PROJECT REPORT

Project Specification

- Name : Word Box Juice
- Genre : Space shooters
- Mode : 2D Shoot ‘em up
- Scenario: Player control a spacecraft with the ability of moving and shooting. A random word is chosen from a text file, and the player must collect the specific number of letters which form the chosen word to obtain points. Other letters must be avoided or shot down. If the player gets hit, then the game is over.

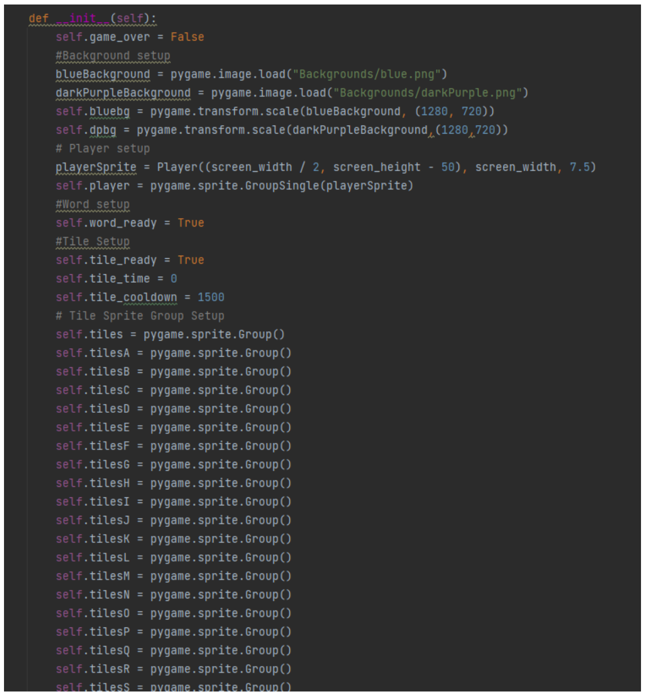
Solution Design

The project was designed with the objective of providing a fun and classic gamified way of learning a set of given words. It only has one classic endless mode.

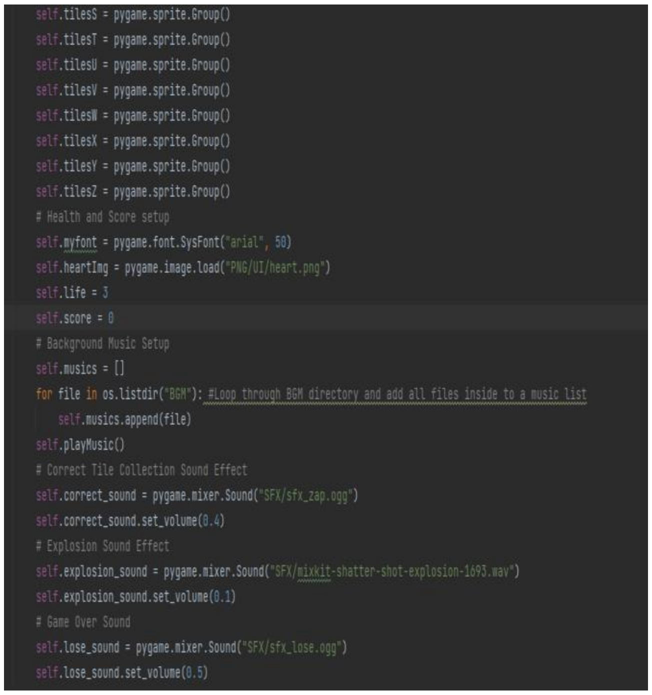
Discussion

- Main.py

Game class has the following functions:

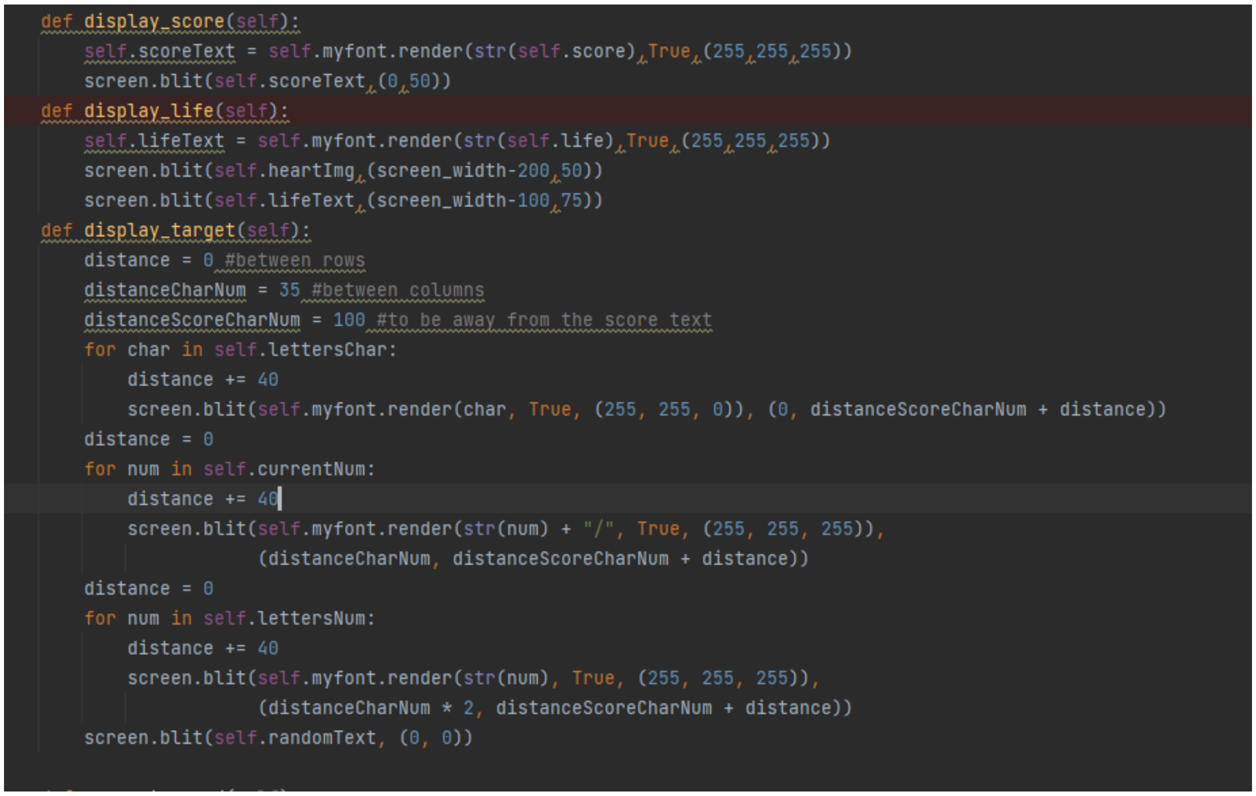


```python
def __init__(self):
    self.game_over = False
    #Background setup
    blueBackground = pygame.image.load("Backgrounds/blue.png")
    darkPurpleBackground = pygame.image.load("Backgrounds/darkPurple.png")
    self.bluebg = pygame.transform.scale(blueBackground, (1280, 720))
    self.dpbg = pygame.transform.scale(darkPurpleBackground,(1280,720))
    # Player setup
    playerSprite = Player((screen_width / 2, screen_height - 50), screen_width, 7.5)
    self.player = pygame.sprite.GroupSingle(playerSprite)
    #Word setup
    self.word_ready = True
    #Tile Setup
    self.tile_ready = True
    self.tile_time = 0
    self.tile_cooldown = 1500
    # Tile Sprite Group Setup
    self.tiles = pygame.sprite.Group()
    self.tilesA = pygame.sprite.Group()
    self.tilesB = pygame.sprite.Group()
    self.tilesC = pygame.sprite.Group()
    self.tilesD = pygame.sprite.Group()
    self.tilesE = pygame.sprite.Group()
    self.tilesF = pygame.sprite.Group()
    self.tilesG = pygame.sprite.Group()
    self.tilesH = pygame.sprite.Group()
    self.tilesI = pygame.sprite.Group()
    self.tilesJ = pygame.sprite.Group()
    self.tilesK = pygame.sprite.Group()
    self.tilesL = pygame.sprite.Group()
    self.tilesM = pygame.sprite.Group()
    self.tilesN = pygame.sprite.Group()
    self.tilesO = pygame.sprite.Group()
    self.tilesP = pygame.sprite.Group()
    self.tilesQ = pygame.sprite.Group()
    self.tilesR = pygame.sprite.Group()
    self.tilesS = pygame.sprite.Group()
```



