Project Report

Project Name: The last wizard of 3022

Name: Arish Madataly
Student ID: 2502049706
Class: L1CC

Table of content

I. GAME OVERVIEW………………………………………………………… 2

II. INSPIRATION…………………………………………………………… 2

III. SOLUTION DESIGN…………………………………………………… 3

IV. MAIN MECHANISM………………………………………………….. 4

V. GAMEPLAY……………………………………………………………... 5

VI. EVIDENCE OF WORKING PROGRAM…………………………… 15

VII. POSSIBLE EXTENSIONS AND FUTURE AMELIORATION…… 18

VIII. SOURCES OF THE WEBSITES USED…………………………… 19

I. GAME OVERVIEW

Genre

This project is a 2D game called “The Last Wizard in 3022”. It’s a high score based game which means the goal is not to move any higher level but to get a higher score each time you play, with a playing time limited by your health level.

Visual aspect

The game was made to be played on a smaller scaled screen of 1200px of width and 800px of height. Animations have been included using sprite sheets to make it interactive and allow the user to feel the pressure and the fun of the game

Background story

This game takes place around a hundred years from now, in a society where robots have taken over the world. Heroes long forgotten called the wizards decided to fight them and free the society but the plan didn’t go as expected and they all got defeated apart from one. He is now fighting these robots alone with his fireballs.

Game concept

The player has certain health shown on the upper left part of the screen in form of a bar that goes down each time the enemies’ bullets hit him, the goal is to get the highest score possible before your health gets to 0. Your score shown right above the health bar goes up each time you kill an enemy. You can move up/down/left/right to try to dodge the attacks coming at you. The enemies also have their health shown in form of a bar on top of them and they keep respawning after they die, there are two different types of enemies with different velocities, bullets’ velocities and health, and of course different scores if you kill them. They keep shooting their laser bullets randomly moving back and forth from the place the player is at.

Specifications

This game is fairly simple that is best mainly on the competitiveness of getting a better score than before. It doesn’t have any complicated controls or shocking images, it is, therefore, suitable for all ages. It improves your reflexes to use the keys of the keyboard because you will have to dodge the bullets that go at a fairly high speed and the enemies will respawn randomly on the screen after their death.

II. INSPIRATION

Space invaders

The shooting concept was inspired by space invaders, while doing my research I found saw many different types of shooting games all looking like space invaders and I thought that maybe it could look better if instead of a spaceship it was a human shooting at enemies but the idea of a wizard shooting planes in the sky didn’t look that well. It’s at that moment that I decided that the player should be on the ground and fight some robots, I have never seen a game getting together wizards and robots so I thought it would be a good idea to try this new universe.

III. SOLUTION DESIGN

The project is the combination of a few different screens all together starting from the main menu, to then go to the instructions, the game screen and finally the end screen. And it is also the combination of 5 different python files, the ImagesAndSounds.py file which is where all the images and sounds were uploaded with the pygame module, the Players.py where all the characters' classes are regrouped such as the player' and the enemies' classes, the manual.py that contains the controls and instructions of the game, the game.py file where the game loop is found alongside with the end screen and finally the main.py file which contains the main game loop starting from the main menu and allowing the user to travel from one screen to another. (pictures in evidence of the working program)

Main menu

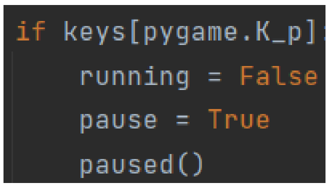
The main menu is the first screen we see when we open the game, it can lead to the other menus by clicking on the buttons. It has three options either go to the MANUAL page, to the game by clicking on the START button or to EXIT the game by clicking on the exit button. The high score is also present on the screen right under the title.

Manual

The user gets to the manual page after clicking on the manual button of the main menu, it displays a little information about what the game is about and the controls to play and how to pause the game while playing. In addition, on the bottom right corner, you can see a button that gives the user the possibility to reset his high score. A back button is available to go back to the main menu when the user is done.

Game

The user gets to the game screen after clicking on the start button of the main menu, there will be a loading screen for about 3 seconds while the images and the sound effects are loading and to avoid the enemies to shoot at the player before he is even ready to dodge. On the game screen, we see the player, the enemies, the health bars that reduce and the score that increases during the game. The player is also given the possibility to pause the game any time by pressing on the P key.



```
if keys[pygame.K_p]
    running = False
    pause = True
    paused()
```

Pause screen

The pause screen is accessible whenever the game is running by pressing on the P_key as stated above. The game is stopped and the user is given the possibility to either continue or quit the game by clicking on one of the buttons. If the user clicks on "continue" the game continues from where it stopped with the same score, the same health bars and even the bullets and the fireballs that were still moving when the key was pressed will continue from their position. If, on the other hand, the user decided to click on quit, he will go back to the main menu and lose all his progress on the ongoing game.

End screen

And finally, the end screen that appears at the end of the game when the player has 0 health points left and he has lost. The “GAME OVER” text appears on the screen followed by a voice saying it. The high score is displayed followed by the actual score and finally a message that tells you what you can do next. You can either restart by pressing the R key and you’ll see the loading screen again before going back to the game with full health and a fresh score, or you could go back to the main menu by pressing the ESCAPE key.

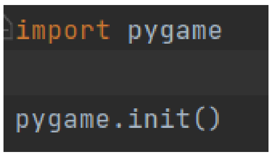
IV. MAIN MECHANISM

Main objective

As stated before, “The last wizard in 3022” is like a basic arcade game where the main goal is to get the highest score possible. The enemies will keep respawning so as long as the player has some health he has the possibility to increase his score.

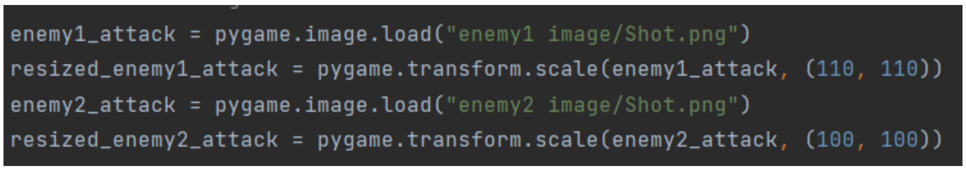
Functions and modules

The whole game was made using a python library called pygame designed especially for the creation of video games, it allows the creation of animations and the implementation of sound effects more easily and effectively. After importing the module, a simple initiation of the module and all its features are available to any user for free.

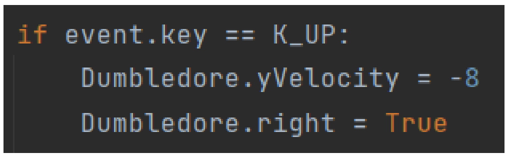


```
import pygame

pygame.init()
```

The pygame functions to load images onto the screen and the one that detects any keyboard inputs are the foundation of my game. A few examples are shown below.



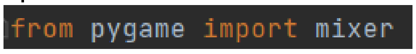
```
enemy1_attack = pygame.image.load("enemy1 image/Shot.png")
resized_enemy1_attack = pygame.transform.scale(enemy1_attack, (110, 110))
enemy2_attack = pygame.image.load("enemy2 image/Shot.png")
resized_enemy2_attack = pygame.transform.scale(enemy2_attack, (100, 100))
```



```
if event.key == K_UP:
    Dumbledore.yVelocity = -8
    Dumbledore.right = True
```

The sprites were gotten from an open-source called crafpix, the images and sound effects were uploaded in a different file using built-in pygame functions. The “pygame.image.load” was used to upload the different images into the game and then the “transform.scale” was used to change the sizes because they were either too big or too small.

The sound effects were also gotten from two different open sources freesound and mixkit, they were added into pygame built-in function that is called a mixer specifically used for sounds, it needs to be imported before it can be used because it is not directly included in the pygame module.



```
from pygame import mixer
```

The clock functionality of pygame was also used to help control the framerate, I decided to set it at 40 after a few trials and errors I concluded that this was the perfect one for my game, not too slow and not too fast.

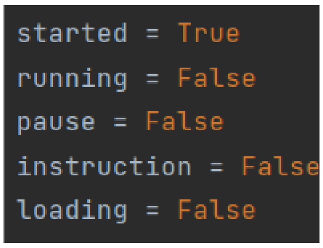
The “pygame.display.update” was also used a lot in my code, it’s a basic pygame function to make the display Surface that I made with all the draw methods actually appear on the user’s monitor.

Classes

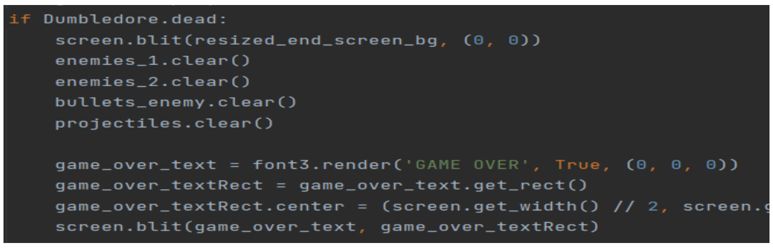
Apart from this python module, I also used quite a lot of classes in order to make my characters and players, it made it easier to create the enemies on the screen and also it made my code more organized, all of these will be explored in more details later on.

