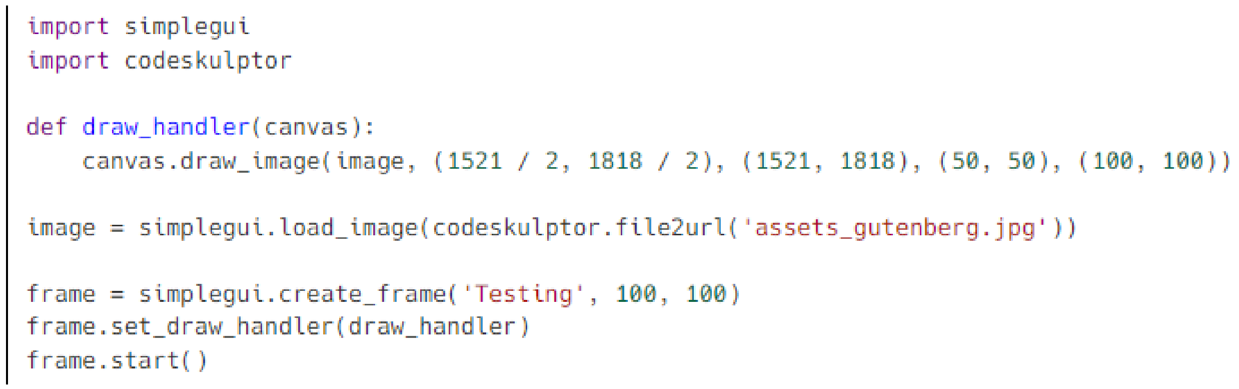


Key Concepts:
SimpleGUI as represented on CodeSkulptor is a library used for building interactive interfaces and games.It allows users to easily build 2D games with the help of SimpleGUI library.There numerous modules in the library which can be used to create an interface and games.For example,in the graphics module there consists several other modules such as images,sound,timers,canvas and frame which allows you to create a interactive program. The frame module for example is a container for the controls and allows you to create canvas.Another example is the images module which allows images to be loaded and drawn in the canvas.It also contains a sound module to load sounds into the program making it more interactive.Finally constant module is one of the main modules when it comes to creating interactive games as it allows you to create keyboard event handlers which is important in our game a code snippet of it is shown below.The image is the code which controls the main sprite the spaceship that the player controls.



```
class Keyboard:
    def __init__(self):
        self.move = False
        #able to unbind movement when player dies
        self.dead = True

    def keyDown(self, key):
        #controls keyboard input for player movement
        global STEP
        global engine_on
        global start
        if not self.dead:
            if key == simplegui.KEY_MAP['right']:
                STEP += 0.062
            if key == simplegui.KEY_MAP['left']:
                STEP -= 0.062
            if key == simplegui.KEY_MAP['up']:
                self.move = True
                engine_on = True
                space_ship_sound.play()



        if key == simplegui.KEY_MAP['space']:
            self.dead = False
            start = True
            background_music.play()



    def keyUp(self, key):
        global STEP
        global engine_on
        if not self.dead:
            if key == simplegui.KEY_MAP['right']:
                STEP -= 0.062
            if key == simplegui.KEY_MAP['left']:
                STEP += 0.062
            if key == simplegui.KEY_MAP['up']:
                self.move = False
                engine_on = False
                space_ship_sound.pause()
                start = False
```

To save images for the game I had to use https://www.pinterest.co.uk/ to save images.Once the pin was created I simply copied the image link address and used it to load my images to the game.Initially I didn't know how to load images so I used CodeSkulptor 3 documentation:https://py3.codeskulptor.org/docs.html#tabs-Python to learn how to do it.An example was given in the documentation and a code snippet is given below I used this for helping me understand how to load images.

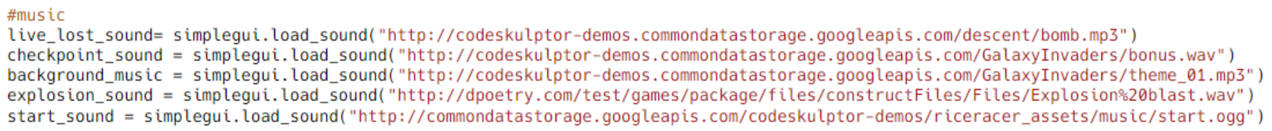


```
import simplegui
import codeskulptor

def draw_handler(canvas):
    canvas.draw_image(image, (1521 / 2, 1818 / 2), (1521, 1818), (50, 50), (100, 100))

image = simplegui.load_image(codeskulptor.file2url('assets_gutenberg.jpg'))

frame = simplegui.create_frame('Testing', 100, 100)
frame.set_draw_handler(draw_handler)
frame.start()
```

I also didn't understand how to get the width and height of my images so I used this documentation to help me.Below is a code snippet I used in my game to help me find width and height.

`image.get_width()` `image.get_height()`

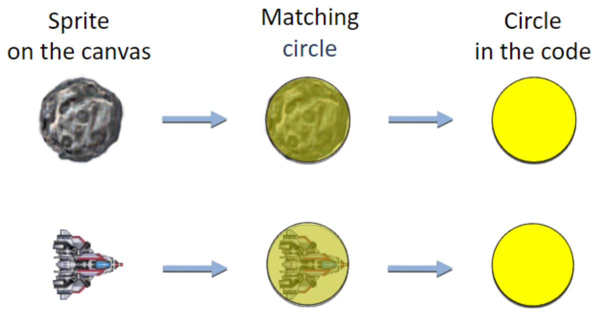
Similarly I used sound module information provided in the documentation to load in sounds.I learned how to load,play and pause the sound.However the main difficulty was finding free to use sound files.After several attempts to find a reliable website where I could borrow sound files,I found this website: https://simpleguics2pygame.readthedocs.io/en/latest/_static/links/snd_links.html it contained all the sound files I required for the game.Below is a example of how I used sound module in my game.



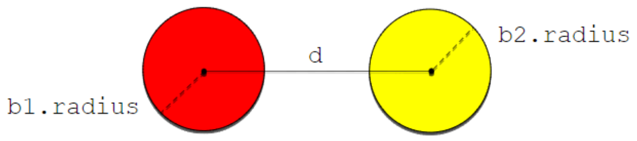
```
#music
live_lost_sound= simplegui.load_sound("http://codeskulptor-demos.commondatastorage.googleapis.com/descent/bomb.mp3")
checkpoint_sound = simplegui.load_sound("http://codeskulptor-demos.commondatastorage.googleapis.com/GalaxyInvaders/bonus.wav")
background_music = simplegui.load_sound("http://codeskulptor-demos.commondatastorage.googleapis.com/GalaxyInvaders/theme_01.mp3")
explosion_sound = simplegui.load_sound("http://dpoetry.com/test/games/package/files/constructFiles/Files/Explosion%20blast.wav")
start_sound = simplegui.load_sound("http://commondatastorage.googleapis.com/codeskulptor-demos/riceracer_assets/music/start.ogg")
```

```
background_music.play()
```

Another key concept was collision detections as it was a key part in our game.I had to do separate online research to further improve my understanding on this topic.I found ideas to help me with this from slides given to us by our teachers.The image below shows a basic graphical representation of how it can be achieved.The basic idea was to use circles to approximate the complex shape and code the circle instead of coding the complex shape saving a lot of time.Even though we didn't make the objects bounce of each other when collision was detected we managed to understand how to detect collision using ideas from given slides.



