**Introduction**

This reports details the production of a top down shooting game, ZOMBRENA, using the simplegui library. It shows the technical implementation of Vector class, collisions, sprite sheets, timers etc., including the challenges faced and solutions solved throughout the development.

**Description**

The main objective of the game is to survive every wave.

Each zombie has to be hit 3 times in order to be killed; however, except for orbital lazer, revolver and lazer rifle that kill the zombies instantly. The number of zombies killed is recorded at the top of the screen. At every even number wave, there will be a zombie boss and it has special abilities such as shooting back, charging towards the player and so on. If the player survives 10 waves, the player wins.

After every wave, zombies will be spawned more. And if the player gets hit 3 times, it’s game over. In addition, there are two types of zombies: regular zombies and underground zombies. For regular zombies, they will chase the player around the map with constant speed, whereas for underground ones, they will pop out randomly from the floor. If they pop out on top of the player, one of the player’s lives is lost. Advice: do NOT pick up the gun that’s not in the middle.

**User Manual**

A screen shot of a video game

Description automatically generatedIn order to play the game, the game file should be downloaded, including sprite sheets and sounds. Open the files in an IDE such as PyCharm and check the terminal whether simplegui is installed or not. If not, type “pip3 install SimpleGUICS2Pygame” in the command prompt. After that, when the game file is opened, it should run properly.

Home Screen 1

Home Screen

As soon as the program is run, the player will be welcomed with the home screen that includes the name of the game and the instruction to the next step as shown above.

A screenshot of a video game

Description automatically generated

Story

After the space key is pressed, the player is informed about the background story regarding the main character in the game.

A screenshot of a video game

Description automatically generated

Instructions

The story slide is followed by an instruction slide, where the user can learn how to play the game. To control the player, as per the instructions page, ‘a’, ‘s’, ‘w’, ‘d’, which represents left, down, up and right respectively, have to be used. Pressing space with one of the movement keys makes the player’s movement faster. The gun is activated when a mouse is clicked on the screen.

A screenshot of a video game

Description automatically generated

Guns

After “Next Page” is clicked, the gun book will be shown which includes all the guns that can be available throughout the game and each gun is presented with the level of damage: LOW, MEDIUM, VERY HIGH.

A screenshot of a video game

Description automatically generated

Game Play

The game starts after “Let’s Go!” is clicked. When it starts, the player will get a regular pistol, and there will be a gun randomly chosen and put in the middle of the screen. In order to pick it up, the player has to be above the gun for around 2 seconds. When the player gets hit by a zombie, one health is reduced. After 3 hits, the player dies. The current wave and the number of zombies killed are shown on the top left corner of the screen, and the health is on the right. The image on left corner represents the gun that the player is currently holding.

After 30 seconds, one wave is finished. To move on to the next wave, ‘Next’ button should be clicked.

A grey rectangular sign with black text

Description automatically generated

Next Button

In addition, if the player wants to restart the game, ‘ESC’ should be pressed and it sends the player back to the main screen. It’s also true when the player dies; the player will be instructed to go back to the home screen. During each even wave, there’s a zombie boss with unique abilities that has more health than normal zombies. There will be a health bar at the bottom right corner for the bosses when they are spawned. In 2nd wave, the tank zombie will chase around the player just like normal zombie. In 4th wave, there’s a charger zombie, that will move towards the player much faster than normal zombies. In 6th wave, there will be a scientist zombie that will run away from the player while shooting orbital lazer, which can be fatal with only one hit sometimes. In 8th wave, both the zombie bosses from 2nd and 4th waves will be spawned. Finally, the big boss has two giant hands moving around the screen, whereas the head will shoot lazer beans directly in the middle at some point. If the player gets hit by any part of the zombie, the player loses a life. The general flow of the game is shown below.

A diagram of a diagram

Description automatically generated

General Flow of the Game

**Key Concepts**

SimpleGUI is a module for creating interactive python programs using canvas, mouse clicks and keyboard inputs. Below is an example to create a canvas (WIDTH and HEIGHT are 700).





For mouse clicks, simply put after creating the frame with a method which includes some actions that should be performed after a mouse click is registered.

To include sprite sheets in the game, the sample program in Additional Resources on Moodle were used.

A screenshot of a computer code

Description automatically generated

draw\_floor method does all the necessary calculations for the sprite including where the sprite should be shown, and draw method draws the sprite on the canvas.

A computer code with text

Description automatically generated

Regarding dealing with the Vector class, one instance is that the direction of underground zombie is generated using Vector class, and the velocity is normalized first before multiplying with the speed of the zombie.