While loops

Each screen is represented by a while that depends on a specific variable so that it doesn’t close unless that variable is changed to false by an action of the user. The main menu screen depends on the “started” variable, the manual on the “instruction” variable, the loading screen on the “loading” variable, the main game on the “running” variable, the pause screen on the “pause” variable and finally the end screen which is not really based on the variable but more on the state of the player, it depends on the dead attribute of the player. All those variables assure the ability of the game to go from one screen to another as well as the ability to stay on the page unless the user decides otherwise (apart from the loading screen that disappears after 3s)

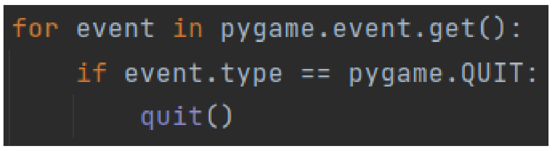


```
started = True
running = False
pause = False
instruction = False
loading = False
```



```
if Dumbledore.dead:
    screen.blit(resized_end_screen_bg, (0, 0))
    enemies_1.clear()
    enemies_2.clear()
    bullets_enemy.clear()
    projectiles.clear()

    game_over_text = font3.render('GAME OVER', True, (0, 0, 0))
    game_over_textRect = game_over_text.get_rect()
    game_over_textRect.center = (screen.get_width() // 2, screen.
    screen.blit(game_over_text, game_over_textRect)
```

Each screen also has an exit function that is built-in into the pygame module, it allows the user to click on the red cross on top of the screen to exit the game, the main menu also has it in addition to the EXIT button, it is also a safety if the game starts to lag, the user can use it to exit.



```
for event in pygame.event.get():
    if event.type == pygame.QUIT:
        quit()
```

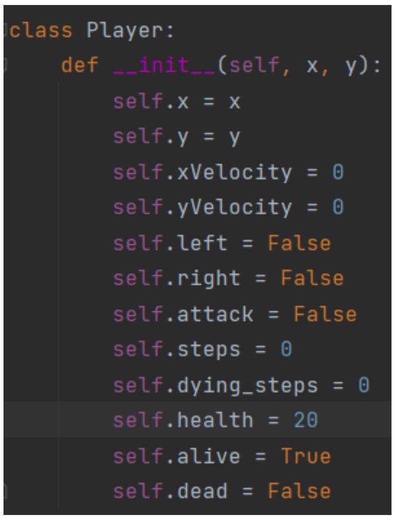
V. GAMEPLAY

Characters

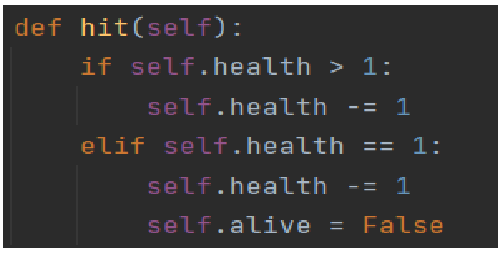
The game has basically three types of characters which are the player, the enemies type 1 and the enemies type 2. To make their use in the game loop more organized and easier, I made classes for each of them: And they also have a draw method that will take as arguments the images they require for the animations such as the image moving to the right or left, … These draw functions are composed of loops that will go through the list of images to create the animations.

- **<u>The player</u>**
  - <u>Attributes and methods</u>

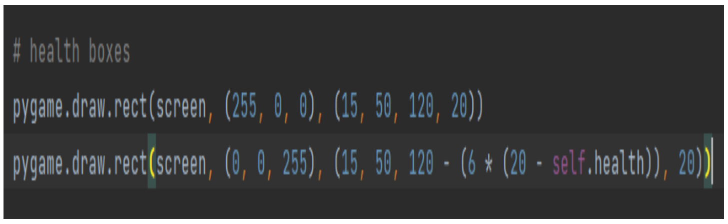
The main player the user is going to control is called by the class Player() shown below, his attributes are x and y that determines its starting position on the screen, its x and y velocities that will be set when he moves and will represent his speed on the screen, his health that will be decreased each he gets hit by a bullet this made possible by the hit() method that reduces the health each time it is called in the game loop and finally set to the player to a dead state when the health gets to 0. The health is not only made as a number but also represented as a health bar with a width that decreases when the player gets hit, it is made of two rectangles of different colours on top of each, one showing only when the other start getting shorter, it gives an illusion of it decreasing.



```python
class Player:
    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.xVelocity = 0
        self.yVelocity = 0
        self.left = False
        self.right = False
        self.attack = False
        self.steps = 0
        self.dying_steps = 0
        self.health = 20
        self.alive = True
        self.dead = False
```

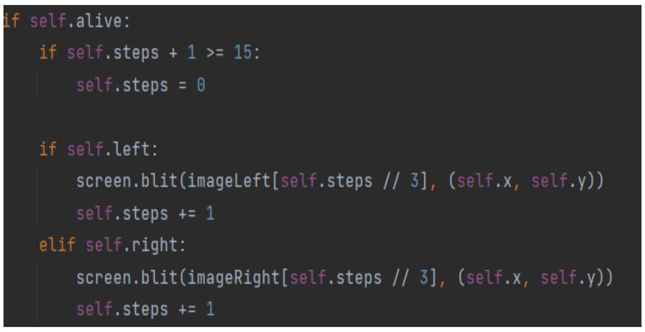


```python
def hit(self):
    if self.health > 1:
        self.health -= 1
    elif self.health == 1:
        self.health -= 1
        self.alive = False
```

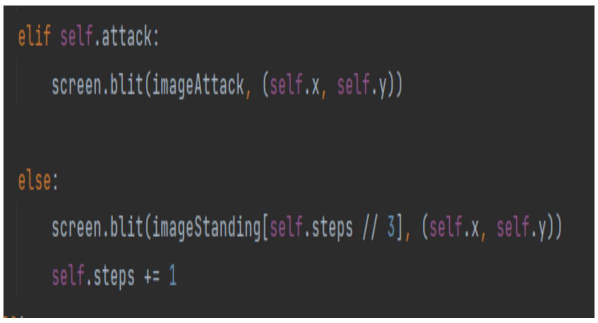


```python
# health boxes
pygame.draw.rect(screen, (255, 0, 0), (15, 50, 120, 20))
pygame.draw.rect(screen, (0, 0, 255), (15, 50, 120 - (6 * (20 - self.health)), 20))
```

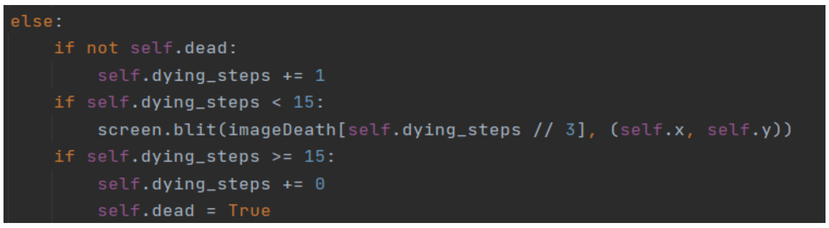
The other attributes were made to allow the animations in the draw method of the class, the left and right attributes are used to start and stop the different animations depending on the direction the player will move, or even if he isn't moving. The attack attribute is used to draw the image of the player in the attack position when the user presses the space bar to throw fireballs. I made the steps and dying steps because I used to list to make the animations and I needed a variable to determine which image of the list should be drawn and when to create a good animation. As seen below I decided to change the image every three steps that is why the limit is at 15 because there are five images for each state of the player. The dying steps were necessary because the steps depend on the user's input while the death animation does not need any user action, that is why the dying steps are automatic.



```python
if self.alive:
    if self.steps + 1 >= 15:
        self.steps = 0

    if self.left:
        screen.blit(imageLeft[self.steps // 3], (self.x, self.y))
        self.steps += 1
    elif self.right:
        screen.blit(imageRight[self.steps // 3], (self.x, self.y))
        self.steps += 1
```

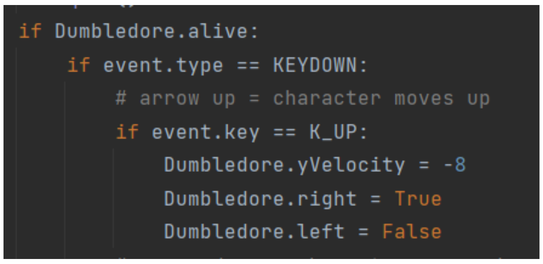


```python
elif self.attack:
    screen.blit(imageAttack, (self.x, self.y))


else:
    screen.blit(imageStanding[self.steps // 3], (self.x, self.y))
    self.steps += 1
```

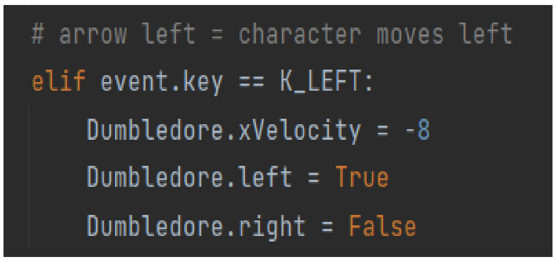


```
else:
    if not self.dead:
        self.dying_steps += 1
    if self.dying_steps < 15:
        screen.blit(imageDeath[self.dying_steps // 3], (self.x, self.y))
    if self.dying_steps >= 15:
        self.dying_steps += 0
        self.dead = True
```

- <u>Movements</u>
  The wizard can move up/down/left/right to dodge the enemies, the movements in the y axis are quite free but the ones in the x-axis are limited because the enemies' bullets only go in one direction and being able to get behind them will make the game a bit too easy. His movements are controlled by the arrows of the keyboard if a key is pressed he starts moving at a constant velocity and when the key is released he stops.
  He can shoot fireballs when the user presses the space bar, his projectile is defined by the Fireball class that will be seen more in detail in the projectile section.