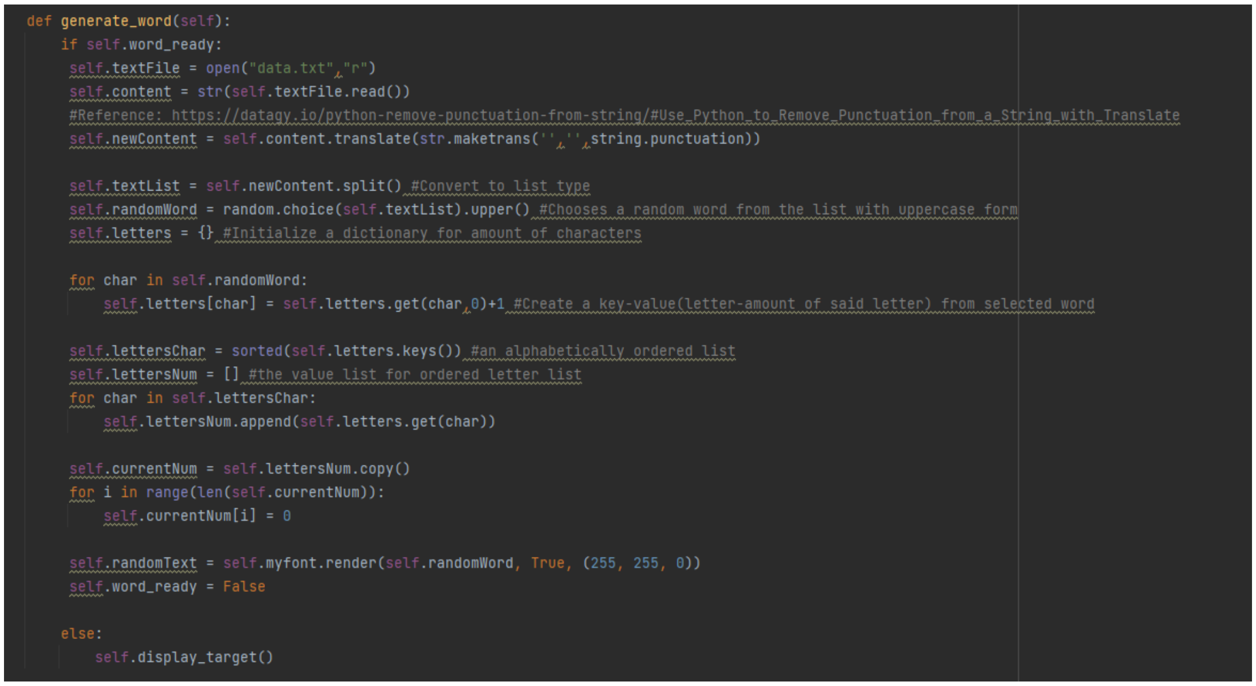
```python
self.tilesS = pygame.sprite.Group()
self.tilesT = pygame.sprite.Group()
self.tilesU = pygame.sprite.Group()
self.tilesV = pygame.sprite.Group()
self.tilesW = pygame.sprite.Group()
self.tilesX = pygame.sprite.Group()
self.tilesY = pygame.sprite.Group()
self.tilesZ = pygame.sprite.Group()
# Health and Score setup
self.myfont = pygame.font.SysFont("arial", 50)
self.heartImg = pygame.image.load("PNG/UI/heart.png")
self.life = 3
self.score = 0
# Background Music Setup
self.musics = []
for file in os.listdir("BGM"): #Loop through BGM directory and add all files inside to a music list
    self.musics.append(file)
self.playMusic()
# Correct Tile Collection Sound Effect
self.correct_sound = pygame.mixer.Sound("SFX/sfx_zap.ogg")
self.correct_sound.set_volume(0.4)
# Explosion Sound Effect
self.explosion_sound = pygame.mixer.Sound("SFX/mixkit-shatter-shot-explosion-1693.wav")
self.explosion_sound.set_volume(0.1)
# Game Over Sound
self.lose_sound = pygame.mixer.Sound("SFX/sfx_lose.ogg")
self.lose_sound.set_volume(0.5)
```

a. init(): assign values to setup the game's state,background,player character, word,tile,health,score,background music and sound effects.