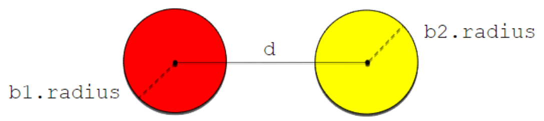
Take `b1` and `b2` objects of class `Ball`.



The distance between the centres:

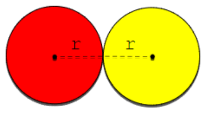
```
d = b1.pos.copy().subtract(b2.pos).length()
```

The distance between the centres is `d`



Collision if

```
d <= b1.radius + b2.radius
```



A moment of reflection:

Reflecting back on my work, if I had to build up everything from scratch, I wouldn’t make major changes myself. Due to the catastrophic and unfortunate pandemic, our development process suffered a small hindrance as one of our group mates suffered from COVID-19 and had to self-isolate for two weeks which restricted for all of us to code together.We caught up with work using teams to arrange online meetings and working together as a team.I would’ve liked to spend more time with each team member so we I could understand them better allowing better bonding. However I was very pleased with how most of us showed up during practical sessions and the outcome of that has been amazing.

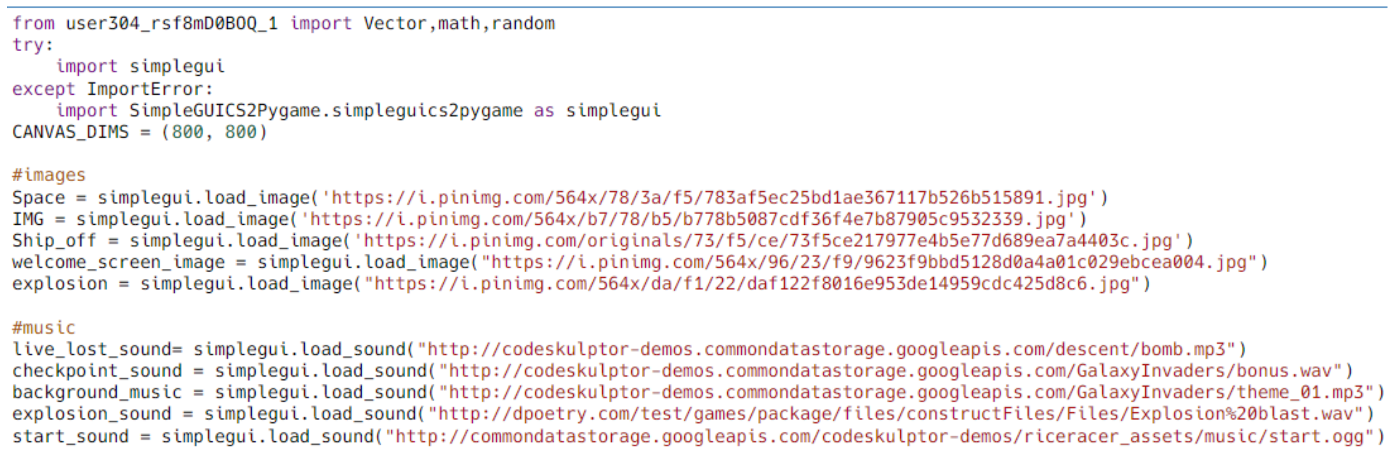
Even whilst my team member was recovering from COVID-19 she didn’t leave all the burden on us. Instead she helped us as much as she possibly could.Me and the others also put in extra effort even with limited time and resources.At first all of us lacked the commitment to keep going as there was a big task ahead however after we reached each milestone one by one and ticked of the specification requirements for the game we all got sudden motivation to complete this project.

Due to lack of time we couldn't add extra features that could’ve made the game more entertaining to play although I'm grateful we managed to complete all the mandatory features on time.The milestone sheets and worked example codes given by our teachers helped us a lot with how and what we wanted to make.In the initial stages having to plan out what game we wanted to make was quite challenging having played a lot of 2D games myself helped my team come up with a game idea.Moving off the initial planning stage allowed us to save more time for developing the actual game.

The hardest part about writing the programme for me was figuring out how to come up with a collision detection method to interact with the main player sprite and the asteroids.I had basic knowledge about how it could be done but later after using my teammates help and further taking inspiration from other projects I became more familiar with what the logic was behind.Another problem which I found tackling was switching from welcome screen to main game screen although after further reading the documentation provided by CodeSkulptor 3 I understood it well and executed it in the game.The game ended up how we expected although at start of testing process we discover many bugs and errors which were difficult to catch however after rechecking the code me and my team swiftly picked out the errors and got the game running smoothly and error free after several tests done.

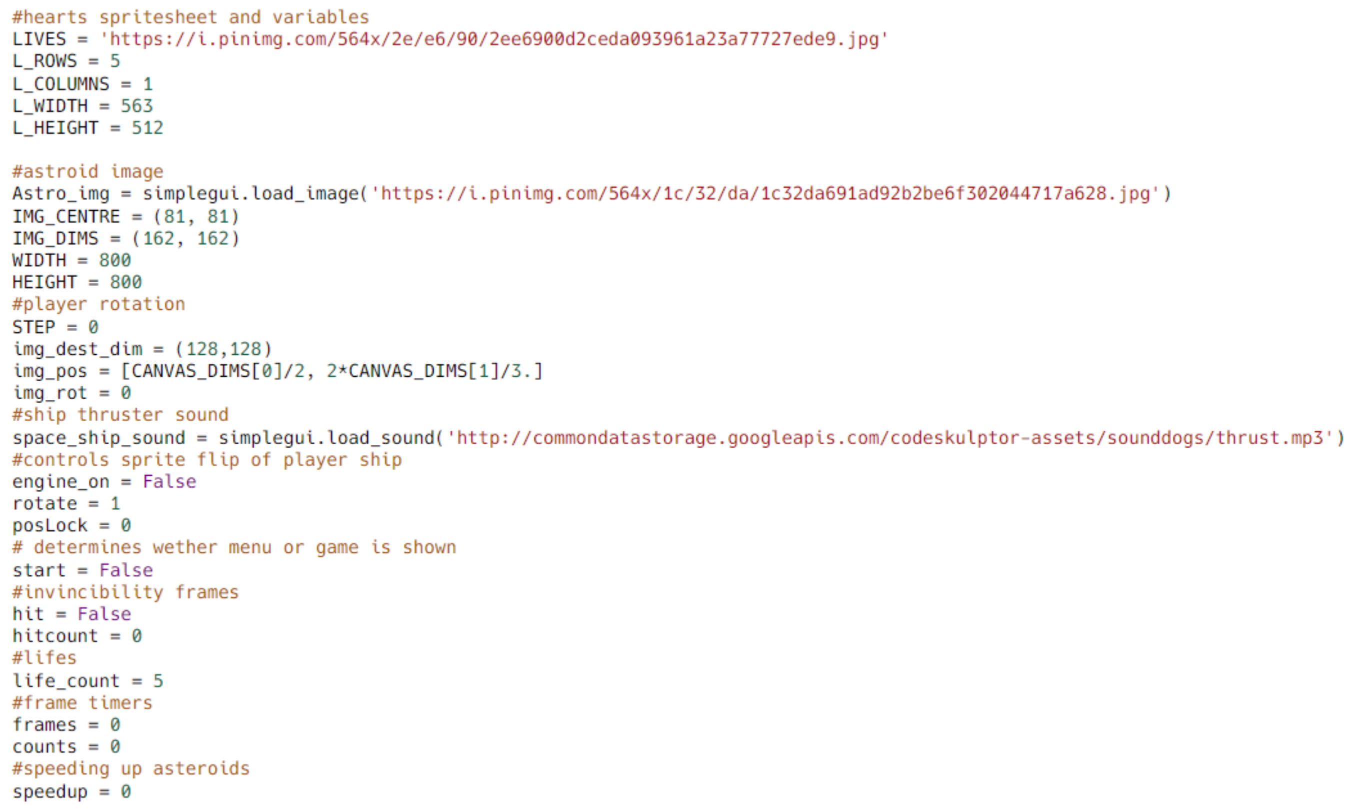
Code Description:

To start off our code begins with importing the required modules into the program.We then load multiple images and sounds for background,welcome screen and sprites to be used for our user interface,scoring system and other required parts of the code.



```python
from user304_rsf8mD0BOQ_1 import Vector,math,random
try:
    import simplegui
except ImportError:
    import SimpleGUICS2Pygame.simpleguics2pygame as simplegui
CANVAS_DIMS = (800, 800)

#images
Space = simplegui.load_image('https://i.pinimg.com/564x/78/3a/f5/783af5ec25bd1ae367117b526b515891.jpg')
IMG = simplegui.load_image('https://i.pinimg.com/564x/b7/78/b5/b778b5087cdf36f4e7b87905c9532339.jpg')
Ship_off = simplegui.load_image('https://i.pinimg.com/originals/73/f5/ce/73f5ce217977e4b5e77d689ea7a4403c.jpg')
welcome_screen_image = simplegui.load_image("https://i.pinimg.com/564x/96/23/f9/9623f9bbd5128d0a4a01c029ebcea004.jpg")
explosion = simplegui.load_image("https://i.pinimg.com/564x/da/f1/22/daf122f8016e953de14959cdc425d8c6.jpg")

#music
live_lost_sound= simplegui.load_sound("http://codeskulptor-demos.commondatastorage.googleapis.com/descent/bomb.mp3")
checkpoint_sound = simplegui.load_sound("http://codeskulptor-demos.commondatastorage.googleapis.com/GalaxyInvaders/bonus.wav")
background_music = simplegui.load_sound("http://codeskulptor-demos.commondatastorage.googleapis.com/GalaxyInvaders/theme_01.mp3")
explosion_sound = simplegui.load_sound("http://dpoetry.com/test/games/package/files/constructFiles/Files/Explosion%20blast.wav")
start_sound = simplegui.load_sound("http://commondatastorage.googleapis.com/codeskulptor-demos/riceracer_assets/music/start.ogg")
```

We then defined all the variables which are used for our heart sprite,asteroid sprite,player rotation,collision detection and other boolean variables used for controlling the player ship.

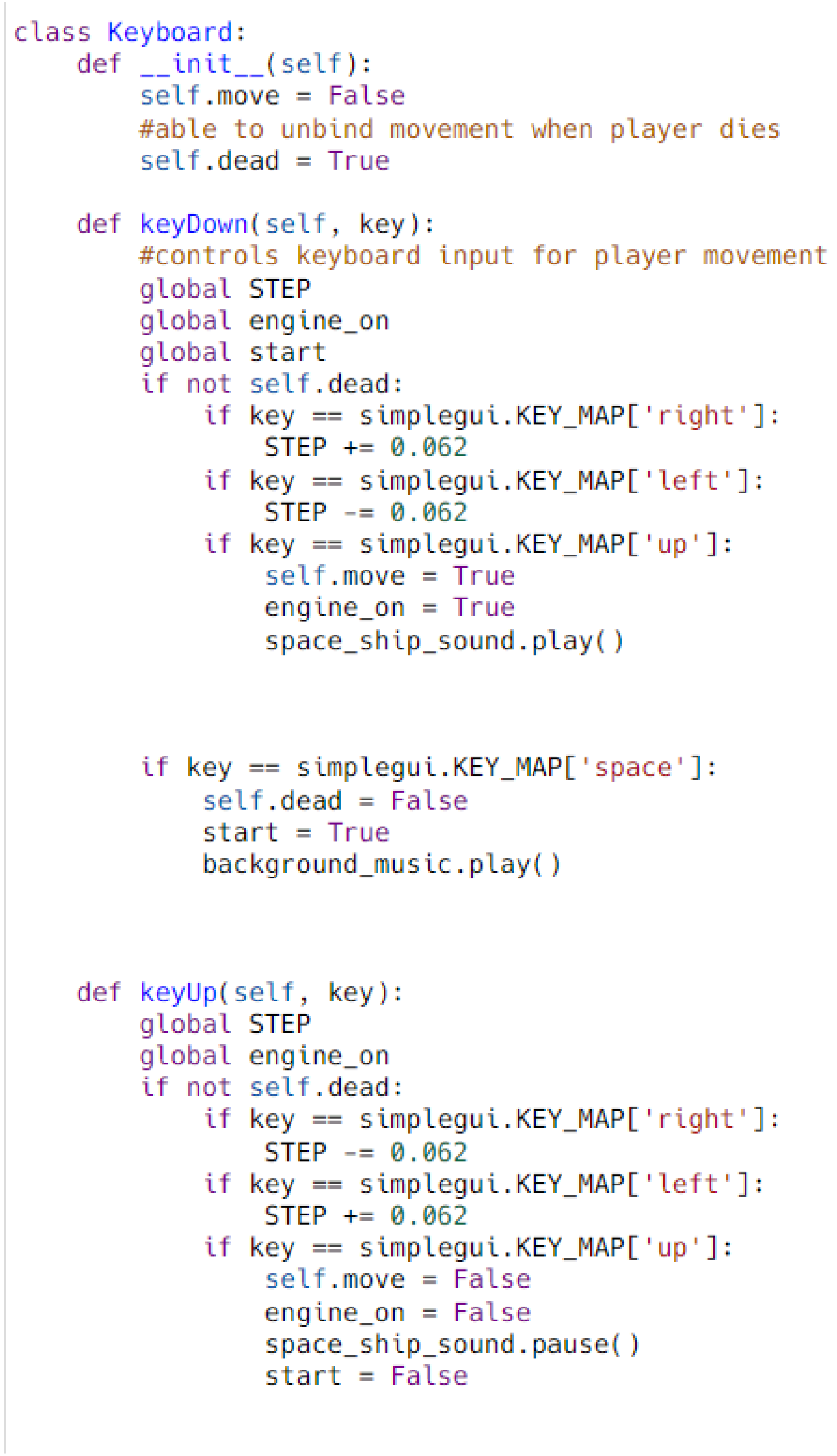


```
#hearts spritesheet and variables
LIVES = 'https://i.pinimg.com/564x/2e/e6/90/2ee6900d2ceda093961a23a77727ede9.jpg'
L_ROWS = 5
L_COLUMNS = 1
L_WIDTH = 563
L_HEIGHT = 512

#astroid image
Astro_img = simplegui.load_image('https://i.pinimg.com/564x/1c/32/da/1c32da691ad92b2be6f302044717a628.jpg')
IMG_CENTRE = (81, 81)
IMG_DIMS = (162, 162)
WIDTH = 800
HEIGHT = 800
#player rotation
STEP = 0
img_dest_dim = (128,128)
img_pos = [CANVAS_DIMS[0]/2, 2*CANVAS_DIMS[1]/3.]
img_rot = 0
#ship thruster sound
space_ship_sound = simplegui.load_sound('http://commondatastorage.googleapis.com/codeskulptor-assets/sounddogs/thrust.mp3')
#controls sprite flip of player ship
engine_on = False
rotate = 1
posLock = 0
# determines wether menu or game is shown
start = False
#invincibility frames
hit = False
hitcount = 0
#lifes
life_count = 5
#frame timers
frames = 0
counts = 0
#speeding up asteroids
speedup = 0
```