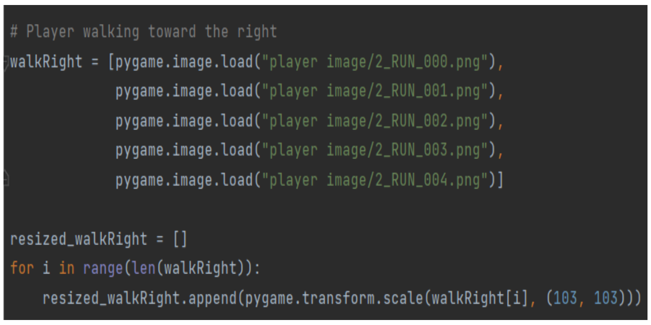


```
if Dumbledore.alive:
    if event.type == KEYDOWN:
        # arrow up = character moves up
        if event.key == K_UP:
            Dumbledore.yVelocity = -8
            Dumbledore.right = True
            Dumbledore.left = False
```

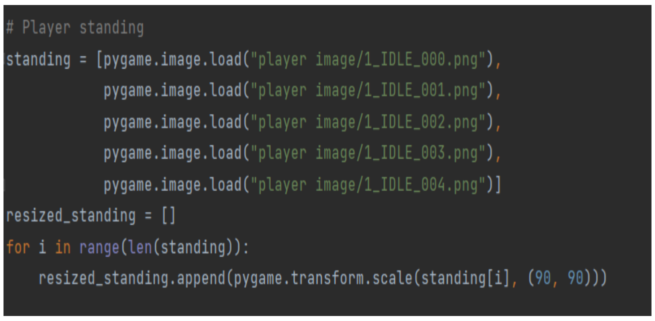


```
# arrow left = character moves left
elif event.key == K_LEFT:
    Dumbledore.xVelocity = -8
    Dumbledore.left = True
    Dumbledore.right = False
```

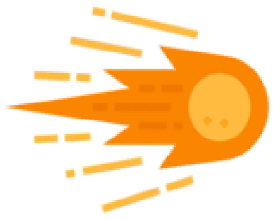
- <u>Images and other animations effects</u>
  For the player, I uploaded the images into lists that will be on repeat when a game is running thus making the illusion of the player running to the left, to the right, or him moving a bit even when standing. Only the attack image and the fireball were made of a single image because their animations were made using the position during the game.



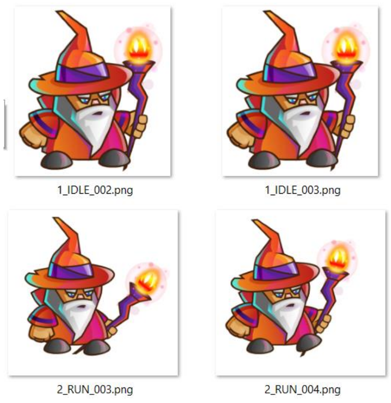
```
# Player walking toward the right
walkRight = [pygame.image.load("player image/2_RUN_000.png"),
             pygame.image.load("player image/2_RUN_001.png"),
             pygame.image.load("player image/2_RUN_002.png"),
             pygame.image.load("player image/2_RUN_003.png"),
             pygame.image.load("player image/2_RUN_004.png")]

resized_walkRight = []
for i in range(len(walkRight)):
    resized_walkRight.append(pygame.transform.scale(walkRight[i], (103, 103)))
```



```
# Player standing
standing = [pygame.image.load("player image/1_IDLE_000.png"),
            pygame.image.load("player image/1_IDLE_001.png"),
            pygame.image.load("player image/1_IDLE_002.png"),
            pygame.image.load("player image/1_IDLE_003.png"),
            pygame.image.load("player image/1_IDLE_004.png")]
resized_standing = []
for i in range(len(standing)):
    resized_standing.append(pygame.transform.scale(standing[i], (90, 90)))
```

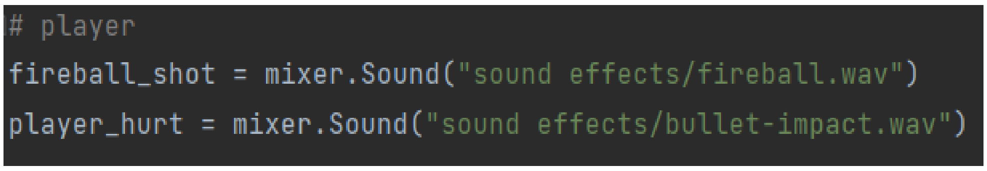


The fireball image



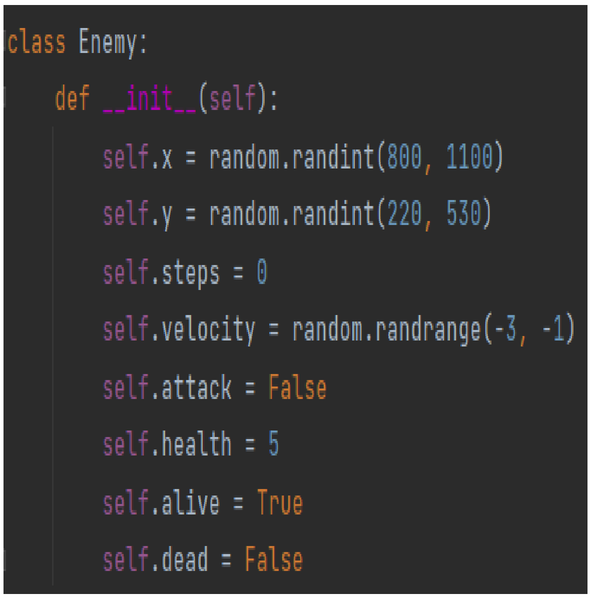
A few example of the images of the player

The player has two sound effects associated with him, a sound of a fireball being launched when he attacks and the sound of the impact being him and the enemies' bullets.

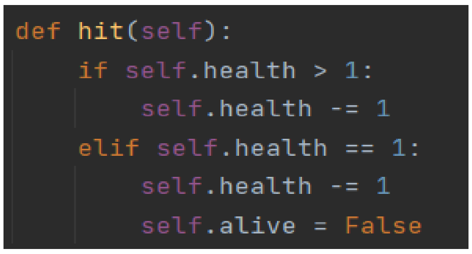


```
# player
fireball_shot = mixer.Sound("sound effects/fireball.wav")
player_hurt = mixer.Sound("sound effects/bullet-impact.wav")
```

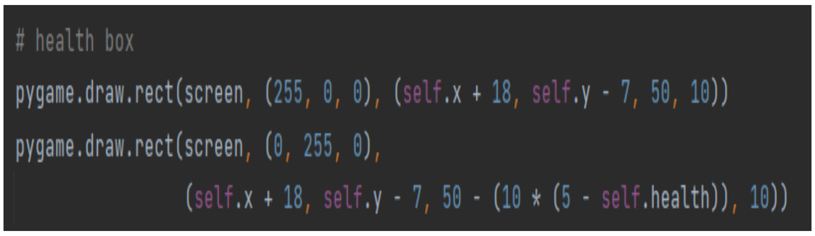
- **<u>The enemies</u>**
  - <u>Attributes and methods</u>
    In this game, the enemies are robots and there are two types of them. The first one is called by the class Enemy() shown below, it has the x and y values that represent the position they are going to appear on the screen, to avoid the player knowing where exactly they will appear they appear at a random position in a certain range. Its velocity varies between 1 and 3, there are negative at the beginning just because of the direction in which it's going when it first appears which is toward the player and the as we when we move from right to left the y component decreases. Its health is fixed to 5, same as the player it has a health bar but this one is on top of its head to see which enemy is actually losing health in case two or more enemies are in the trajectory of the fireball. The health bar is also made of two rectangles the one on top that decrease in which each time the robot loses health when the hit() method is called.



```python
class Enemy:
    def __init__(self):
        self.x = random.randint(800, 1100)
        self.y = random.randint(220, 530)
        self.steps = 0
        self.velocity = random.randrange(-3, -1)
        self.attack = False
        self.health = 5
        self.alive = True
        self.dead = False
```



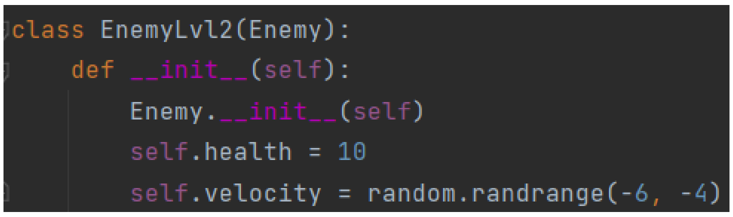
```python
def hit(self):
    if self.health > 1:
        self.health -= 1
    elif self.health == 1:
        self.health -= 1
        self.alive = False
```



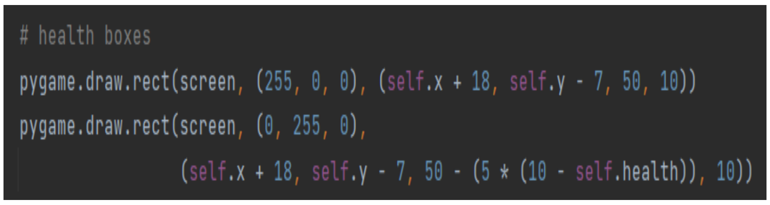
```python
# health box
pygame.draw.rect(screen, (255, 0, 0), (self.x + 18, self.y - 7, 50, 10))
pygame.draw.rect(screen, (0, 255, 0),
                 (self.x + 18, self.y - 7, 50 - (10 * (5 - self.health)), 10))
```

The width decreases as the health decreases, the operation is to get the proportions right, by how much should the width decrease for each HP lost.

