INFO1111: Computing 1A Professionalism

2023 Semester 1

Self-Learning Report

Submission number: 3

Github link: `https://github.com/zliu6203/SelfLearning`

Student name	Zhili Liu
Student ID	520451407
Topic	PyGame
Levels already achieved	B
Levels in this report	D

Contents

Instructions

Important: This section should be removed prior to submission.

You should use this LaTeX template to generate your self-learning report. Keep in mind the following key points:

- **Submissions**: There will be three opportunities during the semester to submit this report. For each submission you can attempt 1 or 2 levels. Each submission should use the same report, but amended to include new information.
- **Assessment**: In order to achieve level B, you must first have achieved level A, and so on for each level up to level D. This means that we will not assess a higher level until a lower level has been achieved (though we will review one level higher and give you feedback to help you in refining your work).
- **Minimum requirement**: Remember that in order to pass the unit, you must achieve at least level A in the self-learning (unless you achieve level B in both the skills and knowledge categories).
- **Using this template**: When completing each section you should remove the explanation text and replace it with your material.
- **Referencing**: You should also ensure that any resources you use are suitably referenced, and references are included into the reference list at the end of this document. You should use the IEEE reference style [**?**] (the reference included here shows you how this can be easily achieved).

1. Level A: Initial Understanding

1.1. Level A Demonstration

The three things that demonstrate my understanding for this topic are:

- Make a successful window for PyGame including some screen elements (sprites).
- Using sound libraries.
- Responding to user input.

1.2. Learning Approach

I approached my learning by looking for the latest tutorials online with PyGame, specifically the ones with project examples. I browsed both video and website tutorials including official documentation for many new methods and modules to build a successful game using the PyGame module of Python.

Experimentation was my main method of learning; testing all the different built-in libraries of methods and objects including sprites ("draw" module), sounds ("mixer" module), and the window itself ("display" module). I self-learnt by using previous knowledge of game development in C# and Java to my advantage - scenes, object priority, using external files etc. Additionally, I also used knowledge of window composition via WinForms and Tkinter.

1.3. Challenges and Difficulties

Due to the new syntax it was difficult to correctly load a window at all with the limited knowledge. The most difficult to learn was the while loop that the game is running on; it also contains a for loop which did not make much sense. I simply accepted it as fact instead of fully understanding it.

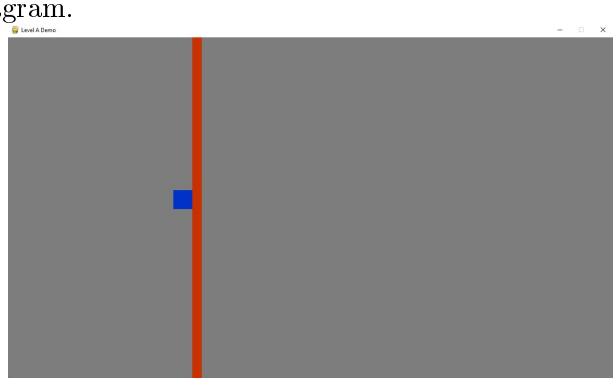
Even after the window was created, it was even more difficult to find any sprites to fill the window as opposed to Tkinter or WinForms, where sprites and screen elements were ready for use. Furthermore, PyGame did not include any built-in sound files and so sound effects & music must be found online or created locally.

1.4. Learning Sources

Learning Source	Contribution to Learning
Wiki tutorial	Understanding what each module does and its methods and objects.
Beginner video tutorial	Understanding the basics of every major aspect of PyGame.
Official documentation (mouse input)	Finding each method and the parameters necessary for these methods.
Sound help	Understanding how sound works and how to implement sound at an appropriate time.
Another video tutorial for shapes	Understanding what to do in the absence of assets with sprites.

1.5. Application artifacts

In level A, I made a very simple program in PyGame demonstrating that a successful window (1280x720) is opened with some screen elements (background, red line, blue player square). The blue square is responding to user input by following the coordinates of the mouse and not allow the mouse to go past the red line or the screen. When the cube touches the red line, a sound effect is played exactly once. This is demonstrated in this diagram.