Leaving the variable initialization part we move on the ship class.As you can see in the code snippet below the self is used to represent instances of the class allowing us to access attributes and methods of the classes.The __init__ function passes two other parameters pos and radius with the value 10 stored inside.The draw function draws the ship on top the the ball so instead of the ball the ship is seen by the user.The global variable img_rot will be used to control player sprite rotation.The update function updates the ship movement by adding the position of the ship to the vector then multiplying the vector by 0.85.The offset function is used for collision,it returns point by using get_p().The damage function is used as a test case to check whether the program is working as expected.

```
class Ship:
    def __init__(self, pos, radius=10):
        self.pos = pos
        self.vel = Vector()
        self.radius = max(radius, 10)
        self.colour = 'black'
        self.dead = False
        self.reset = False

    def draw(self, canvas):
        #drawing circle object and spaceship sprite ontop
        global img_rot
        img_rot += STEP
        canvas.draw_circle(self.pos.get_p(), self.radius, 1, self.colour, self.colour)
        if self.dead == True:
            canvas.draw_image(explosion, IMG_CENTRE, IMG_DIMS, self.pos.get_p(), (100,100), img_rot)
        elif engine_on == False:
            canvas.draw_image(IMG, IMG_CENTRE, IMG_DIMS, self.pos.get_p(), (60,60), img_rot)
        else:
            canvas.draw_image(Ship_off, IMG_CENTRE, IMG_DIMS, self.pos.get_p(), (60,60), img_rot)

    def update(self):
        #ship movement
        self.pos.add(self.vel)
        self.vel.multiply(0.85)

    def offset(self):
        #player hitbox for collision
        return int(self.pos.get_p()[0]) + self.radius

    #test case
    def damage(self):
        print("hit")

    def loss(self, int):
        #registers player as dead after delay
        if int == 1:
            self.dead = True



class Astroid:
    def __init__(self, pos, radius, border, color,):
        self.pos = pos
        self.radius = radius
        self.border = border
        self.colour = color
        self.edge = radius - border
        self.vel = Vector()

    def draw(self, canvas):
        #draws circle object with asteroid sprite ontop
        global rotate
        rotate +=.01
        canvas.draw_circle(self.pos.get_p(),
                          self.radius,
                          2*self.border + 1,
                          self.colour)
        canvas.draw_circle(self.pos.get_p(), 28, 1, self.colour, self.colour)
        canvas.draw_image(Astro_img, (114/2,111/2), (114,111), self.pos.get_p(), (60,60), rotate)

    def update(self):
        #astroid movement
        self.pos.add(self.vel)
```

Moving onto the Asteroid class as shown in the image above we initialize the class passing in required parameters inside it and then call them in the fields of the class using self keyword.The draw function is then used to draw the asteroid on top of the circle and the global variable rotate is given value 0.1 and will then increment rotate by +0.1.As mentioned in key concepts we implemented the same idea for asteroids and player space ship two draw out two circles.



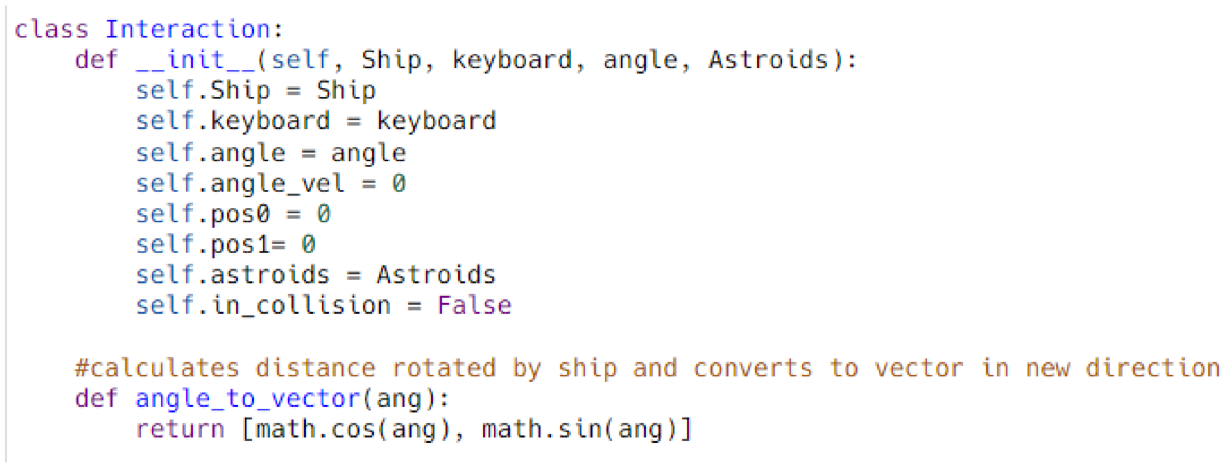
```python
class Keyboard:
    def __init__(self):
        self.move = False
        #able to unbind movement when player dies
        self.dead = True

    def keyDown(self, key):
        #controls keyboard input for player movement
        global STEP
        global engine_on
        global start
        if not self.dead:
            if key == simplegui.KEY_MAP['right']:
                STEP += 0.062
            if key == simplegui.KEY_MAP['left']:
                STEP -= 0.062
            if key == simplegui.KEY_MAP['up']:
                self.move = True
                engine_on = True
                space_ship_sound.play()



        if key == simplegui.KEY_MAP['space']:
            self.dead = False
            start = True
            background_music.play()



    def keyUp(self, key):
        global STEP
        global engine_on
        if not self.dead:
            if key == simplegui.KEY_MAP['right']:
                STEP -= 0.062
            if key == simplegui.KEY_MAP['left']:
                STEP += 0.062
            if key == simplegui.KEY_MAP['up']:
                self.move = False
                engine_on = False
                space_ship_sound.pause()
                start = False
```