The second type is called by the class EnemyLvl2() that inherits all the attributes and methods of the Enemy() class. The only differences are the images, the health that is 10 instead of 5 and the range of velocity that goes from 4 to 6, the draw() method had to be changed slightly due to the health changing the width of the rectangle on top that decreases in width. These were created to increase the difficulty.

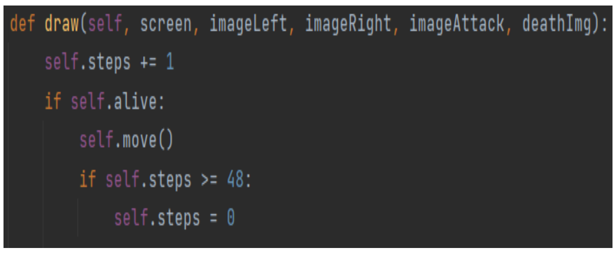


```python
class EnemyLvl2(Enemy):
    def __init__(self):
        Enemy.__init__(self)
        self.health = 10
        self.velocity = random.randrange(-6, -4)
```

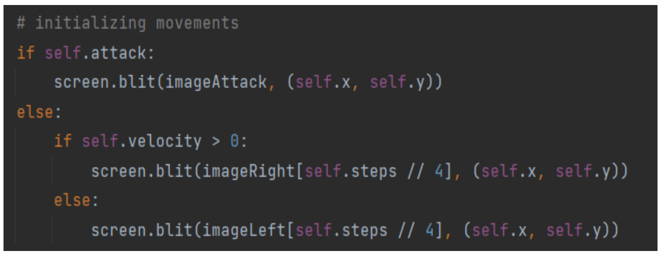


```python
# health boxes
pygame.draw.rect(screen, (255, 0, 0), (self.x + 18, self.y - 7, 50, 10))
pygame.draw.rect(screen, (0, 255, 0),
                 (self.x + 18, self.y - 7, 50 - (5 * (10 - self.health)), 10))
```

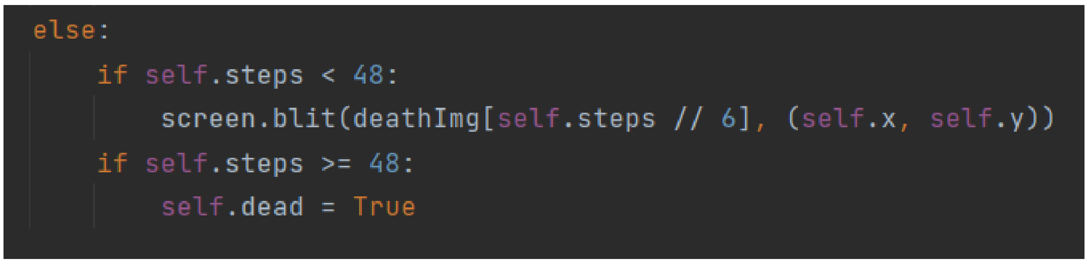
The other attributes are the same as the player, they were made to control the animations that should be displayed. The difference is that for the enemy, the number of steps is automatic like its movements but the image to display in the list is still dependent on the steps. This time it goes back to 0 when it gets to 48 because there are 12 images and I change them every 4 steps. The attack attribute is used each time the enemy shoots a bullet which is at random. The alive method is to make him stop everything when his health gets to 0 and the dead one is to have an animation of him falling when he dies, no need to put dying steps because its steps are already automatic.



```python
def draw(self, screen, imageLeft, imageRight, imageAttack, deathImg):
    self.steps += 1
    if self.alive:
        self.move()
        if self.steps >= 48:
            self.steps = 0
```

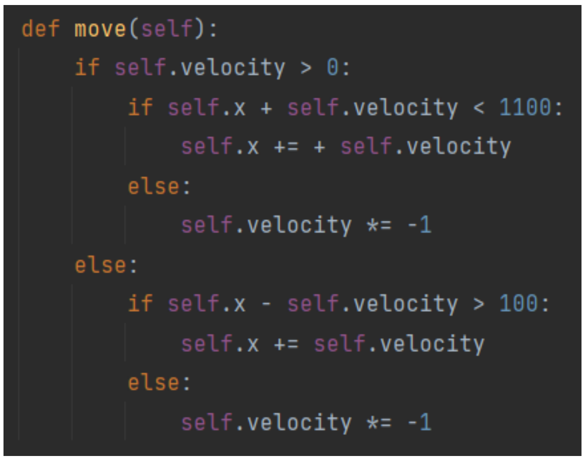


```python
# initializing movements
if self.attack:
    screen.blit(imageAttack, (self.x, self.y))
else:
    if self.velocity > 0:
        screen.blit(imageRight[self.steps // 4], (self.x, self.y))
    else:
        screen.blit(imageLeft[self.steps // 4], (self.x, self.y))
```

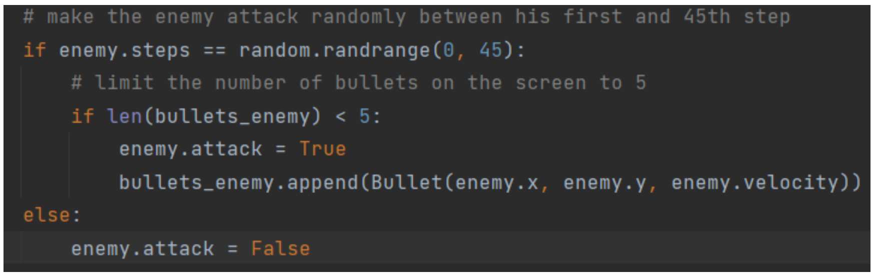


```
else:
    if self.steps < 48:
        screen.blit(deathImg[self.steps // 6], (self.x, self.y))
    if self.steps >= 48:
        self.dead = True
```

- Movements
  The enemies' will move left and right automatically within certain boundaries, because of their movements not being controlled by any user input, another of controlling them automatically was required. That is why I created the move() method, it controls the movements of the enemies by limiting them on the x-axis, moreover, it reverses the velocity and makes the enemy go backwards or forward when they get to the limits imposed which are between 100 and 1100. They also shoot bullets at random moments that are defined in the main game loop but the number of bullets present on the screen is limited to 5, therefore if there are already 5 bullets on the screen the enemies have to wait for them to disappear before shooting again. And are defined by the class Bullet() that will be looked at more in detail later on in the projectile section.



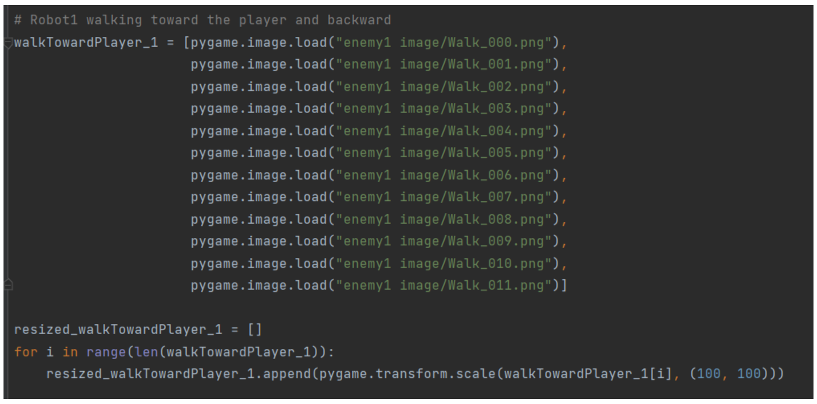
```
def move(self):
    if self.velocity > 0:
        if self.x + self.velocity < 1100:
            self.x += + self.velocity
        else:
            self.velocity *= -1
    else:
        if self.x - self.velocity > 100:
            self.x += self.velocity
        else:
            self.velocity *= -1
```



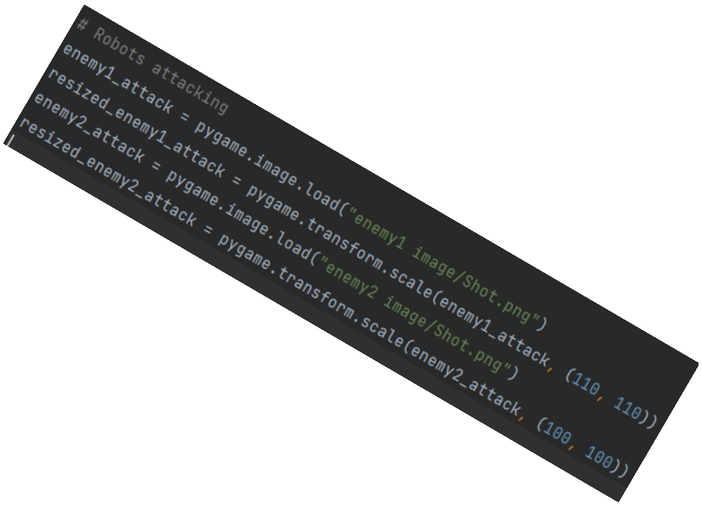
```
# make the enemy attack randomly between his first and 45th step
if enemy.steps == random.randrange(0, 45):
    # limit the number of bullets on the screen to 5
    if len(bullets_enemy) < 5:
        enemy.attack = True
        bullets_enemy.append(Bullet(enemy.x, enemy.y, enemy.velocity))
else:
    enemy.attack = False
```

This if statement basically states that in the range of its 45 steps the enemy will shoot bullets at random times, I made it in the range of 45 to be able to place the attack image when they attack so that it matches with the other images.