In the code, I used simple python syntax with some new PyGame methods and elements. In the follwing screengrabs, I showed this by using the main function and setting initial variables.

```python
import pygame
import os

pygame.init()
pygame.mixer.init()

# initialise some values
WIDTH, HEIGHT = 1280, 720
WIN = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Level A Demo")
FPS = 60

BG_COLOUR = (125, 125, 125)

def draw(timer):
    # Draws the scene frame by frame

    # get the mouse position
    MOUSE_X, MOUSE_Y = pygame.mouse.get_pos()

    if MOUSE_X > 370:
        MOUSE_X = 370
        SOUND1 = pygame.mixer.Sound(os.path.join('sound_effect1.wav'))
        if timer == 0:
            pygame.mixer.Sound.play(SOUND1)
    # if the mouse passes the red line, trigger sound effect
```

(a) Code showing variables

```python
def main():
    clock = pygame.time.Clock()
    running = True
    k = 0
    # this allows the program to run continuously
    while running:
        clock.tick(FPS)
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False

        # for the timing of the sound effect
        if pygame.mouse.get_pos()[0] > 370:
            k += 1
        else:
            k = 9999999999999
        draw(k % 10000000000000)

    # once the program stops running, quit program
    pygame.quit()

if __name__ == "__main__":
    main()
```

(b) Code showing main function

Additionally, I used a simple sound effect from the web (sound_effect1.wav) to play whenever the player hits the red line. I also used pyinstaller to make the .py into a .exe which is also included in the Git repository.

2. Level B: Basic Application

Whilst level A is about doing something simple with the topic to just show that you have started to be able to use the tool or technology, level B is about doing something practical that might actually be useful.

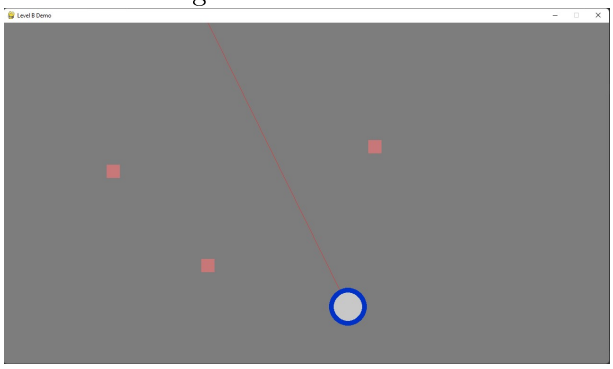
2.1. Level B Demonstration

The application I have developed is a simple game in PyGame. This involves the three points in the level B proposal: events such as scenes, making a simple game with an end goal (win/lose), and program simple enemies. I have demonstrated the first point by making different levels in the game, satisfied the second point by creating a game with a goal (to finish all 5 levels), and also programmed simple enemies with random movement and bouncing back from the screen if it goes offscreen.

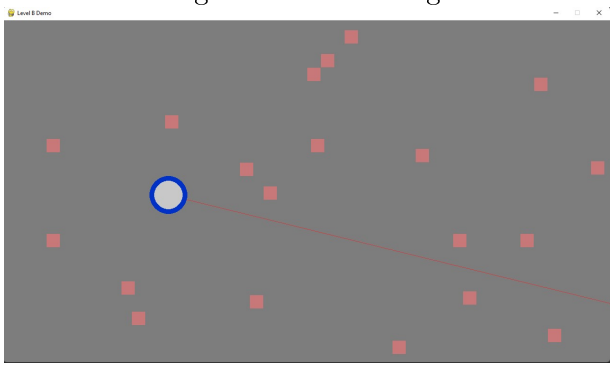
2.2. Application artifacts

I have created a simple topdown shooter game using built-in assets, modules, and tools only. The game I have made is simple: shoot all enemies using a periodic laser which can destroy all enemies collided with the laser. I used classes and small aspects of object oriented programming to achieve this.