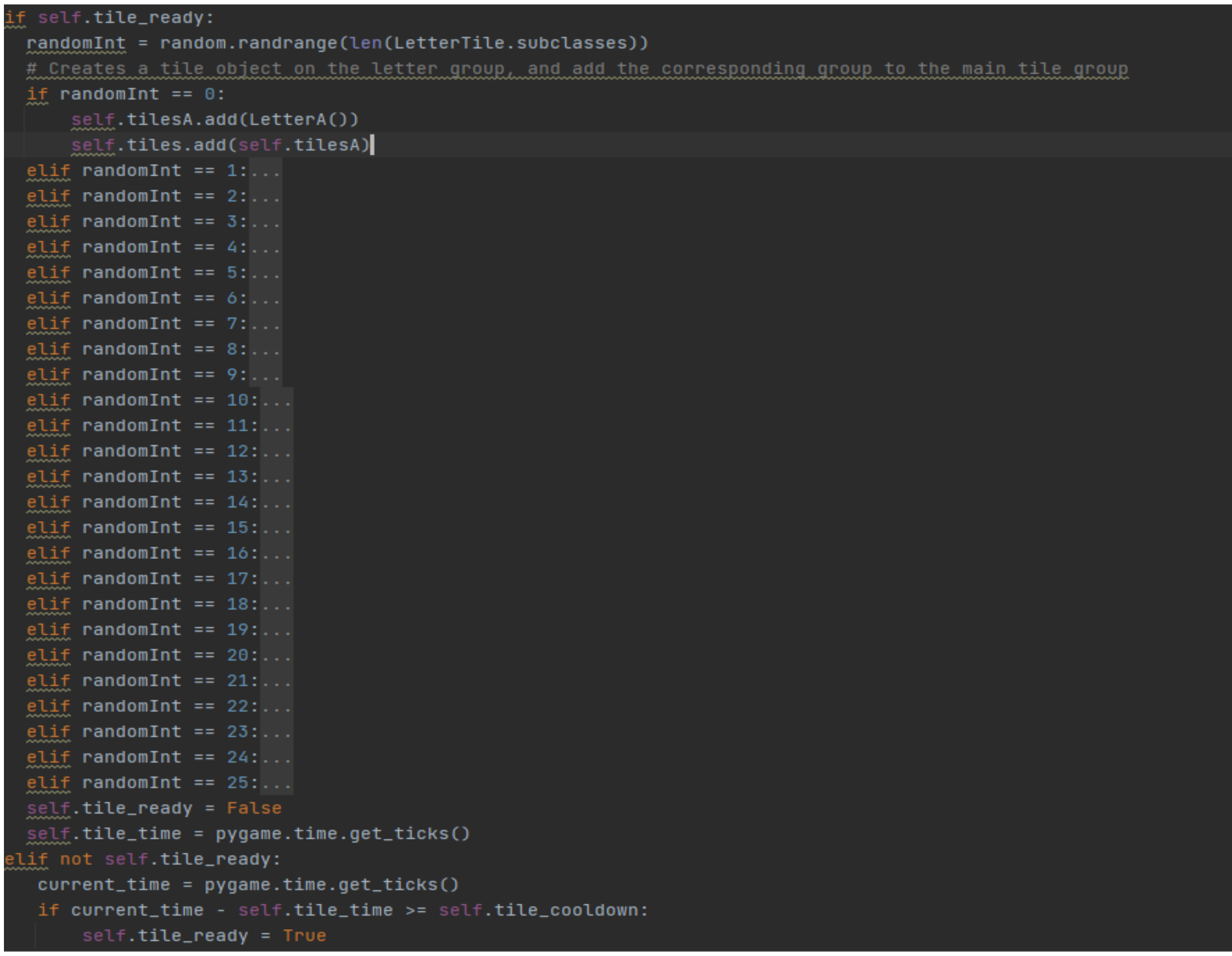
```python
def display_score(self):
    self.scoreText = self.myfont.render(str(self.score),True,(255,255,255))
    screen.blit(self.scoreText,(0,50))
def display_life(self):
    self.lifeText = self.myfont.render(str(self.life),True,(255,255,255))
    screen.blit(self.heartImg,(screen_width-200,50))
    screen.blit(self.lifeText,(screen_width-100,75))
def display_target(self):
    distance = 0 #between rows
    distanceCharNum = 35 #between columns
    distanceScoreCharNum = 100 #to be away from the score text
    for char in self.lettersChar:
        distance += 40
        screen.blit(self.myfont.render(char, True, (255, 255, 0)), (0, distanceScoreCharNum + distance))
    distance = 0
    for num in self.currentNum:
        distance += 40
        screen.blit(self.myfont.render(str(num) + "/", True, (255, 255, 255)),
                    (distanceCharNum, distanceScoreCharNum + distance))
    distance = 0
    for num in self.lettersNum:
        distance += 40
        screen.blit(self.myfont.render(str(num), True, (255, 255, 255)),
                    (distanceCharNum * 2, distanceScoreCharNum + distance))
    screen.blit(self.randomText, (0, 0))
```

b. display_score(): render the score as a text on the screen
c. display_life(): render the remaining life as a text on the screen
d. display_target(): render the required amount of characters to form the randomized word on the screen



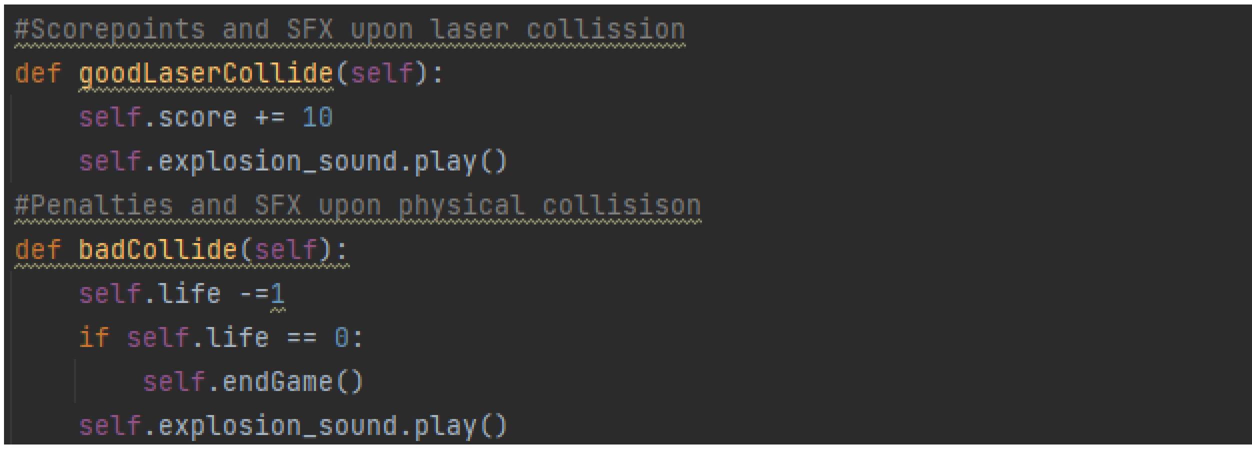
```python
def generate_word(self):
    if self.word_ready:
        self.textFile = open("data.txt","r")
        self.content = str(self.textFile.read())
        #Reference: https://datagy.io/python-remove-punctuation-from-string/#Use_Python_to_Remove_Punctuation_from_a_String_with_Translate
        self.newContent = self.content.translate(str.maketrans('','',string.punctuation))

        self.textList = self.newContent.split() #Convert to list type
        self.randomWord = random.choice(self.textList).upper() #Chooses a random word from the list with uppercase form
        self.letters = {} #Initialize a dictionary for amount of characters

        for char in self.randomWord:
            self.letters[char] = self.letters.get(char,0)+1 #Create a key-value(letter-amount of said letter) from selected word

        self.lettersChar = sorted(self.letters.keys()) #an alphabetically ordered list
        self.lettersNum = [] #the value list for ordered letter list
        for char in self.lettersChar:
            self.lettersNum.append(self.letters.get(char))

        self.currentNum = self.lettersNum.copy()
        for i in range(len(self.currentNum)):
            self.currentNum[i] = 0

        self.randomText = self.myfont.render(self.randomWord, True, (255, 255, 0))
        self.word_ready = False

    else:
        self.display_target()
```

e. generate_word(): get a random word from a read text file and create lists based on dictionary of the amount of character/letter(s) of the word. Calls the display function when not generating.