- Images and other animations effects
  As for the player, the images of the enemies were added into lists that will be on repeat while the game is running, thus creating some kind of illusion that the enemies are moving, also an illusion of falling when they die. The attack image is the only one that is made of a single image because as the player its animation is made by changing its position during the game



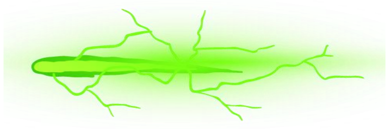
```
# Robot1 walking toward the player and backward
walkTowardPlayer_1 = [pygame.image.load("enemy1 image/Walk_000.png"),
                      pygame.image.load("enemy1 image/Walk_001.png"),
                      pygame.image.load("enemy1 image/Walk_002.png"),
                      pygame.image.load("enemy1 image/Walk_003.png"),
                      pygame.image.load("enemy1 image/Walk_004.png"),
                      pygame.image.load("enemy1 image/Walk_005.png"),
                      pygame.image.load("enemy1 image/Walk_006.png"),
                      pygame.image.load("enemy1 image/Walk_007.png"),
                      pygame.image.load("enemy1 image/Walk_008.png"),
                      pygame.image.load("enemy1 image/Walk_009.png"),
                      pygame.image.load("enemy1 image/Walk_010.png"),
                      pygame.image.load("enemy1 image/Walk_011.png")]

resized_walkTowardPlayer_1 = []
for i in range(len(walkTowardPlayer_1)):
    resized_walkTowardPlayer_1.append(pygame.transform.scale(walkTowardPlayer_1[i], (100, 100)))
```



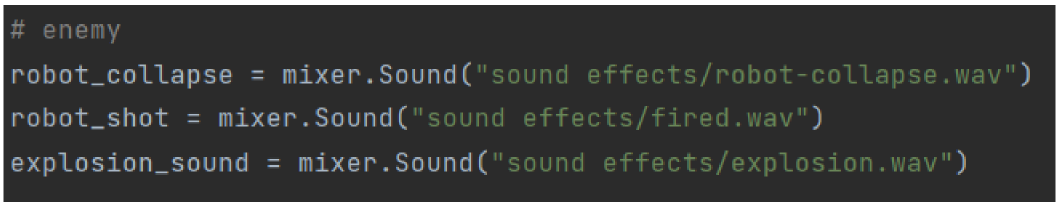
```
# Robots attacking
enemy1_attack = pygame.image.load("enemy1 image/Shot.png")
resized_enemy1_attack = pygame.transform.scale(enemy1_attack, (110, 110))
enemy2_attack = pygame.image.load("enemy2 image/Shot.png")
resized_enemy2_attack = pygame.transform.scale(enemy2_attack, (100, 100))
```

Example of robot1 image

Example of robot2 image


Enemies' bullet image

As per the sound effects, there are three associated with the enemies and apply to both types, one for the bullet being shot which is the same as a bullet being shot in reality, one for the dying which is the sound of a machine collapsing.

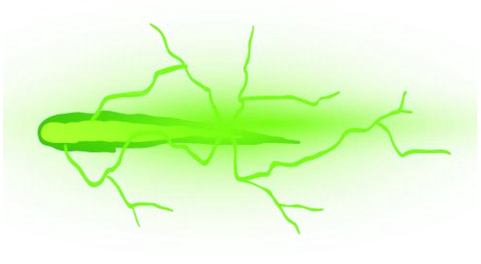


```
# enemy
robot_collapse = mixer.Sound("sound effects/robot-collapse.wav")
robot_shot = mixer.Sound("sound effects/fired.wav")
explosion_sound = mixer.Sound("sound effects/explosion.wav")
```

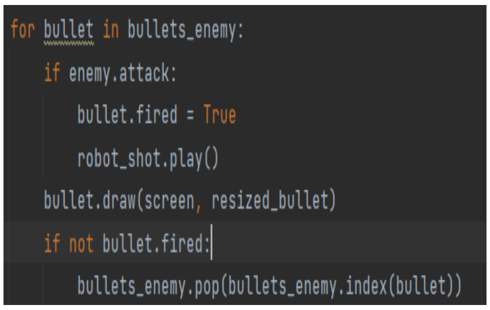
Projectiles

As introduced briefly in the previous section, there are two types of projectiles, the fireball of the main player and the bullets of the enemy.

Types

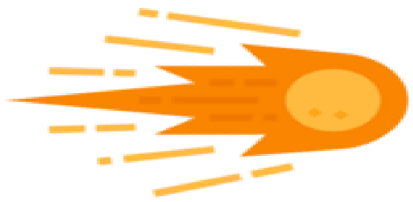


The **bullets** are called by the class Bullet() that has three arguments the x, y and the velocity, all three of them are made from the enemies attributes from whom they come from, which means if the enemy is closer and faster, the bullet will also come from closer and will be faster. The velocity is multiplied by 0.4 to slow them down a bit because with 40 frames per second if they go at the same speed as the enemies they are way too fast to dodge. The attribute fired determines when the bullet is called and when it should disappear.

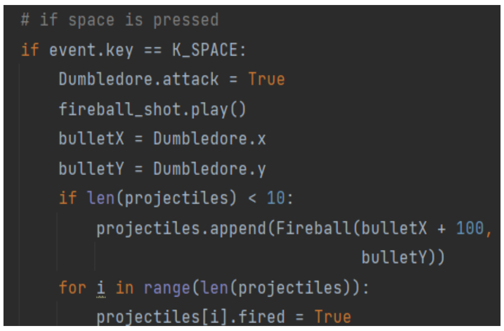


```
for bullet in bullets_enemy:
    if enemy.attack:
        bullet.fired = True
        robot_shot.play()
    bullet.draw(screen, resized_bullet)
    if not bullet.fired:
        bullets_enemy.pop(bullets_enemy.index(bullet))
```

The number of bullets is limited to 5 present on the screen at the same time, this is done by adding them into a list. Their draw functions are called each time the enemy attack and are accompanied by a sound effect, they are popped out of the list when they collide with the fireballs, the player or they get to the limit boundary which is 10.



The **fireballs** are called by the class Fireball() that has two arguments x and y that are both linked to the player's attribute, to the x value is added 100 to make an illusion of the fireballs coming out of the stick of the wizard. The velocity is constant at 8. It also has the attribute fired that determines when it is called and when it should disappear.



```
# if space is pressed
if event.key == K_SPACE:
    Dumbledore.attack = True
    fireball_shot.play()
    bulletX = Dumbledore.x
    bulletY = Dumbledore.y
    if len(projectiles) < 10:
        projectiles.append(Fireball(bulletX + 100,
                                    bulletY))
    for i in range(len(projectiles)):
        projectiles[i].fired = True
```

The number on the screen at the same time is limited to 10, I also made a list and each time the user presses the space bar one is added to it and the fireball is drawn on the screen accompanied by a sound effect. They are popped out of the list when they collide with an enemy or they get to the boundary which is 1100.

Collisions

Collisions are the point at which two objects will be in contact, this depends on the distance between the two and this distance d is defined by a math equation like the following:

$$d = \sqrt{(x2 - x1)^2 + (y2 - y1)^2}$$

The x1 and y1 are the coordinates of the first object and x2 and y2 are the coordinate of the second object. For the game, I didn't always use the actual x and y coordinate. I added or subtracted a certain number for the player and the enemy for example because the x and y coordinate are the middles of the image and the bullet doesn't need to get to the middle of the wizard to hurt him.

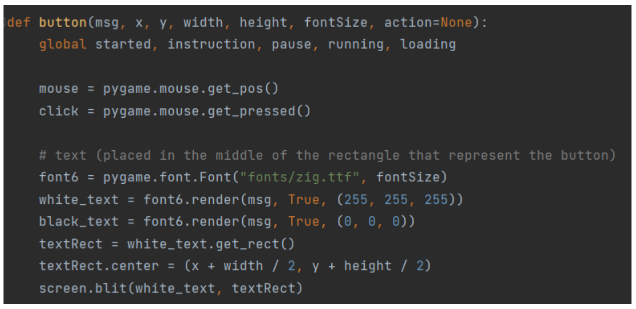
Not only can the projectiles collide with the characters but they can also collide between themselves and both disappear.

Other important functionalities

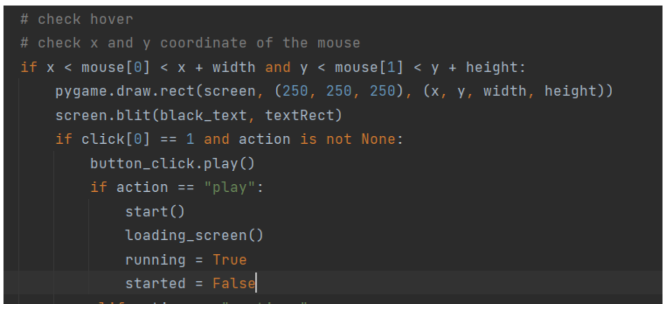
Button

When I first learnt that pygame didn't have a button built-in function I was quite shocked because buttons are necessary to game development. But then I realized that it gives us all the tools to make it any way we want. For my buttons, because I wanted them to have the same design to make some sort of consistency on to make the user know which one part can he click and it will have an effect I created a function called button(). It has seven arguments that are going to be written on it, the x and y components that determine its position, its width, its height, the font size of the message's text and the action. I made a hover effect based on the position of the mouse, if it is anywhere over the rectangle the colour of the rectangle give the illusion of it being something different from the other components of the screen. The action is the argument that determines what will the button do, it is determined by if statements that check what if the action is one of one given and then execute what is assigned to it, they are mostly based on the global variable that controls the while loop, most buttons will make one False and another True, to start and stop the while loops giving the illusion of changing screen.

