SPACE OWNERS

ITEC5010W APPLIED PROG ('23)

INSTRUCTOR
Sima Soltanpour

Harsh Shah
Harsh Rupesh Shah 101182508

OBJECTIVES:

Data types, functions, conditional statements, loops, and object-oriented programming are just a few of the key ideas in Python that the Applied Programming with Python course intends to cover in addition to the fundamentals of programming. Using these ideas as a foundation, I created Space Owners, a space shooting game that will bring back memories for anyone who has ever used a computer.

This will be a Python-based arcade game that is evocative of a number of early 1980s "shot 'em up" games. The game's mechanics are highly dynamic, despite the fact that they were intended to be simple. The game's gameplay is also intriguing. In order to capture some of the necessary OS features, the pygame package and the os package are both widely utilized in this game. Below are some of the objectives behind choosing this project idea:

- To challenge players with obstacles that they must defeat
- To give an immersive and interesting environment that represents space-related concepts
- To create a compelling gameplay experience for gamers.

The following list of features for this gaming program:

1. **Navigation and movement**: Include movement via a 3D spatial environment and include characteristics like rotation, acceleration, and deceleration.
2. **Combat and weapon systems**: They include fighting off other obstacles and utilizing weapons like laser bullets.
3. **Resource management**: Each player can only use three laser bullets in a fair game.
4. **Dynamic gameplay**: Each participant has a total of 10 lives, and the game loops itself to keep everyone entertained until one of them gives up.
5. **Narrative**: Both players will have fun for free thanks to the colorful laser bullets, continuous action, and explosive sound effects in the game.

DETAILS OF THE PROJECT PLAN:

The complete project design plan for application interference is broken down into the following seven stages.

1. **Game Concept and Design**:

The concept for a simple peer-to-peer (p2p) space shooter game in Python involve two players battling against each other in a space-themed arena. Here are some elements that have been considered when designing the game:

- **Objective**: Each player must eliminate their opponent by shooting at their spaceship and dodging opposing fire in order to win the game.
- **Setting**: The game is played in a space-themed arena with a starry backdrop.
- **Game mechanics**: The arrow keys on their keyboard are used by players to steer their spaceship, and the ctrl buttons are used to fire their weaponry.
- **Multiplayer functionality**: The game is designed to be played between two players, with each player controlling their own spaceship.
- **Sound and visuals**: The game has 2D graphics with realistic spaceships. The soundtrack and sound effects for the game's weaponry, explosions are all space-themed.
- **User interface**: Players can examine their score, and check their health on the game's straightforward user interface.

Essentially, the gameplay mechanics and user interface are kept basic and clear in order to give players an engaging and competitive experience.

2. **Game Engine Selection**:

Here is a list of some popular game engines available in Python [1]:

- **Pygame** - a well-liked game engine for Python-based 2D games.
- **Arcade** - An OpenGL-compatible Python module for developing 2D games.
- **PyOpenGL** - a Python OpenGL API binding that may be used to create 3D games.
- **Panda3D** - A Python-based game engine that facilitates the creation of both 2D and 3D games.
- **PyOgre** - OGRE's 3D graphics engine has a Python binding that may be used to make 3D games.

While all of these game engines have their own strengths and weaknesses, I chose Pygame for several reasons [2][3]:

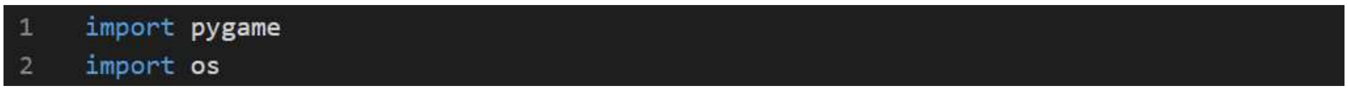
- **Ease of use**: Pygame is a fantastic choice for those who are just getting started with Python game creation because of its straightforward and simple-to-understand API.
- **Large user community**: Because Pygame has a wide user base, there are several online resources, such as tutorials, documentation, and examples.
- **Cross-platform support**: Pygame makes it simple to create games for several platforms because it is compatible with a variety of operating systems, including Windows, macOS, and Linux.
- **Powerful features**: A wide range of potent features, such as support for music, graphics, and input processing, are available in Pygame. It also permits the use of third-party libraries and programs, such as Tiled for producing game maps and PyTMX for building game levels.
- **Flexibility**: Platformers, puzzle games, shooters, and other game genres may all be created with Pygame's adaptable game engine. Moreover, it permits modification and expansion through the use of third-party tools and libraries.