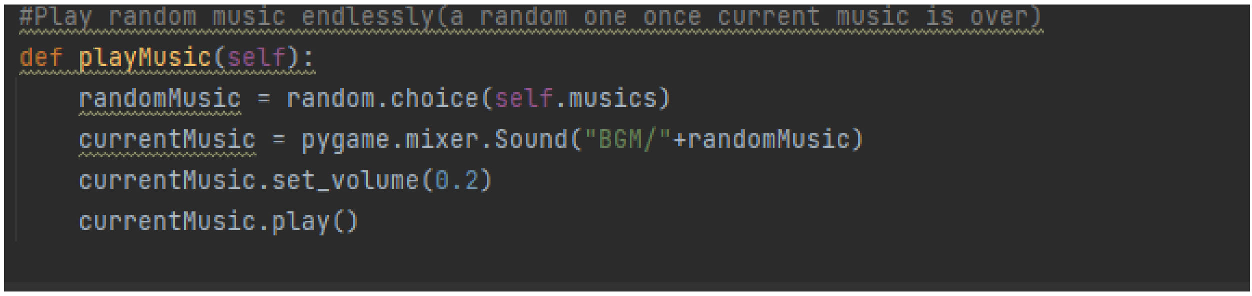
```
if self.tile_ready:
  randomInt = random.randrange(len(LetterTile.subclasses))
  # Creates a tile object on the letter group, and add the corresponding group to the main tile group
  if randomInt == 0:
      self.tilesA.add(LetterA())
      self.tiles.add(self.tilesA)
  elif randomInt == 1:...
  elif randomInt == 2:...
  elif randomInt == 3:...
  elif randomInt == 4:...
  elif randomInt == 5:...
  elif randomInt == 6:...
  elif randomInt == 7:...
  elif randomInt == 8:...
  elif randomInt == 9:...
  elif randomInt == 10:...
  elif randomInt == 11:...
  elif randomInt == 12:...
  elif randomInt == 13:...
  elif randomInt == 14:...
  elif randomInt == 15:...
  elif randomInt == 16:...
  elif randomInt == 17:...
  elif randomInt == 18:...
  elif randomInt == 19:...
  elif randomInt == 20:...
  elif randomInt == 21:...
  elif randomInt == 22:...
  elif randomInt == 23:...
  elif randomInt == 24:...
  elif randomInt == 25:...
  self.tile_ready = False
  self.tile_time = pygame.time.get_ticks()
elif not self.tile_ready:
   current_time = pygame.time.get_ticks()
   if current_time - self.tile_time >= self.tile_cooldown:
       self.tile_ready = True
```

f. generate_tile(): If boolean condition is met, with the range of the length of alphabets, randomly select a number and create a letterTile instance accordingly, which group is then added to the primary tiles group. It also checks whether the cooldown time has passed and if so, the boolean state will be True to allow the condition to generate a tile to be met.



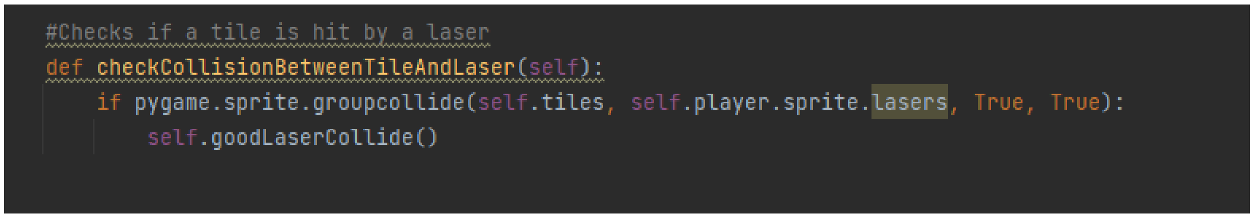
```
#Scorepoints and SFX upon laser collission
def goodLaserCollide(self):
    self.score += 10
    self.explosion_sound.play()
#Penalties and SFX upon physical collisison
def badCollide(self):
    self.life -=1
    if self.life == 0:
        self.endGame()
    self.explosion_sound.play()
```

g. goodShipCollide(): adds scorepoint and play “glass shattering” explosion sound
h. badCollide(): deduct one lifepoint, check if there’s no more life left(if so, then call the end game function) and play “glass shattering” explosion sound



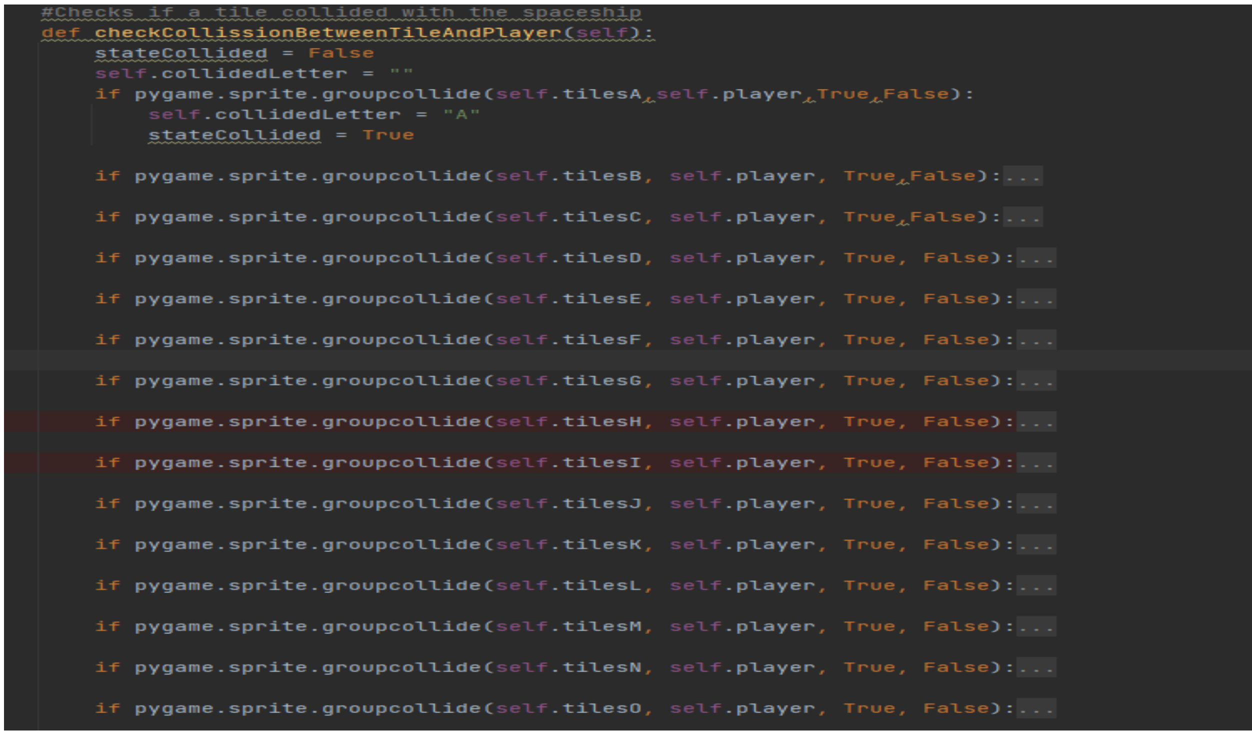
```
#Play random music endlessly(a random one once current music is over)
def playMusic(self):
    randomMusic = random.choice(self.musics)
    currentMusic = pygame.mixer.Sound("BGM/"+randomMusic)
    currentMusic.set_volume(0.2)
    currentMusic.play()
```

i. playMusic(): make a list from the background music directory, then randomly choose one to be played.