Action	What will the button do
"play"	Reset all the game elements like the health and the score, then start the main game loop by first going to the loading screen
"quit"	Stop the running game and go back to the main menu
"manual"	Go to the instruction page
"continue"	Continue the running game from the moment it was paused without resetting anything
"exit"	Stop the whole program and leave the game (same as the cross on the top right corner)
"reset"	Open the high_score file and reset the value to 0.



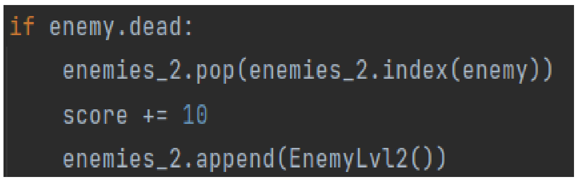
```python
def button(msg, x, y, width, height, fontSize, action=None):
    global started, instruction, pause, running, loading

    mouse = pygame.mouse.get_pos()
    click = pygame.mouse.get_pressed()

    # text (placed in the middle of the rectangle that represent the button)
    font6 = pygame.font.Font("fonts/zig.ttf", fontSize)
    white_text = font6.render(msg, True, (255, 255, 255))
    black_text = font6.render(msg, True, (0, 0, 0))
    textRect = white_text.get_rect()
    textRect.center = (x + width / 2, y + height / 2)
    screen.blit(white_text, textRect)
```



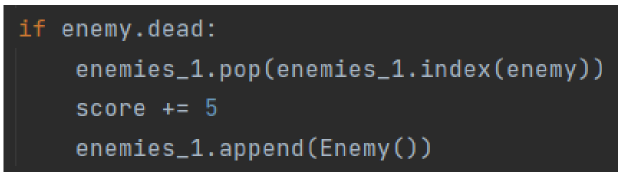
```python
# check hover
# check x and y coordinate of the mouse
if x < mouse[0] < x + width and y < mouse[1] < y + height:
    pygame.draw.rect(screen, (250, 250, 250), (x, y, width, height))
    screen.blit(black_text, textRect)
    if click[0] == 1 and action is not None:
        button_click.play()
        if action == "play":
            start()
            loading_screen()
            running = True
            started = False
```

Score

The score is displayed on the game screen all the time, it is on the top left corner of the screen. It increases by 5 if you kill a normal enemy and 10 if you kill an enemy level 2. I used a global variable score that is defined at the beginning of the game.py file and is called each it needs to be increased.



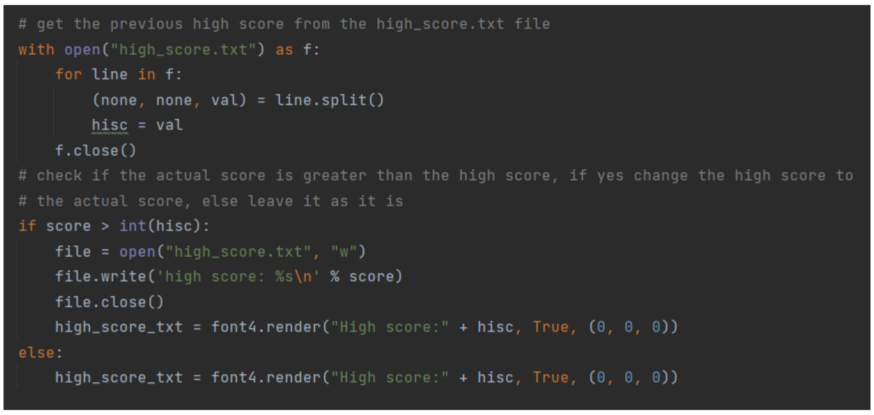
```
if enemy.dead:
    enemies_2.pop(enemies_2.index(enemy))
    score += 10
    enemies_2.append(EnemyLvl2())
```



```
if enemy.dead:
    enemies_1.pop(enemies_1.index(enemy))
    score += 5
    enemies_1.append(Enemy())
```

High score

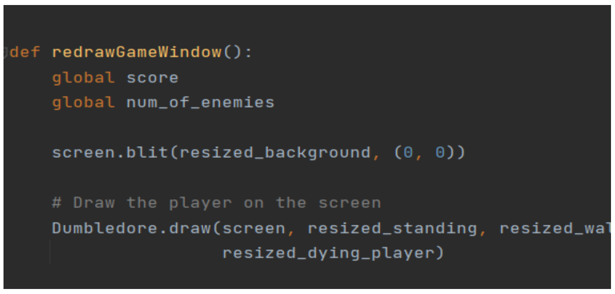
The high is saved into another file called high_score.txt that will be opened at the end of each game to check if the value is bigger than the actual score or not, if it is then the file will be rewritten with the actual score as the new value. If not, then it will keep the same value and display both the high score and the actual score. This file is also opened each time we are on the main menu, to display the previous high score and let the user know this is the score he has to beat.



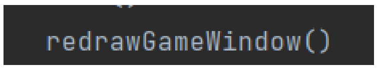
```
# get the previous high score from the high_score.txt file
with open("high_score.txt") as f:
    for line in f:
        (none, none, val) = line.split()
        hisc = val
    f.close()
# check if the actual score is greater than the high score, if yes change the high score to
# the actual score, else leave it as it is
if score > int(hisc):
    file = open("high_score.txt", "w")
    file.write('high score: %s\n' % score)
    file.close()
    high_score_txt = font4.render("High score:" + hisc, True, (0, 0, 0))
else:
    high_score_txt = font4.render("High score:" + hisc, True, (0, 0, 0))
```

Draw function

In order to make my code more organized, I decided to put all the things that are related to drawing into one function that I will call the end of the game loop because putting all the code directly into the loop made it look really long and quite mess. All the things that are related to drawing from drawing the images of the player and the fireballs to adding enemies to the lists.



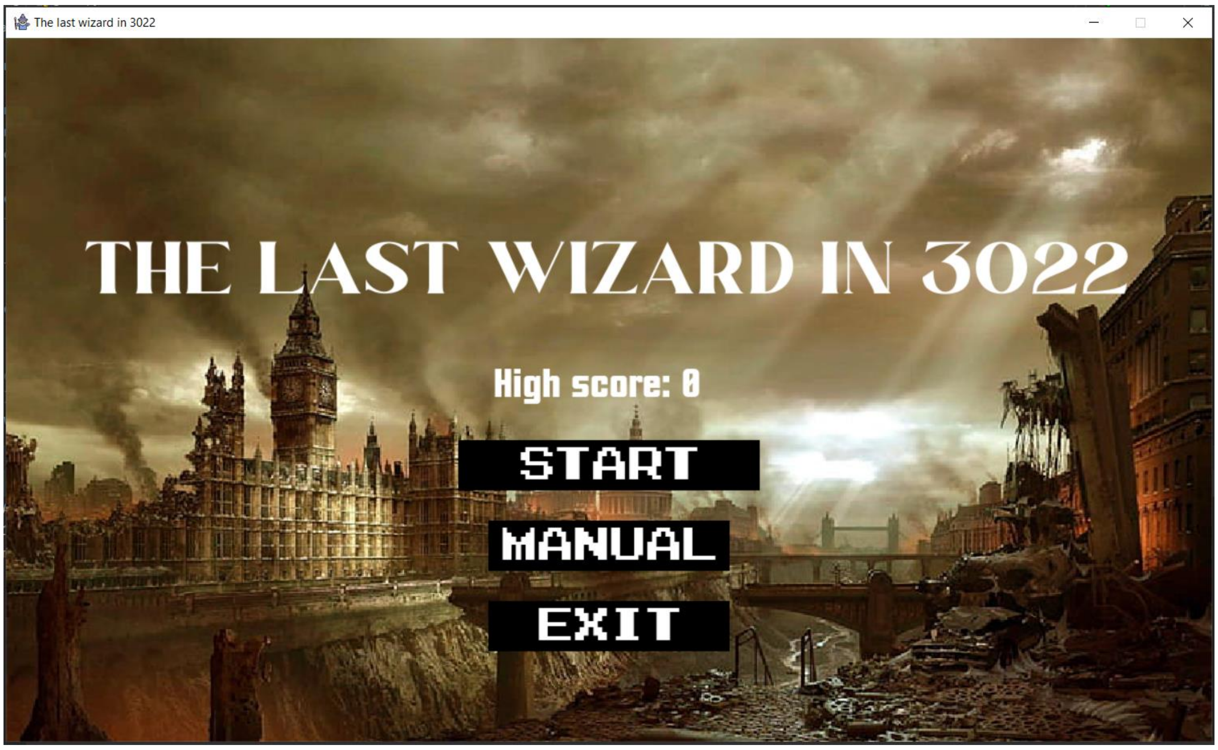
```
def redrawGameWindow():
    global score
    global num_of_enemies

    screen.blit(resized_background, (0, 0))

    # Draw the player on the screen
    Dumbledore.draw(screen, resized_standing, resized_wa
                    resized_dying_player)
```