The keyboard class controls the player movement.It starts off by initializing the class and setting self.move to False and self.dead to true.The keyDown and keyUp methods are very important as it controls player movements and plays a part in the games control flow.When the key is down the actions are triggered by the global variables STEP,engine_on and start.If statements are used in this method to check if player is not dead and if it's true the if statements below the first if statement is triggered.If right key is pressed down player turns to the right and vice versa for left key.Finally when up key is pressed self.move and engine_on is set to true and player moves activating the thrust sound.

The space key is used to change background using if statement and setting self.dead to false and start as true.When space is pressed it indicates the game has begun and background music starts playing.
The keyUp function is similar to the keyDown function but it stops the spaceship from moving when key is up and player is no longer interacting with the key.

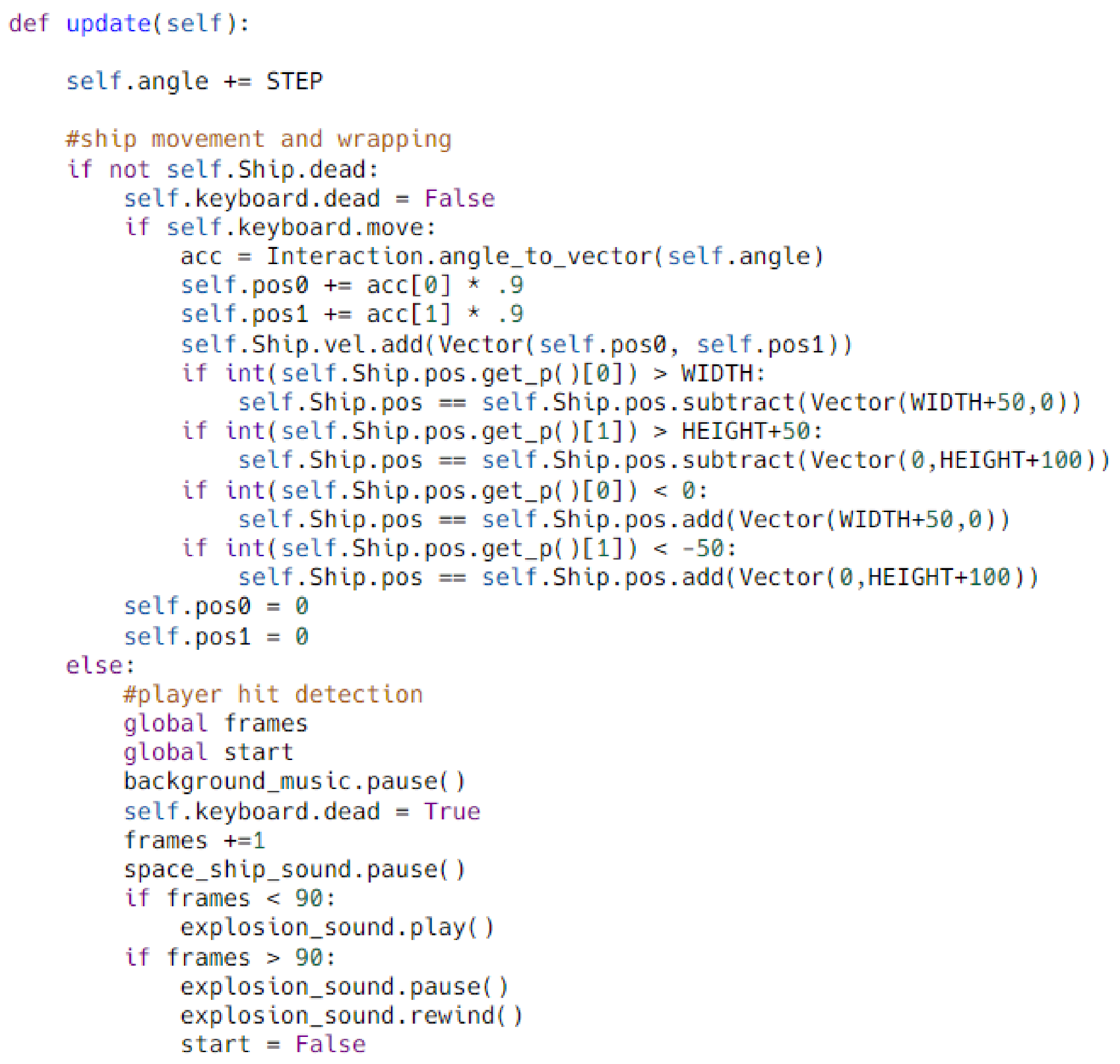
```
class Spritesheet:

    def __init__(self, imgurl, width, height, columns, rows):

        self.img = simplegui.load_image(imgurl)
        self.width = width
        self.height = height
        self.columns = columns
        self.rows = rows


        #frame dimensions

        self.frame_width = width
        self.frame_height = height / rows
        self.frame_centre_x = self.frame_width / 2
        self.frame_centre_y = self.frame_height / 2

        #frame indexing

        self.frame_index = [0,0]

    def update(self):
            #print("update")

            self.frame_index[0] = (self.frame_index[0] + 1) % self.columns
            if self.frame_index[0] == 0:
                self.frame_index[1] = (self.frame_index[1] + 1) % self.rows


    def draw(self, canvas):

        source_centre = (
            self.frame_width * self.frame_index[0] + self.frame_centre_x,
            self.frame_height * self.frame_index[1] + self.frame_centre_y
        )

        source_size = (self.frame_width, self.frame_height)
        dest_centre = (75,75)
        dest_size = (150,65)


        #drawing

        canvas.draw_image(self.img,source_centre, source_size,dest_centre, dest_size)
```

The spritesheet class above controls the hearts image that is displayed in the game.As done with previous classes,the constructor method initializes the class and passes the given parameters.These parameters are then stored in a field.The frame dimensions is calculated from lines 188 to 191.It calculates each row of the image instead of individual frame.The update method changes the frame by adding 1 to the index making the hearts disappear when player loses health.Finally the draw method draws out the image onto the canvas.



```
class Interaction:
    def __init__(self, Ship, keyboard, angle, Astroids):
        self.Ship = Ship
        self.keyboard = keyboard
        self.angle = angle
        self.angle_vel = 0
        self.pos0 = 0
        self.pos1= 0
        self.astroids = Astroids
        self.in_collision = False

    #calculates distance rotated by ship and converts to vector in new direction
    def angle_to_vector(ang):
        return [math.cos(ang), math.sin(ang)]
```