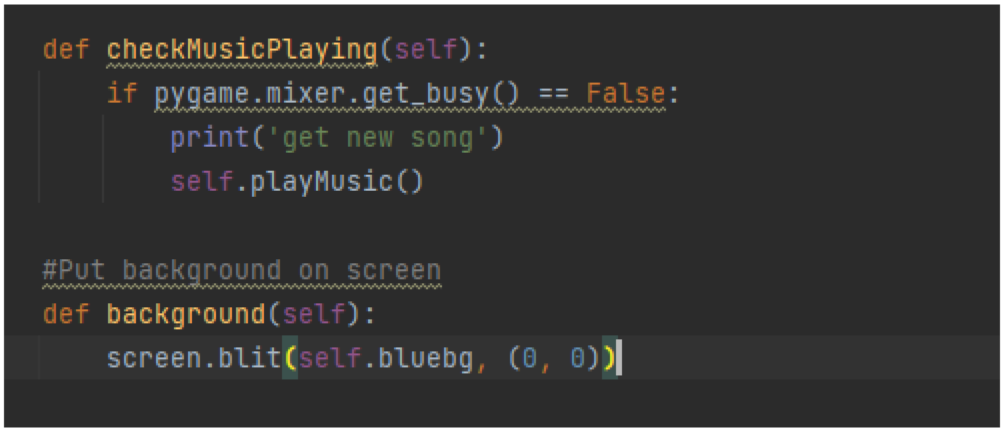
```
#Checks if a tile is hit by a laser
def checkCollisionBetweenTileAndLaser(self):
    if pygame.sprite.groupcollide(self.tiles, self.player.sprite.lasers, True, True):
        self.goodLaserCollide()
```

j. checkCollissionBetweenTileAndLaser(): call goodLaserCollide function upon a collission between tiles and laser group



```python
#Checks if a tile collided with the spaceship
def checkCollissionBetweenTileAndPlayer(self):
    stateCollided = False
    self.collidedLetter = ""
    if pygame.sprite.groupcollide(self.tilesA,self.player,True,False):
        self.collidedLetter = "A"
        stateCollided = True

    if pygame.sprite.groupcollide(self.tilesB, self.player, True,False):...

    if pygame.sprite.groupcollide(self.tilesC, self.player, True,False):...

    if pygame.sprite.groupcollide(self.tilesD, self.player, True, False):...

    if pygame.sprite.groupcollide(self.tilesE, self.player, True, False):...

    if pygame.sprite.groupcollide(self.tilesF, self.player, True, False):...

    if pygame.sprite.groupcollide(self.tilesG, self.player, True, False):...

    if pygame.sprite.groupcollide(self.tilesH, self.player, True, False):...

    if pygame.sprite.groupcollide(self.tilesI, self.player, True, False):...

    if pygame.sprite.groupcollide(self.tilesJ, self.player, True, False):...

    if pygame.sprite.groupcollide(self.tilesK, self.player, True, False):...

    if pygame.sprite.groupcollide(self.tilesL, self.player, True, False):...

    if pygame.sprite.groupcollide(self.tilesM, self.player, True, False):...

    if pygame.sprite.groupcollide(self.tilesN, self.player, True, False):...

    if pygame.sprite.groupcollide(self.tilesO, self.player, True, False):...
```

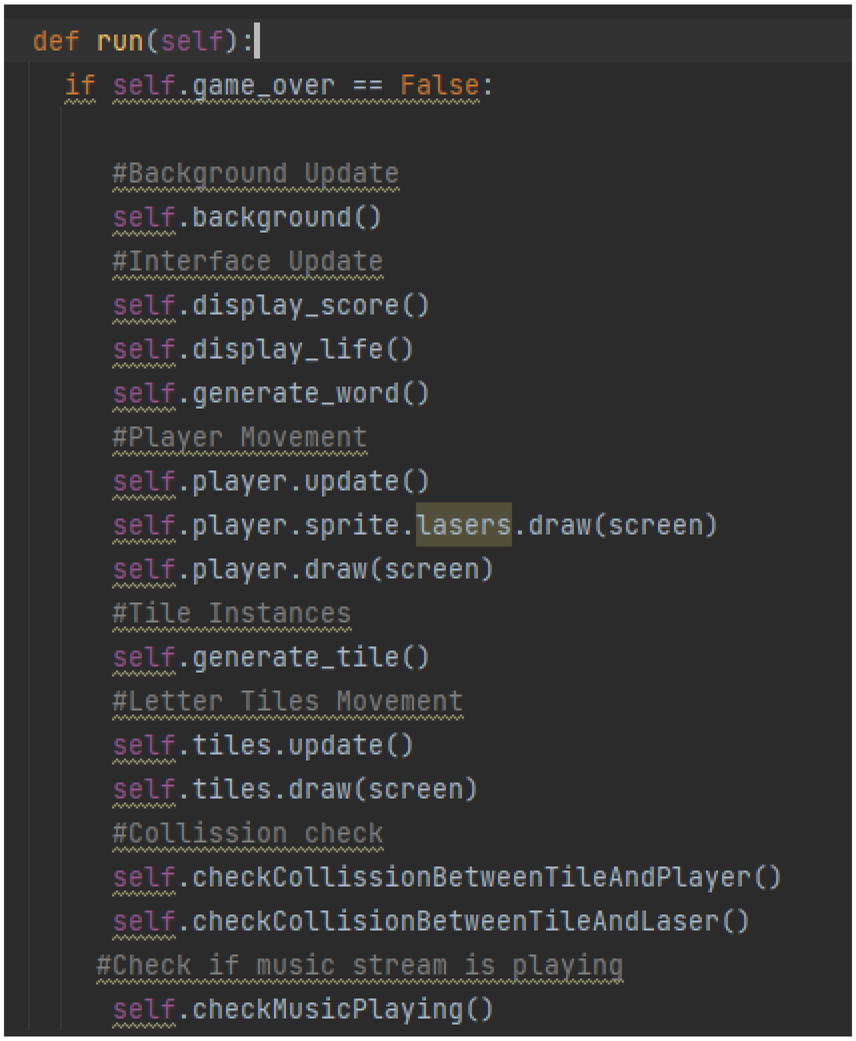
k. checkCollissionBetweenTileAndPlayer(): upon collission between tile and player group, determine which group the tile belong to assign the letter to a variable, check if the letter is in target: if is, then call goodShipCollide function but if not, then call badCollide function.



```python
def checkMusicPlaying(self):
    if pygame.mixer.get_busy() == False:
        print('get new song')
        self.playMusic()

#Put background on screen
def background(self):
    screen.blit(self.bluebg, (0, 0))
```