In the following screengrab, the player can shoot a laser which only fires when the left mouse button is pressed and the cooldown for the laser is done. This laser collides with all enemies within its vicinity and destroys them. The player in blue and white can be controlled using WASD and the laser is shot in the direction of the cursor.



This screengrab shows the huge number of enemies in level 4 of this game:



This code snippet shows the object oriented programming and the hierarchy of classes and objects. The Entity class includes both the Player and Enemy classes as its children. The enemy class also contains code for random movement (see more in GitHub).

```python
class Entity:

    def __init__(self, x, y, speed):
        self.x = x
        self.y = y
        self.speed = speed

    def move_up(self):
        self.y -= self.speed

    def move_down(self):
        self.y += self.speed

    def move_left(self):
        self.x -= self.speed

    def move_right(self):
        self.x += self.speed


class Player(Entity):

    def display(self):
        pygame.draw.circle(WIN, (0, 50, 200), (self.x, self.y), 40)
        pygame.draw.circle(WIN, (200, 200, 200), (self.x, self.y), 30)


class Enemy(Entity):

    def __init__(self, x, y):
        super().__init__(x, y, 1)
        self.direction = "down"

    def display(self):
```

These two code snippets shows the Game class being the main class which handles all the events.

```python
class Game:

    LASER_COLOUR = 125, 125, 125
    def __init__(self, level):
        self.enemies = []
        for coord in level:
            self.enemies.append(Enemy(coord[0], coord[1]))
        self.player = Player(640, 360, 4)
        self.firing = False
        self.LASER_COLOUR = 125, 125, 125
        self.timer = 0


    def draw(self):
        # Draws the scene frame by frame

        WIN.fill(BG_COLOUR)
        MOUSE_X, MOUSE_Y = pygame.mouse.get_pos()

        is_left = pygame.mouse.get_pressed()[0] # is left click

        self.firing = is_left and self.timer % 40 in range(0, 14)

        if self.firing:
            LASER_COLOUR = 210, 60, 60
        else:
            LASER_COLOUR = 125, 125, 125

        pygame.draw.line(WIN, LASER_COLOUR, (self.player.x, self.player.y),
                         (512 * (MOUSE_X - self.player.x),
                          512 * (MOUSE_Y - self.player.y)))


        self.player.display()
```

(a) Code showing Game class

```python
        laser_coords = [(20/i * (MOUSE_X - self.player.x) + self.player.x,
                         20/i * (MOUSE_Y - self.player.y) + self.player.y) for i in range(1, 200)]


        for e in self.enemies:
            if self.firing:
                for coords in laser_coords:
                    if abs(coords[0] - e.x) < 19 and abs(coords[1] - e.y) < 19:
                        try:
                            self.enemies.remove(e)
                        except ValueError:
                            print("Error handler")

            if self.timer % 50 == 0 or e.x < 1 or e.y < 1 or e.x > 1251 or e.y > 691:
                e.change_dir()
            e.display()


        ke = pygame.key.get_pressed()
        if ke[pygame.K_a]:
            self.player.move_left()
        if ke[pygame.K_w]:
            self.player.move_up()
        if ke[pygame.K_d]:
            self.player.move_right()
        if ke[pygame.K_s]:
            self.player.move_down()

        self.timer += 1

        pygame.display.update()
        print(len(self.enemies))
        return len(self.enemies)
```

(b) Code showing how enemies work

3. Level C: Deeper Understanding

Level C focuses on showing that you have actually understood the tool or technology at a relatively advanced level. You will need to compare it to alternatives, identifying key strengths and weaknesses, and the areas where this tool is most effective.

3.1. Strengths

Python is an easy language to learn and PyGame is very lightweight (i.e., takes up little memory). PyGame can be a valuable introduction to game development as it allows beginners to learn the fundamentals of graphics programming, manipulating the GUI, interactivity, using sound and music libraries, and many more. Furthermore, PyGame excels at implementing simple functionalities using much less code than alternatives such as Unity or Unreal Engine, for example collisions and player input [**?**]. The PyGame community has plenty of sprites and sound files to choose from so the programmer does not need to make them from scratch.