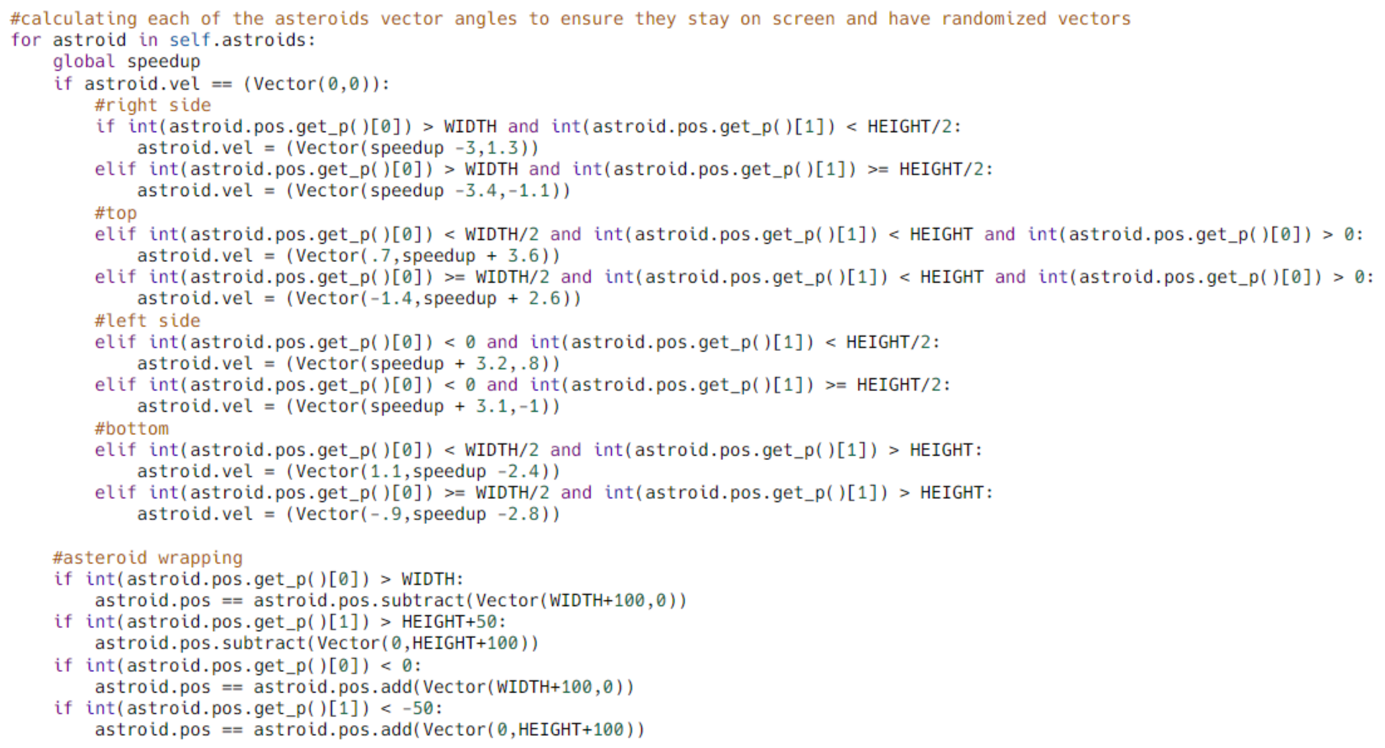
The interaction class controls the interaction between the sprites.The angle_to_vector function returns a direction by calculating the distance rotated by ship and converts it into a new direction.



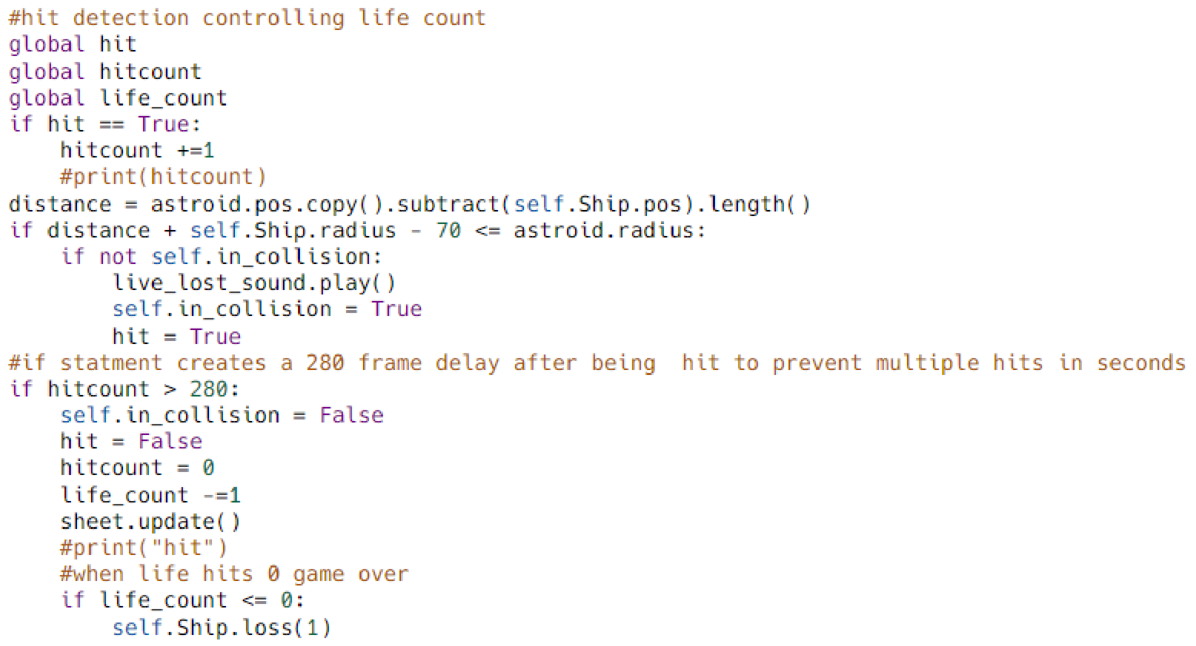
```
def update(self):

    self.angle += STEP

    #ship movement and wrapping
    if not self.Ship.dead:
        self.keyboard.dead = False
        if self.keyboard.move:
            acc = Interaction.angle_to_vector(self.angle)
            self.pos0 += acc[0] * .9
            self.pos1 += acc[1] * .9
            self.Ship.vel.add(Vector(self.pos0, self.pos1))
            if int(self.Ship.pos.get_p()[0]) > WIDTH:
                self.Ship.pos == self.Ship.pos.subtract(Vector(WIDTH+50,0))
            if int(self.Ship.pos.get_p()[1]) > HEIGHT+50:
                self.Ship.pos == self.Ship.pos.subtract(Vector(0,HEIGHT+100))
            if int(self.Ship.pos.get_p()[0]) < 0:
                self.Ship.pos == self.Ship.pos.add(Vector(WIDTH+50,0))
            if int(self.Ship.pos.get_p()[1]) < -50:
                self.Ship.pos == self.Ship.pos.add(Vector(0,HEIGHT+100))
        self.pos0 = 0
        self.pos1 = 0
    else:
        #player hit detection
        global frames
        global start
        background_music.pause()
        self.keyboard.dead = True
        frames +=1
        space_ship_sound.pause()
        if frames < 90:
            explosion_sound.play()
        if frames > 90:
            explosion_sound.pause()
            explosion_sound.rewind()
            start = False
```

The update method inside the interaction class controls the movement of the ship using if statements.It also allows the ship to wrap in and out the map by checking the changing the position of the ship when it’s at the edge of the map.