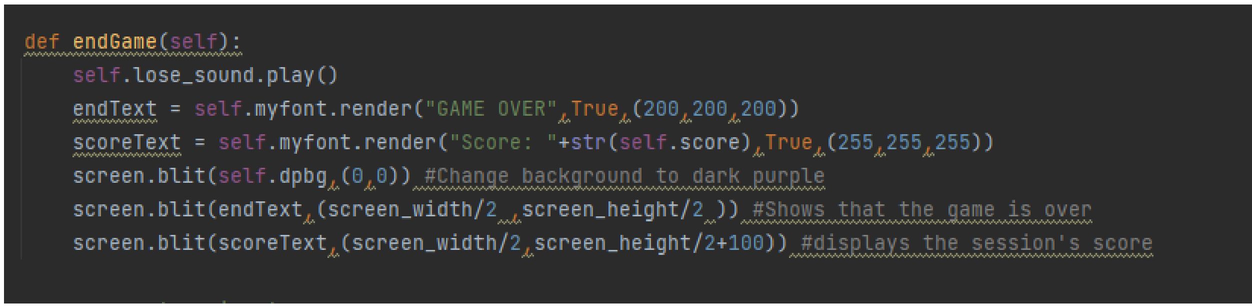
l. checkMusicPlaying(): check whether music streamer is busy or not, when it's not, then call the playMusic function.

m. background(): draw the background on the screen



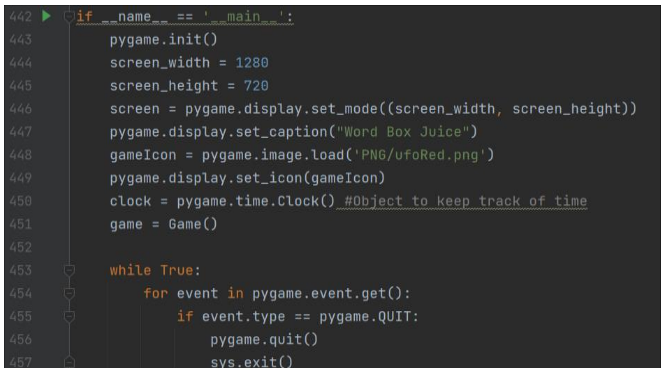
```python
def run(self):
    if self.game_over == False:

        #Background Update
        self.background()
        #Interface Update
        self.display_score()
        self.display_life()
        self.generate_word()
        #Player Movement
        self.player.update()
        self.player.sprite.lasers.draw(screen)
        self.player.draw(screen)
        #Tile Instances
        self.generate_tile()
        #Letter Tiles Movement
        self.tiles.update()
        self.tiles.draw(screen)
        #Collission check
        self.checkCollissionBetweenTileAndPlayer()
        self.checkCollisionBetweenTileAndLaser()
       #Check if music stream is playing
        self.checkMusicPlaying()
```

n. run(): call all the functons for update and checking conditions



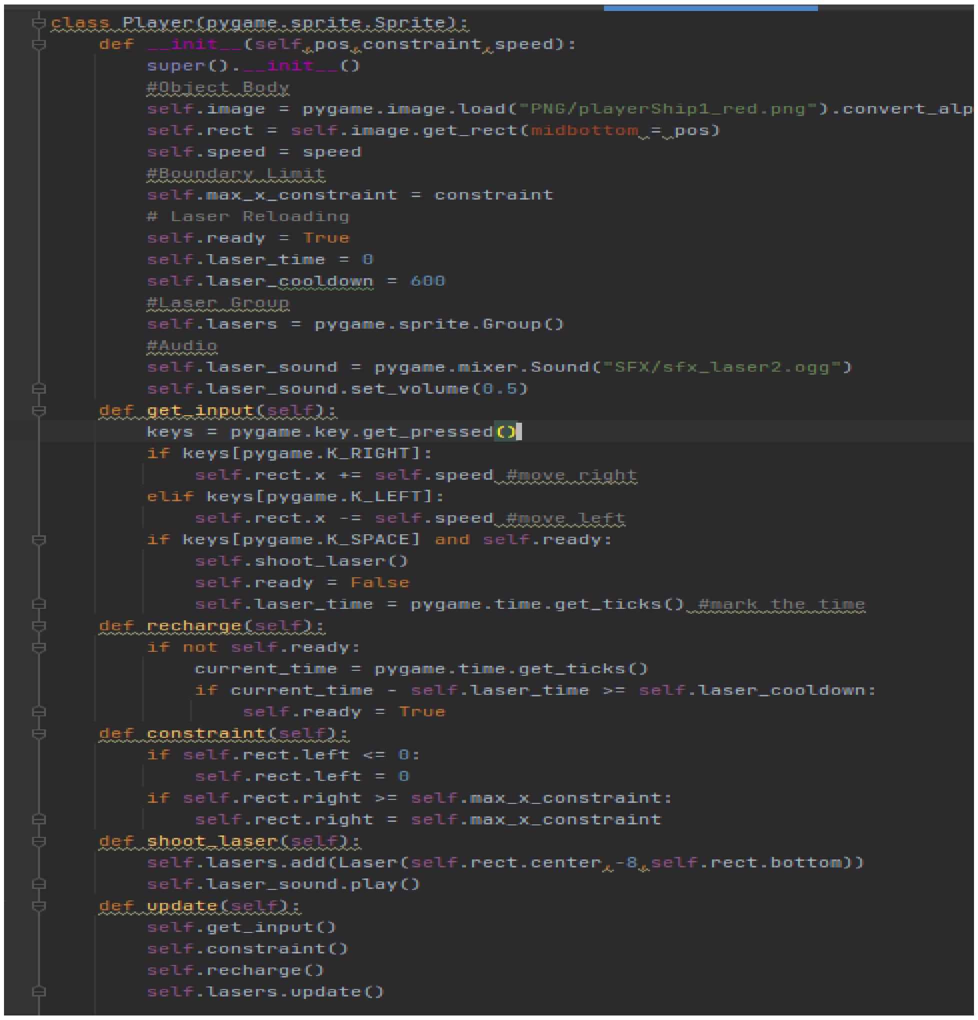
```python
def endGame(self):
    self.lose_sound.play()
    endText = self.myfont.render("GAME OVER",True,(200,200,200))
    scoreText = self.myfont.render("Score: "+str(self.score),True,(255,255,255))
    screen.blit(self.dpbg,(0,0)) #Change background to dark purple
    screen.blit(endText,(screen_width/2 ,screen_height/2 )) #Shows that the game is over
    screen.blit(scoreText,(screen_width/2,screen_height/2+100)) #displays the session's score
```

o. endgame(): change the background and displays a ‘GAME OVER’ text along with the session’s score



```python
if __name__ == '__main__':
    pygame.init()
    screen_width = 1280
    screen_height = 720
    screen = pygame.display.set_mode((screen_width, screen_height))
    pygame.display.set_caption("Word Box Juice")
    gameIcon = pygame.image.load('PNG/ufoRed.png')
    pygame.display.set_icon(gameIcon)
    clock = pygame.time.Clock() #Object to keep track of time
    game = Game()

    while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
```