3.2. Weaknesses

PyGame contains limited features as well as lacking in speed when compared to alternatives. The Python language is generally much slower and more minimalistic than other C based languages including Java, C#, and C++, the latter two are what Unity and Unreal Engine are based on. As such, PyGame syntax can be very confusing for more complex mechanics such as firing projectiles or implementing physics. PyGame is also "semi-portable" as distribution onto different platforms requires a native version of PyGame [**?**], in addition to being difficult to distribute due to Python's syntax, which are major inconveniences.

3.3. Usefulness

PyGame is useful for developing small games in a short period of time as Python is very efficient at executing simple tasks [**?**]. One scenario is where an experienced game programmer or team comes up with an original idea for a game; instead of building the game directly, they can use PyGame to create a fast and simple design (as a planning phase) to spot potential loopholes and development issues. PyGame can be used as a tool for designing and testing more advanced game ideas using other engines simply due to how rapid the development can be.

3.4. Key Question 1 - Is PyGame useful for 3D First Person games? Or only 2D games?

PyGame itself is only useful for 2D games due to a lack of physics engines, rendering software, and using 3D models and environments. PyGame can handle basic 3D graphics but that is done with incredibly long and complex code, while also being very slow. Furthermore, as discussed earlier, it is difficult to distribute the completed software, making it even less ideal to package 3D games. However, with the introduction of other modules such as PyOpenSQL and PyGlet to form PyEngine3D [**?**], it is much more viable to create simple 3D games, many of which can be found in the community. However, it is observed that very few are first person 3D games, which shows the complexity of this task.

3.5. Key Question 2 - How good a Python programmer do you need to be before you can use PyGame effectively?

PyGame, much like every other game engine, require a solid knowledge of objects, methods, modules and data from the parent language (Python) to create effective games. High-level control systems, such as the ones mentioned before, are needed to allow the developed game to be interactive and responsive to input. An understanding of classes

and methods allows manipulation of one group of objects and easy replication of features (e.g., enemy or game structure). Moreover, other modules and using data is important for distribution of software, allow variability from other projects (promotes original ideas), and most importantly saving information to a local or online database for future access.[?]

4. Level D: Evolution of skills

4.1. Level D Demonstration

I have developed a simple top-down shooter game where the goal is to clear a level of all enemies before moving on. In total there are 5 levels and 5 different types of enemies with their own unique abilities. The player controls a laser (space or mouse click) which can decrease the health of any enemies in contact with them. The game contains a main menu, as well as screens for victory, game over and level clear (see below). To run the program from GitHub, run the "level_D.exe" file provided (does not work with Mac).

4.2. Application artifacts

The actual application developed from PyGame is a .exe file generated from "auto-py-to-exe", much like level A:

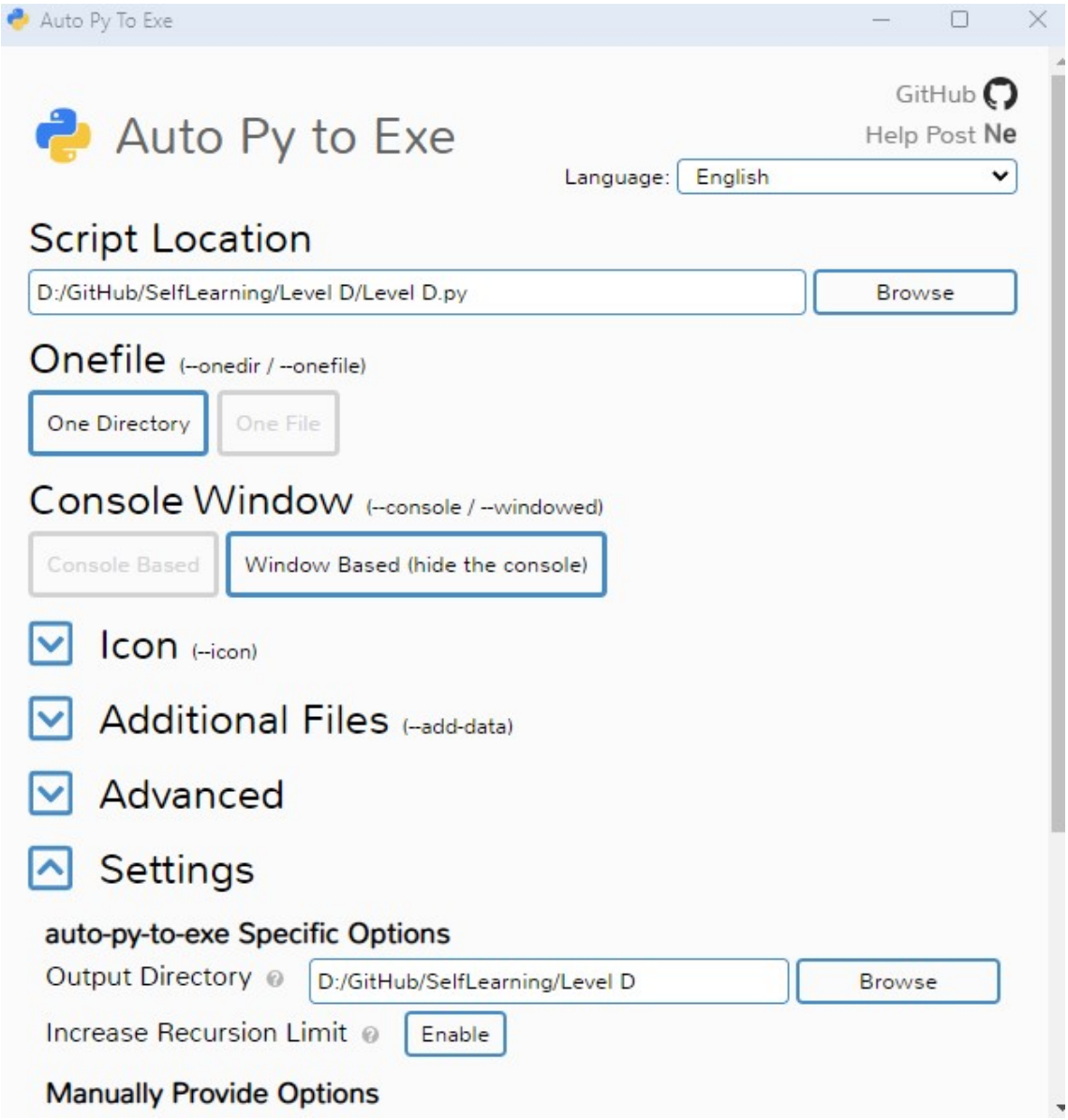


Figure 3: Using level D python file and the directory contents to convert to exe file.

When opened, the game greets the player with a menu screen, generated from the Menu class. This is always the first thing executed when main() is run, which makes it useful for replaying a game-over or victory:

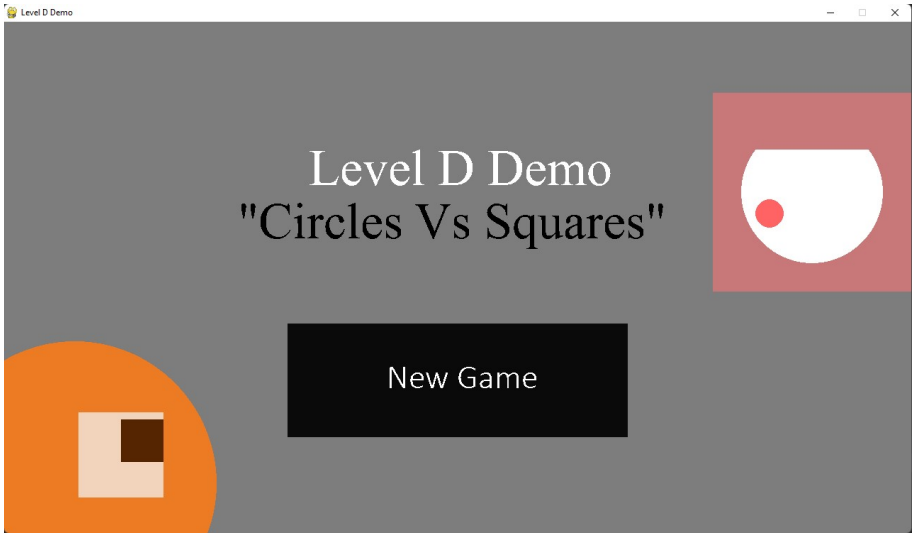


Figure 4: Main menu, "New Game" button turns gray when highlighted.

```
63  class Menu:
64
65      def __init__(self):
66          pygame.mixer.music.load(f'menu.wav')
67          pygame.mixer.music.play(-1)
68          self.button_col = 10, 10, 10
69
70      def draw(self):
71          WIN.fill(BG_COLOUR)
72          MOUSE_X, MOUSE_Y = pygame.mouse.get_pos()
73
74          pygame.draw.rect(WIN, self.button_col, (400, 425, 480, 160))
75
76          menu_font = pygame.font.SysFont('Times New Roman', 72)
77          button_font = pygame.font.SysFont('Calibri', 48)
78          title1 = menu_font.render('Level D Demo', False, "White")
79          title2 = menu_font.render('"Circles Vs Squares"', False, "Black")
80          title3 = button_font.render('New Game', False, "White")
81          WIN.blit(title1, (430, 165))
82          WIN.blit(title2, (330, 240))
83          WIN.blit(title3, (540, 480))
```

Figure 5: Menu class

Furthermore, the user will also experience multiple music tracks and sound effects in the playthrough. These sounds are used from freesound.org [**?**] as they are copyright free. How the music and SFX were implemented was based solely off of the official PyGame docs website [**?**]. Some sounds were manipulated in FL Studio for EQ - making the game volume consistent as some sounds were much louder than others.

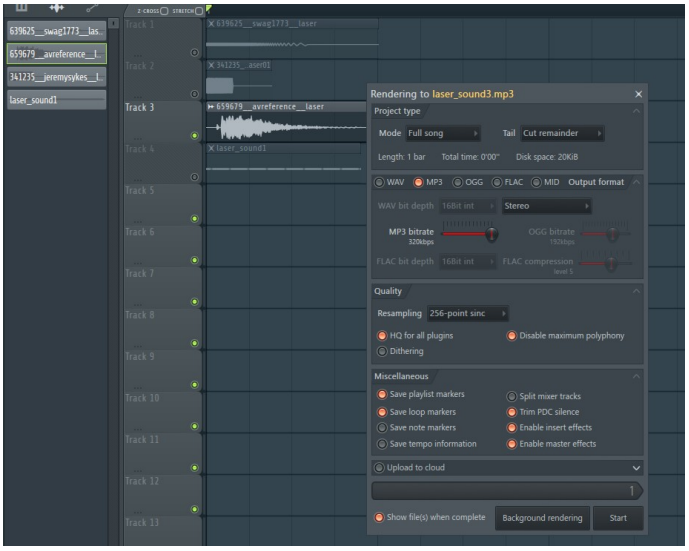


Figure 6: The use of FL Studio to balance gain or volume of music/sound effects.

```
140     def draw(self):
141         # Draws the scene frame by frame
142         WIN.fill(BG_COLOUR)
143         MOUSE_X, MOUSE_Y = pygame.mouse.get_pos()
144         ke = pygame.key.get_pressed()
145         shooting = pygame.mouse.get_pressed()[0] or ke[pygame.K_SPACE]
146         fx = pygame.mixer.Sound(f"laser_sound{random.randint(1, 4)}.mp3")
147         self.firing = shooting and self.timer % 40 in range(0, 14)
148 
149         if self.firing:
150             if self.timer % 40 in range(0, 1):
151                 pygame.mixer.Sound.play(fx)
152             LASER_COLOUR = 210, 60, 60
153         else:
154             LASER_COLOUR = 125, 125, 125
```

Figure 7: Implementation of the randomised shooting sound heard in-game.