The else statement controls player hit detection and activates the explosion sound when it hits the asteroid.This is done using if statements.If frames less than 90 explosion audio is played and when its greater start is set to false suggesting the game has ended.



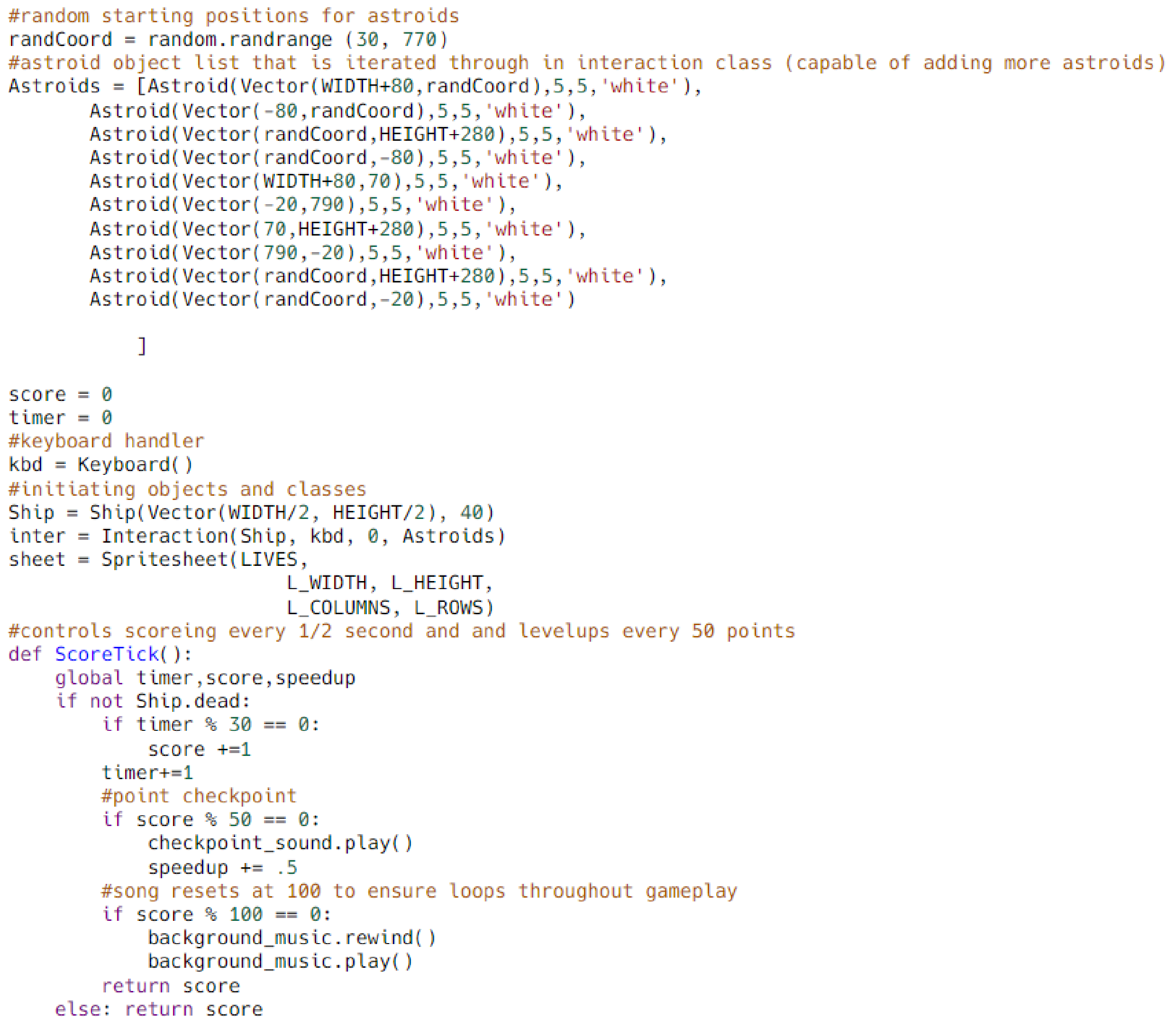
```
#calculating each of the asteroids vector angles to ensure they stay on screen and have randomized vectors
for astroid in self.astroids:
    global speedup
    if astroid.vel == (Vector(0,0)):
        #right side
        if int(astroid.pos.get_p()[0]) > WIDTH and int(astroid.pos.get_p()[1]) < HEIGHT/2:
            astroid.vel = (Vector(speedup -3,1.3))
        elif int(astroid.pos.get_p()[0]) > WIDTH and int(astroid.pos.get_p()[1]) >= HEIGHT/2:
            astroid.vel = (Vector(speedup -3.4,-1.1))
        #top
        elif int(astroid.pos.get_p()[0]) < WIDTH/2 and int(astroid.pos.get_p()[1]) < HEIGHT and int(astroid.pos.get_p()[0]) > 0:
            astroid.vel = (Vector(.7,speedup + 3.6))
        elif int(astroid.pos.get_p()[0]) >= WIDTH/2 and int(astroid.pos.get_p()[1]) < HEIGHT and int(astroid.pos.get_p()[0]) > 0:
            astroid.vel = (Vector(-1.4,speedup + 2.6))
        #left side
        elif int(astroid.pos.get_p()[0]) < 0 and int(astroid.pos.get_p()[1]) < HEIGHT/2:
            astroid.vel = (Vector(speedup + 3.2,.8))
        elif int(astroid.pos.get_p()[0]) < 0 and int(astroid.pos.get_p()[1]) >= HEIGHT/2:
            astroid.vel = (Vector(speedup + 3.1,-1))
        #bottom
        elif int(astroid.pos.get_p()[0]) < WIDTH/2 and int(astroid.pos.get_p()[1]) > HEIGHT:
            astroid.vel = (Vector(1.1,speedup -2.4))
        elif int(astroid.pos.get_p()[0]) >= WIDTH/2 and int(astroid.pos.get_p()[1]) > HEIGHT:
            astroid.vel = (Vector(-.9,speedup -2.8))

    #asteroid wrapping
    if int(astroid.pos.get_p()[0]) > WIDTH:
        astroid.pos == astroid.pos.subtract(Vector(WIDTH+100,0))
    if int(astroid.pos.get_p()[1]) > HEIGHT+50:
        astroid.pos.subtract(Vector(0,HEIGHT+100))
    if int(astroid.pos.get_p()[0]) < 0:
        astroid.pos == astroid.pos.add(Vector(WIDTH+100,0))
    if int(astroid.pos.get_p()[1]) < -50:
        astroid.pos == astroid.pos.add(Vector(0,HEIGHT+100))
```



```
#hit detection controlling life count
global hit
global hitcount
global life_count
if hit == True:
    hitcount +=1
    #print(hitcount)
distance = astroid.pos.copy().subtract(self.Ship.pos).length()
if distance + self.Ship.radius - 70 <= astroid.radius:
    if not self.in_collision:
        live_lost_sound.play()
        self.in_collision = True
        hit = True
#if statment creates a 280 frame delay after being  hit to prevent multiple hits in seconds
if hitcount > 280:
    self.in_collision = False
    hit = False
    hitcount = 0
    life_count -=1
    sheet.update()
    #print("hit")
    #when life hits 0 game over
    if life_count <= 0:
        self.Ship.loss(1)
```