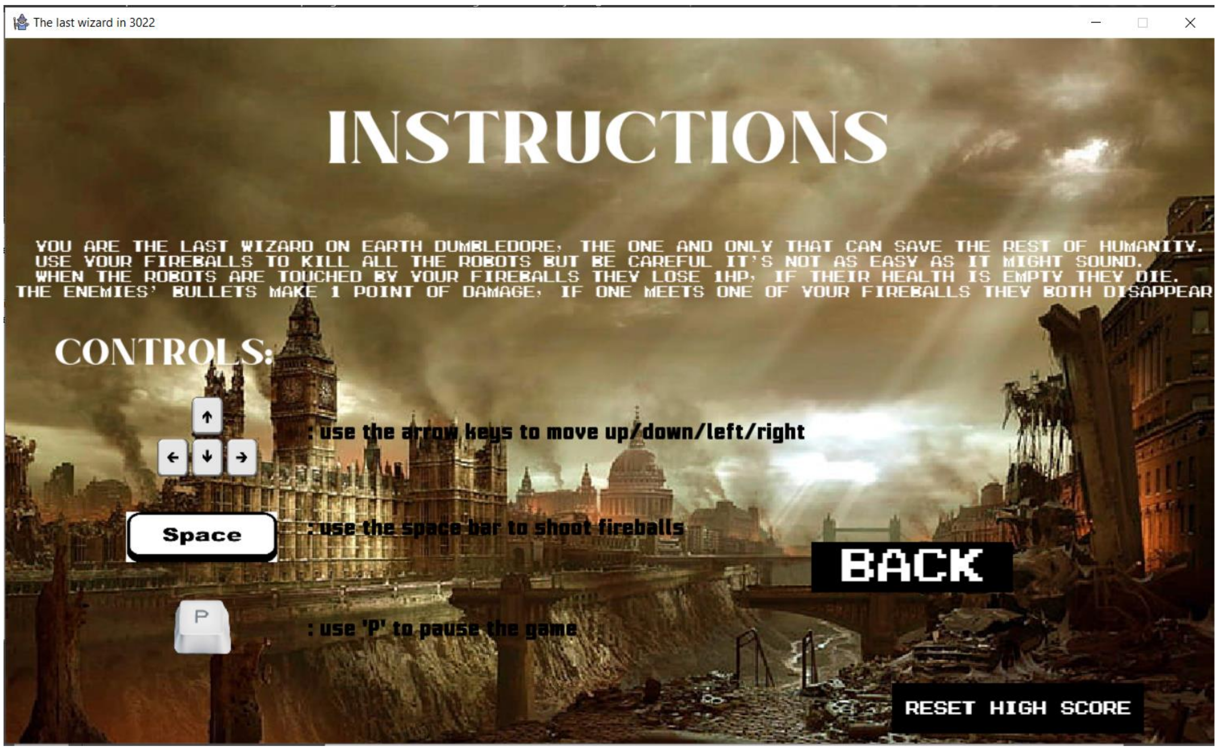
```
redrawGameWindow()
```

VI. EVIDENCE OF WORKING PROGRAM

Main menu:



Manual page:


The last wizard in 3022

INSTRUCTIONS

```
  YOU ARE THE LAST WIZARD ON EARTH DUMBLEDORE, THE ONE AND ONLY THAT CAN SAVE THE REST OF HUMANITY.
  USE YOUR FIREBALLS TO KILL ALL THE ROBOTS BUT BE CAREFUL IT'S NOT AS EASY AS IT MIGHT SOUND.
  WHEN THE ROBOTS ARE TOUCHED BY YOUR FIREBALLS THEY LOSE 1HP, IF THEIR HEALTH IS EMPTY THEY DIE.
THE ENEMIES' BULLETS MAKE 1 POINT OF DAMAGE, IF ONE MEETS ONE OF YOUR FIREBALLS THEY BOTH DISAPPEAR
```

CONTROLS:

: use the arrow keys to move up/down/left/right

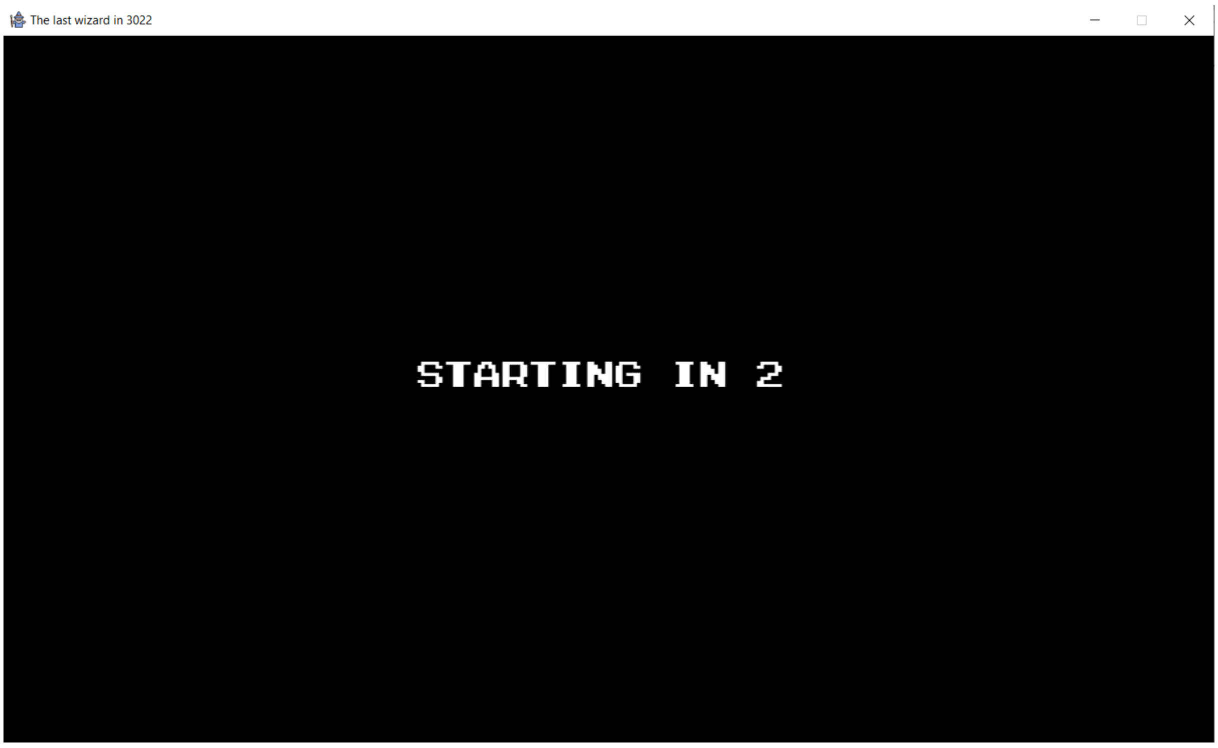
Space : use the space bar to shoot fireballs