When the game starts, every level contains its own music and "waves", which are set swarms of enemies defined by multiple csv files (comma separated values). The numbers shown in figure 8 represents how many of a particular type of enemy is to be spawned. Each row is one wave of enemy, so level 5 has 7 waves of enemies. Note that these csv files only define the number and types of enemies spawned in a wave/level and not their spawn location. The spawn location is instead defined by the function "enemy_spawn" which returns a list of random coordinates outside of the screen. Enemies move much fast and are invincible outside the screen (figure 9).

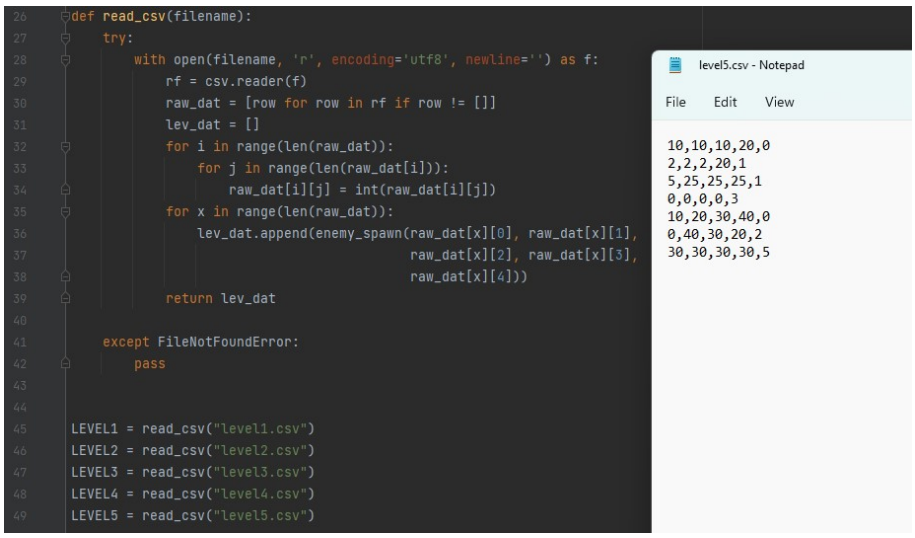


Figure 8: Csv module in Python and working with csv files.

Figure 9: Function which randomises enemy spawn coordinates.

There are five different enemies implemented:

- NormalEnemy - enemy with average health and speed (green)
- FastEnemy - enemy with low health and fast speed (red)
- TankyEnemy - enemy with high health and low speed (blue)
- RegenEnemy - enemy with average health and speed but regenerates health (magenta)
- BossEnemy - enemy which has the properties of all enemies (i.e., fast, tanky, and regenerates health) (black)

Each enemy has their own class which are children of the Enemy class, which is in turn a child of the Entity class. This class structure was explained level B. Each enemy's unique sprites were made by trial and error - adjusting the sprite depending on observation in game.

Enemies show their health bar when their health is not full. This also applies for the player. BossEnemy removes more health than all other enemies, making them dangerous. All classes and objects (except Game and Menu) are defined in another file called "objects.py".

Some gameplay and code for enemies are shown:

```
class NormalEnemy(Enemy):
    def __init__(self, x, y):
        super().__init__(x, y)
        self.s = random.randint(5, 13) / 10
        self.maxhealth = random.randint(28, 38)
        self.health = self.maxhealth

    def display(self):
        pygame.draw.rect(WIN, (120, 200, 120), (self.x, self.y, 28, 28))
        if self.x <= -200 or self.x >= 1480 or self.y <= -200 or self.y >= 920:
            self.speed = 5
        else:
            self.speed = self.s
        if 0 < self.x + 14 < 1280 and 0 < self.y + 14 < 720:
            pygame.draw.circle(WIN, "white", (self.x + 14, self.y + 14), 10)
            pygame.draw.circle(WIN, (30, 200, 30), (self.x + 14, self.y + 14), 3)

        if self.health < self.maxhealth:
            self.show_healthbar(self.health, self.maxhealth)
```