The for loop inside the interaction class is used to iterate over a sequence of asteroids and controls how the asteroids spawn and attack the player from.The get_p() function allows asteroids to come from random vector points.It also contains the wrapping of asteroid which is executed in a similar way done with the spaceship sprite.

The hitcount of the player is stored inside this function and if hit is true hitcount is increased by +1 .The if statement on line 327 creates a 280 frame delay after being hit to prevent multiple hits being detected in a second.The random coordinates of asteroids are stored in the randCoord variable.



```
#random starting positions for astroids
randCoord = random.randrange (30, 770)
#astroid object list that is iterated through in interaction class (capable of adding more astroids)
Astroids = [Astroid(Vector(WIDTH+80,randCoord),5,5,'white'),
       Astroid(Vector(-80,randCoord),5,5,'white'),
       Astroid(Vector(randCoord,HEIGHT+280),5,5,'white'),
       Astroid(Vector(randCoord,-80),5,5,'white'),
       Astroid(Vector(WIDTH+80,70),5,5,'white'),
       Astroid(Vector(-20,790),5,5,'white'),
       Astroid(Vector(70,HEIGHT+280),5,5,'white'),
       Astroid(Vector(790,-20),5,5,'white'),
       Astroid(Vector(randCoord,HEIGHT+280),5,5,'white'),
       Astroid(Vector(randCoord,-20),5,5,'white')

           ]

score = 0
timer = 0
#keyboard handler
kbd = Keyboard()
#initiating objects and classes
Ship = Ship(Vector(WIDTH/2, HEIGHT/2), 40)
inter = Interaction(Ship, kbd, 0, Astroids)
sheet = Spritesheet(LIVES,
                        L_WIDTH, L_HEIGHT,
                        L_COLUMNS, L_ROWS)
#controls scoreing every 1/2 second and and levelups every 50 points
def ScoreTick():
    global timer,score,speedup
    if not Ship.dead:
        if timer % 30 == 0:
            score +=1
        timer+=1
        #point checkpoint
        if score % 50 == 0:
            checkpoint_sound.play()
            speedup += .5
        #song resets at 100 to ensure loops throughout gameplay
        if score % 100 == 0:
            background_music.rewind()
            background_music.play()
        return score
    else: return score
```

The image above stars with variables for storing score and time.The keyboard handler is called and other objects and classes are initialised.The scoreTick function controls the scoring system by using if and else statements Modulus(%) is used so if the timer%30=0 it increases score by +1 each time.To ensure user knowns when they reach a checkpoint another if statement is used to play checkpoint sound.As long as score % 50 = 0 it will continue playing the sound.We made sure at 100 the song resets so it can be played throughout the gameplay.

```python
#main draw method
def draw(canvas):
    #global reset variables
    global Astroids
    global life_count
    global STEP
    global counts
    global frames
    global score
    #in menu screen
    if start == False:
        Keyboard.move = False
        life_count = 5
        STEP = 0
        frames = 0
        score = 0
        Ship.dead = False
        width = welcome_screen_image.get_width()
        height = welcome_screen_image.get_height()
        canvas.draw_image(welcome_screen_image,(564/2,317/2),(564,317),[400,400],(800,800))
        background_music.rewind()
        space_ship_sound.pause()
        counts +=1
        space_ship_sound.pause()

        if counts < 120:
            start_sound.play()
        if counts > 180:
            start_sound.pause()
            start_sound.rewind()


    #in game
    else:
        counts = 0
        canvas.draw_image(Space,(564/2,564/2),(564,564),[400,400],(800,800))
        #canvas.draw_image(Astro_img,(114/2,111/2),(114,111),[400,400],(800,800))
        inter.update()
        Ship.update()
        Ship.draw(canvas)
        sheet.draw(canvas)
        for astro in Astroids:
            astro.update()
            astro.draw(canvas)
        #displays final score
        if life_count == 0:
                canvas.draw_text("Final Score:", [250, 280], 50, "White")
                canvas.draw_text(str(ScoreTick()), [520, 280], 50, "Red")
        #displays score and live count (possible small bug with life sprite sheet so number count as backup)
        #if player is hit after game spritesheet becomes desycned
        canvas.draw_text("Score", [680, 30], 22, "White")
        canvas.draw_text(str(ScoreTick()), [740, 30], 22, "White")
        canvas.draw_text("Lives", [40, 30], 22, "White")
        canvas.draw_text(str(life_count), [100, 30], 22, "Red")


frame = simplegui.create_frame('Interactions', WIDTH, HEIGHT)
frame.set_canvas_background('black')
frame.set_draw_handler(draw)
frame.set_keydown_handler(kbd.keyDown)
frame.set_keyup_handler(kbd.keyUp)
frame.start()
```

The main draw method controls the animation for switching between menu screen and game screen.As shown in the code above If statement is used so if start is false the draw image method is called to draw out the welcome screen.In game audios are all paused during this time to prevent mix up.The else statement contains the in game images which is drawn if start is True which is triggered when space is pressed.The else statement also contains final score message which is displayed when user dies.This is done using the draw_text method and checking if life_count=0.The draw_text also shows lives left and live score counter.Finally the frame is created with given title and is set with other handlers.

Conclusion:
Overall our group has successfully managed to execute all the key concepts our 2D game required to have the game completed.We have fulfilled all the necessary requirements to make this 2D game and shown great teamwork and resilience to complete this game.We have executed our game plan well and overachieved with the amount of time and resources given.Furthermore we have substantially improved our documentation of code and improved our coding skills.Improvement can be made on managing time for efficiently so we can integrate other additional features to escalate our game “Asteroids Attack”.Improving the collision detection system further by making the spaceship and asteroids collide will help make the game better.Features such as equipping spaceship with a weapon and adding harder levels to the game after player reaches new checkpoints can bring more excitement to the game.Finally improving on working with teammates closer to understand what everyone is comfortable doing can help spread the workload to everyone equally and hence improving load management which our group can improve on.