**Code Description**

GameWalls.py

**A grid of numbers and symbols

Description automatically generated**

**A computer screen shot of text

Description automatically generated**

**A screenshot of a computer code

Description automatically generatedA screenshot of a computer code

Description automatically generated**

**A screenshot of a computer code

Description automatically generated**

*Level* class takes two arguments. One is LEVEL\_GRID as shown above, and the other one is the broken object of *BrokenGlass* class.

hit\_count records the value for each cell, and is initially set to 0. In try\_hit method, it checks if the value is more than or equal to 3, and if it’s wall, it returns true. In draw\_cell method, if it’s true, it will change the sprite sheet to broken glass, and then make a sound.

\_init\_frames initializes the specific grid of the wall with the sprite sheet, since the sprite sheet is just a regular image, it’s necessary to associate each grid with some parts of the imageA screenshot of a grid

Description automatically generated as shown above.

Frame Association

The sound value is recorded in sound\_effect, which is a 2D list and is initialized to 0 at first. However, if the wall is broken and that cell hasn’t made a sound yet, Smashing\_Glass.ogg is played, and the value is changed to 1, which means the sound has been played.

A screenshot of a computer program

Description automatically generatedA screenshot of a computer program

Description automatically generatedGameMain.py

In click method, first x1, y1, x2, y2 are initialized. If showStory is true, and then if the user clicks between the coordinates, showInstructions becomes true, pos is (0,0) and terminates the method by return. The same process applies to showInstructions and showGunBook, finally, the user can shoot the gun when the game starts.

A screen shot of a computer code

Description automatically generated

try\_move\_player is a method that controls player’s movement. It adds the Vector argument with the player’s position, but if the player hits the wall or the overlay is being show, they player can’t be moved to avoid picking up guns.

A screen shot of a computer code

Description automatically generated

As mentioned above, the value of hit\_count increases as the zombies the wall. However, after each round, the walls are restored, so the value is re-set to 0.

ChargerBoss.py

A screenshot of a computer program

Description automatically generated

The piece of code shown above is the charge function is ChargerBoss.py that controls the direction of the charger boss. First, it gets the previous position of the player, then it calculates the direction by subtracting it with it’s own position. The direction\_vector is normalized. Before charging, it checks whether the length of direction\_vector is greater than the distance\_threshold, and if the condition is satisfied, the boss moves and charging becomes true.

**A moment of reflection**

During the development stage, our group exhibited strong teamwork. I was lucky enough to be grouped with hardworking teammates and each member contributed diligently. Before developing the game, we created a plan in which there were some tasks for everyone to achieve each week to ensure steady progress. We started implementing the game after the third milestone sheet.

I chose games instead of robots because of my limited proficiency in Python. I was worried a lot about me not being able to keep up with the progress. The complexity of the sample game projects on Codeskulptor was daunting. In addition, merging pieces of code together in *Interaction* class was a challenge because it was not easy to understand my teammates’ codes and logics.

However, after creating this game in Python, I learned a lot about its class implementations, how they work and so on. Not only that, I also learned how to use GitHub to collaborate with my teammates. I was a bit disappointed that we didn’t have that much time to implement the game to run it perfectly. In addition, reflecting on the development process, we incorporated a lot of unique features into the game, resulting in slower gameplay, which we didn’t think about in the planning stage of the game. If I were given another opportunity to re-make the game, I’d allocate more time to optimize the game’s performance to ensure smoother gameplay. Additionally, a more comprehensive testing would be conducted frequently to mitigate potential performance issues.

While the game may not have met our initial expectations, it is a testament to our teamwork and dedication. Eventually, we managed to include all necessary parts for the game such as collisions, the use of vector class, OOP design and so on.

**Conclusion**

To conclude, ZOMBRENA was a collaborative effort that required cooperation from teammates. Sprite sheets, timers, collisions, vectors and complicated algorithms were used. Despite encountering challenges such as not being able to integrate music and display sprite sheets from local files, we successfully delivered the game within the designated timeframe. Throughout the project, out team exhibited resilience and effective communication, which were important in overcoming challenges. We approached challenges as opportunities for learning and used our resources to find creative solutions. Not only does this project enhances our python skills, it also improves communication and collaboration within a group setting. Moving forward, I will remember the valuable lessons learned during the development process and will apply them in future projects.

**Reference**

1. Glass shattering: “Glass Smash” (https://pixabay.com/sound-effects/glass-smash-6266/)