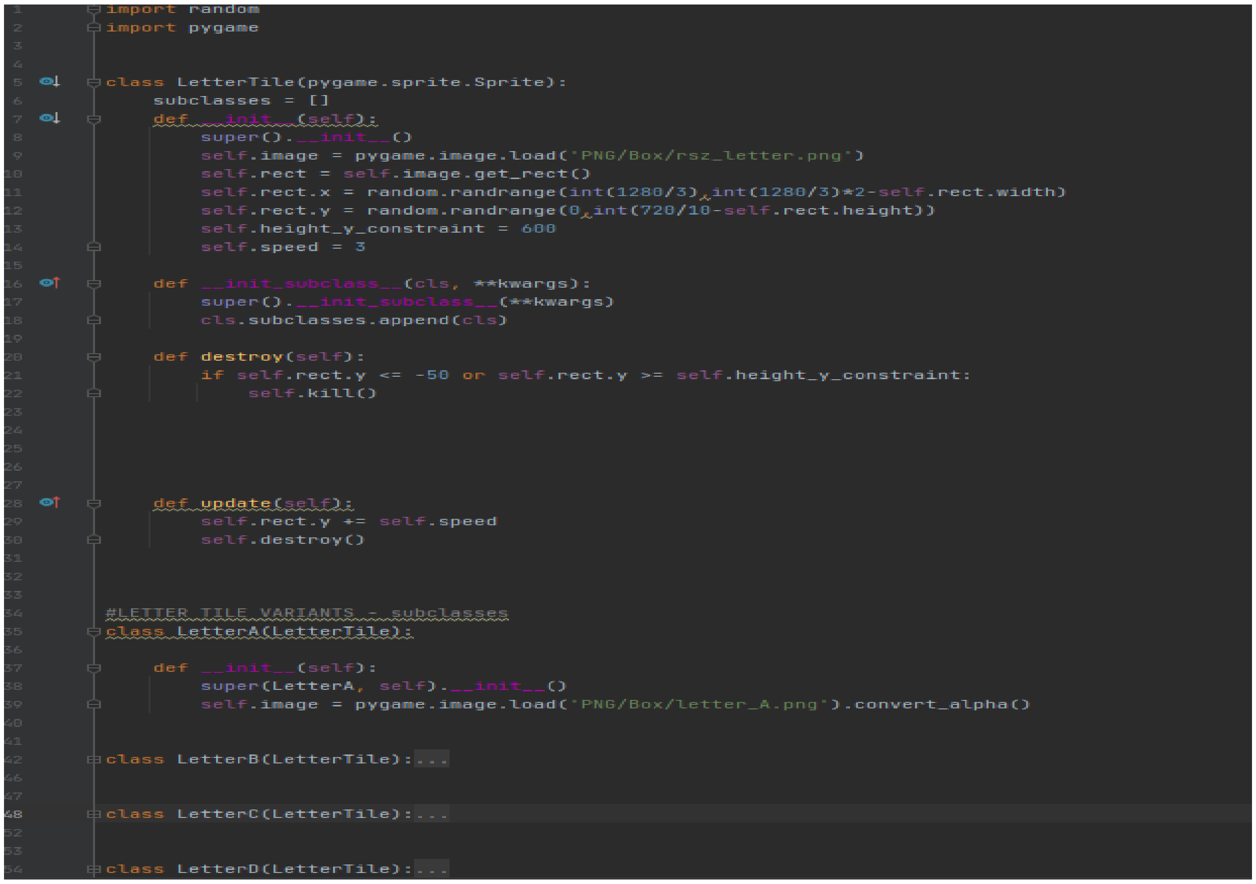
Outside the Game Class, there’s no function, but it has it’s purpose to initialize the pygame window screen, along with basic details such as caption,icon,clock and calling the Game run function. Most importantly, it updates the entire display at an assigned framerate(60fps).



```python
class Player(pygame.sprite.Sprite):
    def __init__(self,pos,constraint,speed):
        super().__init__()
        #Object Body
        self.image = pygame.image.load("PNG/playerShip1_red.png").convert_alp
        self.rect = self.image.get_rect(midbottom = pos)
        self.speed = speed
        #Boundary Limit
        self.max_x_constraint = constraint
        # Laser Reloading
        self.ready = True
        self.laser_time = 0
        self.laser_cooldown = 600
        #Laser Group
        self.lasers = pygame.sprite.Group()
        #Audio
        self.laser_sound = pygame.mixer.Sound("SFX/sfx_laser2.ogg")
        self.laser_sound.set_volume(0.5)
    def get_input(self):
        keys = pygame.key.get_pressed()
        if keys[pygame.K_RIGHT]:
            self.rect.x += self.speed #move right
        elif keys[pygame.K_LEFT]:
            self.rect.x -= self.speed #move left
        if keys[pygame.K_SPACE] and self.ready:
            self.shoot_laser()
            self.ready = False
            self.laser_time = pygame.time.get_ticks() #mark the time
    def recharge(self):
        if not self.ready:
            current_time = pygame.time.get_ticks()
            if current_time - self.laser_time >= self.laser_cooldown:
                self.ready = True
    def constraint(self):
        if self.rect.left <= 0:
            self.rect.left = 0
        if self.rect.right >= self.max_x_constraint:
            self.rect.right = self.max_x_constraint
    def shoot_laser(self):
        self.lasers.add(Laser(self.rect.center,-8,self.rect.bottom))
        self.laser_sound.play()
    def update(self):
        self.get_input()
        self.constraint()
        self.recharge()
        self.lasers.update()
```

- ships.py

a. init() : assign values for object body members, width constraint, laser reload factors, laser group and laser sound effects.
b. get_input(): receive responses from pressed key arrows (left&right) for player ship's movement. Pressing the space key will result in shooting lasers if shooting cooldown condition(self.ready) is met.
c. recharge(): check whether cooldown time has passed and if so, player ship can shoot laser again.
d. constraint(): check whether the rect positions are more than screen width or height, and if so, move rect back at constraint limit
e. shoot_laser(): create a laser instance based on player ship's rect position and play the laser sound effect
f. update(): call for all four ships' functions (excluding init)



```python
import random
import pygame


class LetterTile(pygame.sprite.Sprite):
    subclasses = []
    def __init__(self):
        super().__init__()
        self.image = pygame.image.load('PNG/Box/rsz_letter.png')
        self.rect = self.image.get_rect()
        self.rect.x = random.randrange(int(1280/3),int(1280/3)*2-self.rect.width)
        self.rect.y = random.randrange(0,int(720/10-self.rect.height))
        self.height_y_constraint = 600
        self.speed = 3

    def __init_subclass__(cls, **kwargs):
        super().__init_subclass__(**kwargs)
        cls.subclasses.append(cls)

    def destroy(self):
        if self.rect.y <= -50 or self.rect.y >= self.height_y_constraint:
            self.kill()




    def update(self):
        self.rect.y += self.speed
        self.destroy()



#LETTER TILE VARIANTS - subclasses
class LetterA(LetterTile):

    def __init__(self):
        super(LetterA, self).__init__()
        self.image = pygame.image.load('PNG/Box/letter_A.png').convert_alpha()


class LetterB(LetterTile):...


class LetterC(LetterTile):...


class LetterD(LetterTile):...
```

- letterTile.py

Contains a Super and many Subclasses.