P : use 'P' to pause the game

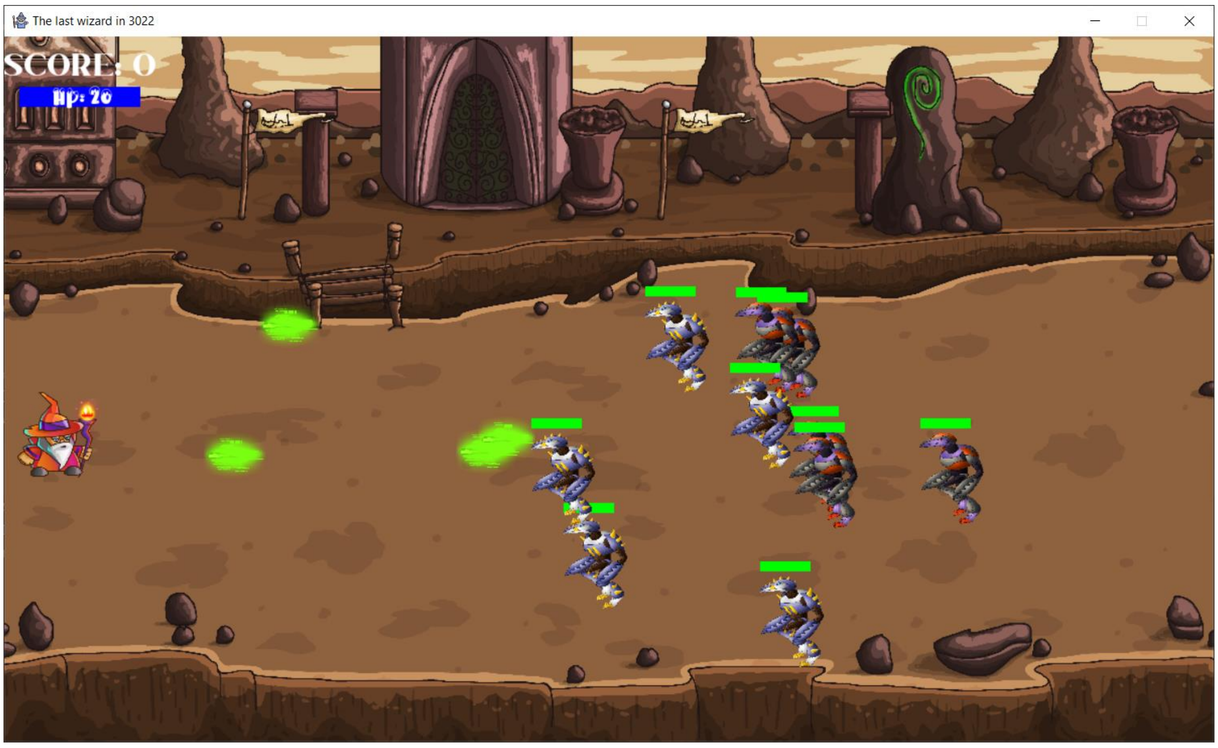
BACK

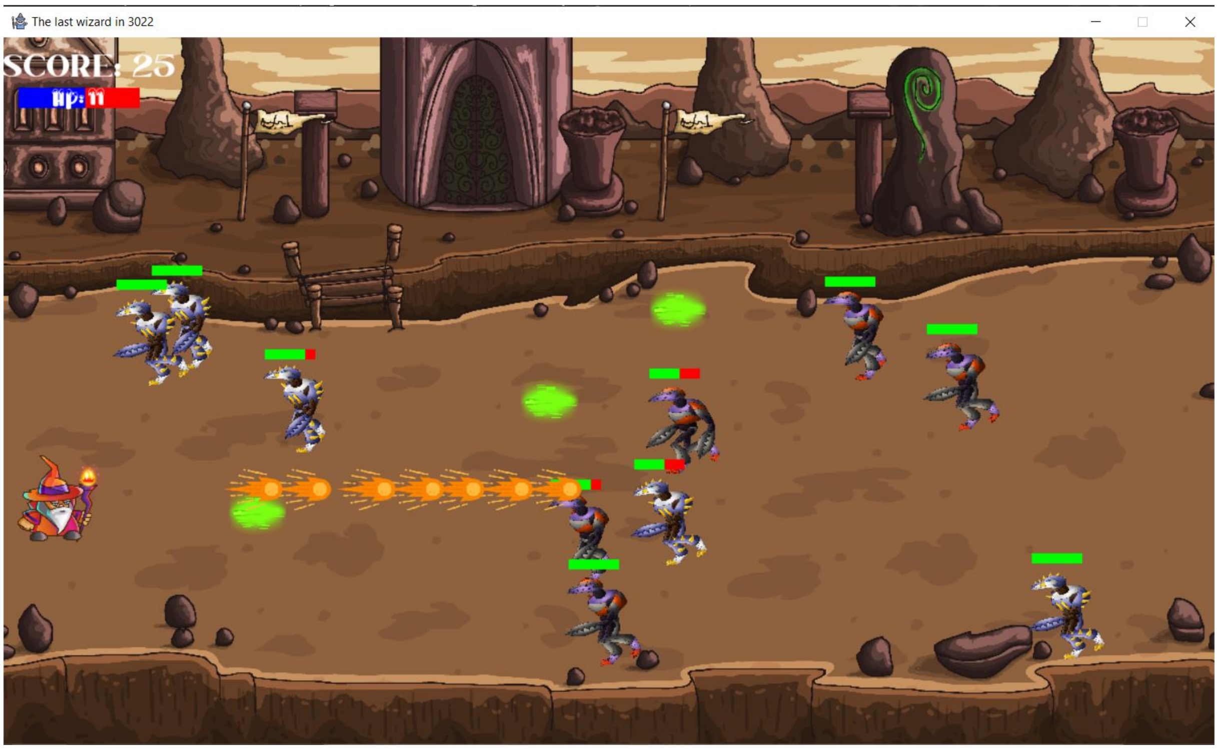
RESET HIGH SCORE

Loading screen:

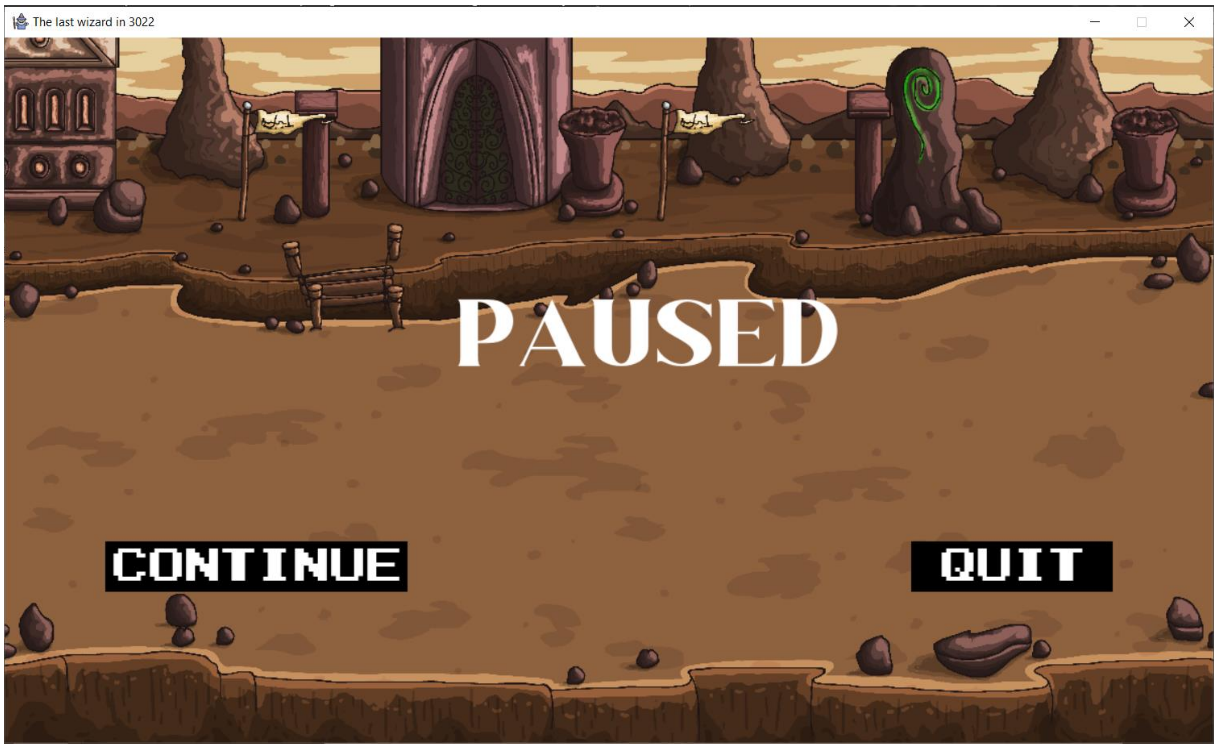


Game screen:

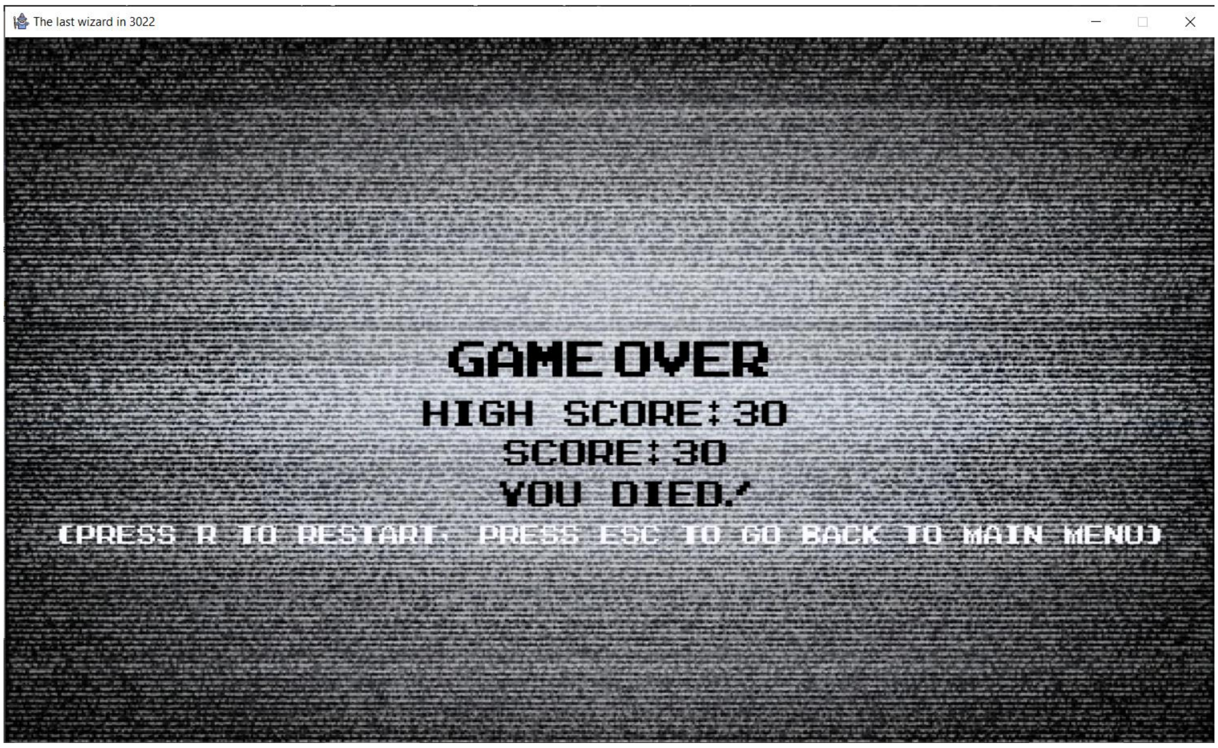




Pause screen:



End screen:


The last wizard in 3022

GAME OVER

HIGH SCORE:30

SCORE:30

YOU DIED.'

(PRESS R TO RESTART; PRESS ESC TO GO BACK TO MAIN MENU)

VII. POSSIBLE EXTENSIONS AND FUTURE AMELIORATION

At the end of the process of making the game, I had a few ideas that could make it better in the future and I judged it important to list them here.

The game could have multiple difficulties with enemies of different speeds, health and bullet damage. Also, the enemies could start randomly more the y axis as well. The implementation of different combos like a bomb that you get when you get to a certain score and maybe will have some area damage to kill multiple enemies at the same time. And also some bonuses to get some health back.

VIII. Sources of websites used for the making of the game

https://craftpix.net/?s=game+over&category=71

https://www.resizepixel.com/?new=true

https://www.1001fonts.com/video-game-fonts.html?page=3

https://www.flaticon.com/

https://www.rapidtables.com/web/color/RGB_Color.html

https://freesound.org/

https://mixkit.co/free-sound-effects/

https://www.bensound.com/royalty-free-music/cinematic