Figure 10: Class NormalEnemy has all properties of the green enemy

```
class RegenEnemy(Enemy):
    def __init__(self, x, y):
        super().__init__(x, y)
        self.s = random.randint(9, 18) / 10
        self.maxhealth = random.randint(50, 80)
        self.health = self.maxhealth
        self.timer = 0

    def display(self):
        self.timer += 1
        if self.x <= -200 or self.x >= 1480 or self.y <= -200 or self.y >= 920:
            self.speed = 5
        else:
            self.speed = self.s
        pygame.draw.rect(WIN, (176, 63, 144), (self.x, self.y, 28, 28))
        if 0 < self.x + 8 < 1280 and 0 < self.y + 8 < 720:
            pygame.draw.circle(WIN, (204, 161, 192), (self.x + 14, self.y + 14), 10)
            eye_font = pygame.font.SysFont('Arial', 30)
            eye_surface = eye_font.render(f'X', False, "Red")
            WIN.blit(eye_surface, (self.x + 6, self.y - 2))

        if self.health < self.maxhealth:
            self.show_healthbar(self.health, self.maxhealth)
            if self.timer % 2:
                self.health += 1
```

Figure 11: Class RegenEnemy allows the enemy to possess regeneration.

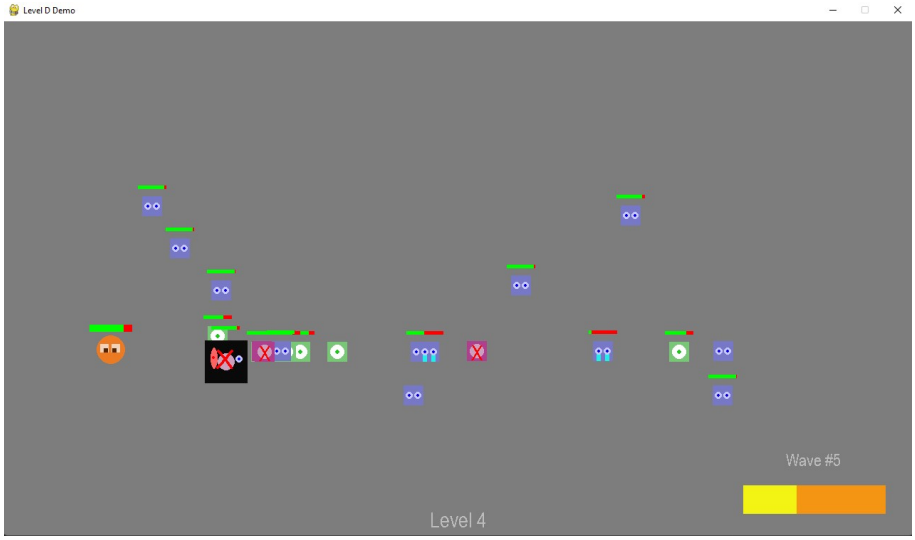


Figure 12: Screenshot of TankyEnemy (blue), RegenEnemy (magenta) and BossEnemy (black).

4.3. Alternative tools/technologies

Unreal Engine and Unity

4.4. Comparative Analysis

Unity is both a 3D and 2D game engine capable of producing high quality end-products. Like Unreal Engine, it is a multi-platform tool written in C languages. PyGame does not natively support 3D engines, but 2D games are much easier to conceptualise and develop in PyGame or Unity due to the simple tool set available (only python is needed to learn PyGame whereas extensive software knowledge is needed for both engines). As such, PyGame excels at designing quick-and-easy solutions, in addition to less computing power and memory/storage.[**?**]

However, PyGame lacks the continual community support and frequent updates unlike Unreal Engine, which, despite its complexity, generates much better graphics and rendering technologies compared to even Unity. Unreal Engine allows development of quality games commonly seen in the commercial market, whereas Unity sees less due to less advanced graphics handling.[**?**]

Ultimately, efficiency and simplicity is PyGame's best features, but it lacks complexity and 3D handling; Unreal Engine contain high quality features but it is difficult to learn and usually requires teams of programmers/graphic designers; Unity also contain high quality features AND workable alone, but it is relatively difficult to learn compared to PyGame and less graphic-intensive than Unreal Engine.