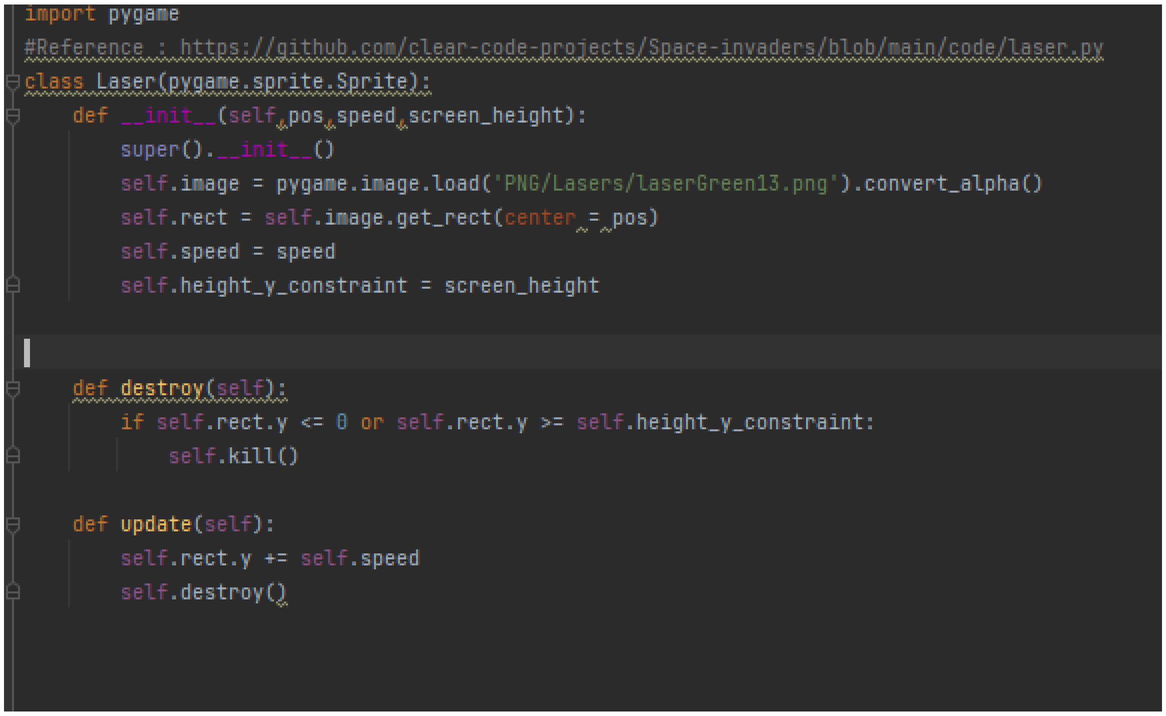
The Super Class has the following functions:

a. Init() : assign values for image, rect, height constraint and speed

b. init subclass (): append it’s subclasses to a list for randomization in generating tiles
c. Destroy(): once the tile breaks the constraint limit, the tile instance will get destroyed
d. update (): moves the tile downwards and calls the destroy function

The Subclasses have the following function:

a. init (): assign a new value for image, based on the alphabet letter



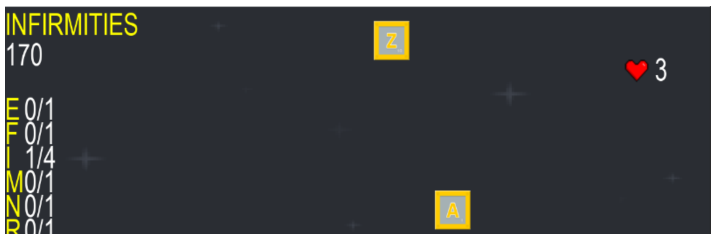
```python
import pygame
#Reference : https://github.com/clear-code-projects/Space-invaders/blob/main/code/laser.py
class Laser(pygame.sprite.Sprite):
    def __init__(self,pos,speed,screen_height):
        super().__init__()
        self.image = pygame.image.load('PNG/Lasers/laserGreen13.png').convert_alpha()
        self.rect = self.image.get_rect(center = pos)
        self.speed = speed
        self.height_y_constraint = screen_height


    def destroy(self):
        if self.rect.y <= 0 or self.rect.y >= self.height_y_constraint:
            self.kill()

    def update(self):
        self.rect.y += self.speed
        self.destroy()
```

- laser.py

Contains the Laser class, which has several functions:

a. init (): assign values to laser instances such as it’s image, rect, speed and boundary limit
b. destroy () : once the laser breakthrough the boundary limit, the instance will be destroyed
c. update (): moves the laser upwards and calls the destroy function

Evidence



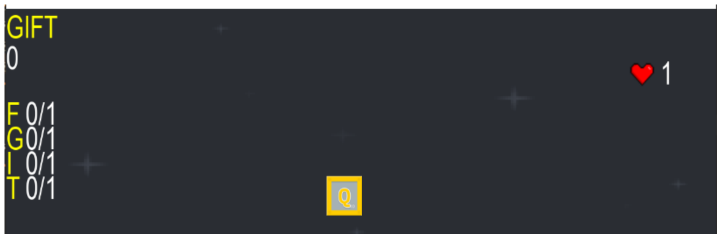


Image 1.1 Collected a Letter

Image 1.2 Received 2 Damages

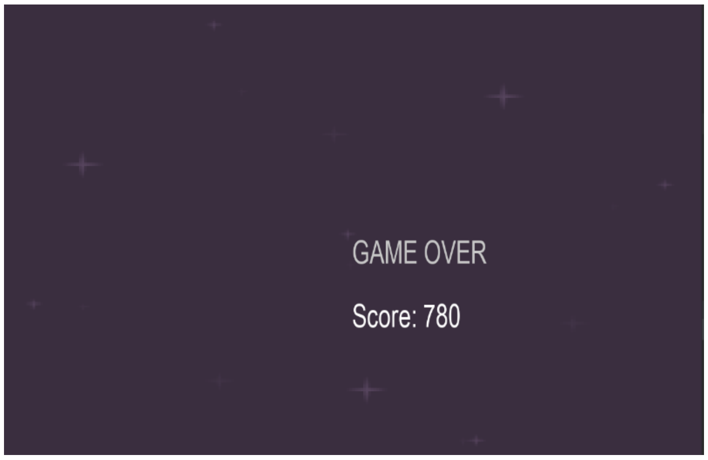


Image 1.3 Game Over Screen