3. Game Assets:

The game assets for this project can be categorized as follows:

- **Spaceships**: After doing some study on vibrant and competitive colors for arcade games, two yellow and one red spaceship were added to the game.
- **Bullets and projectiles**: In this game, laser bullets are used as projectiles, and they are colored like the spaceship from which they originated.
- **Background image**: To give the game a more authentic feel, a photo taken by NASA satellites was used.
- **Sound effects and music**: To produce the sounds of projectiles being fired and explosions, two audio files with a creative commons license were used in this project.
- **User interface (UI) components**: Lives counter, winner name, three bullets every round for a fair game, and an infinite loop of fun unless a player presses the "X" button.

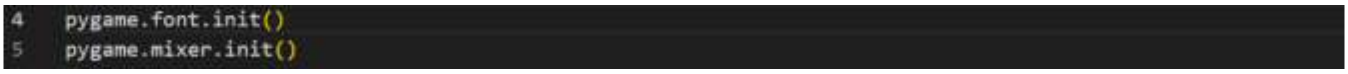
4. **Game Programming**:



```python
import pygame
import os
```

These two lines of code import two Python modules, pygame and os.

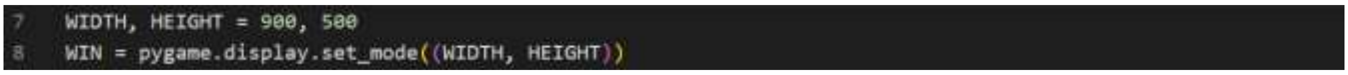
- A well-liked cross-platform package called *pygame* is used by Python programmers to create games and multimedia applications. It offers a variety of game development functions, including managing graphics, sound, input devices, and more.
- The Python standard library includes a module called *os* that offers a means to communicate with the operating system. It provides tools for adding, removing, and changing files and directories as well as methods for learning about the environment and system variables.



```python
pygame.font.init()
pygame.mixer.init()
```

These two lines of code initialize two Pygame modules, pygame.font and pygame.mixer.

- A Pygame module called pygame.font offers support for text rendering on surfaces. It offers methods for adjusting font size, style, and colour, and allows you to create font objects that may be used to output text onto surfaces.
- A Pygame module called pygame.mixer offers assistance for playing sound effects and music. You can use it to load and play sound files in your game.

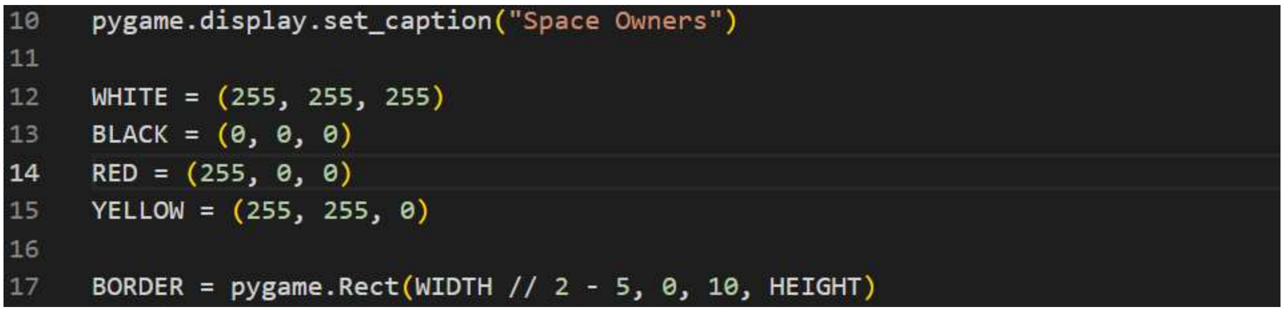


```python
WIDTH, HEIGHT = 900, 500
WIN = pygame.display.set_mode((WIDTH, HEIGHT))
```

These two lines of code create a game window using the pygame.display module and set its dimensions to WIDTH and HEIGHT.

- The game window's width and height are stored in the variables WIDTH and HEIGHT, respectively. In this instance, they are given values of 900 and 500.
- A gaming window with the desired dimensions is created using the Pygame function pygame.display.set_mode(). The function returns a Surface object that symbolizes the game window and accepts a tuple (WIDTH, HEIGHT) as an argument.

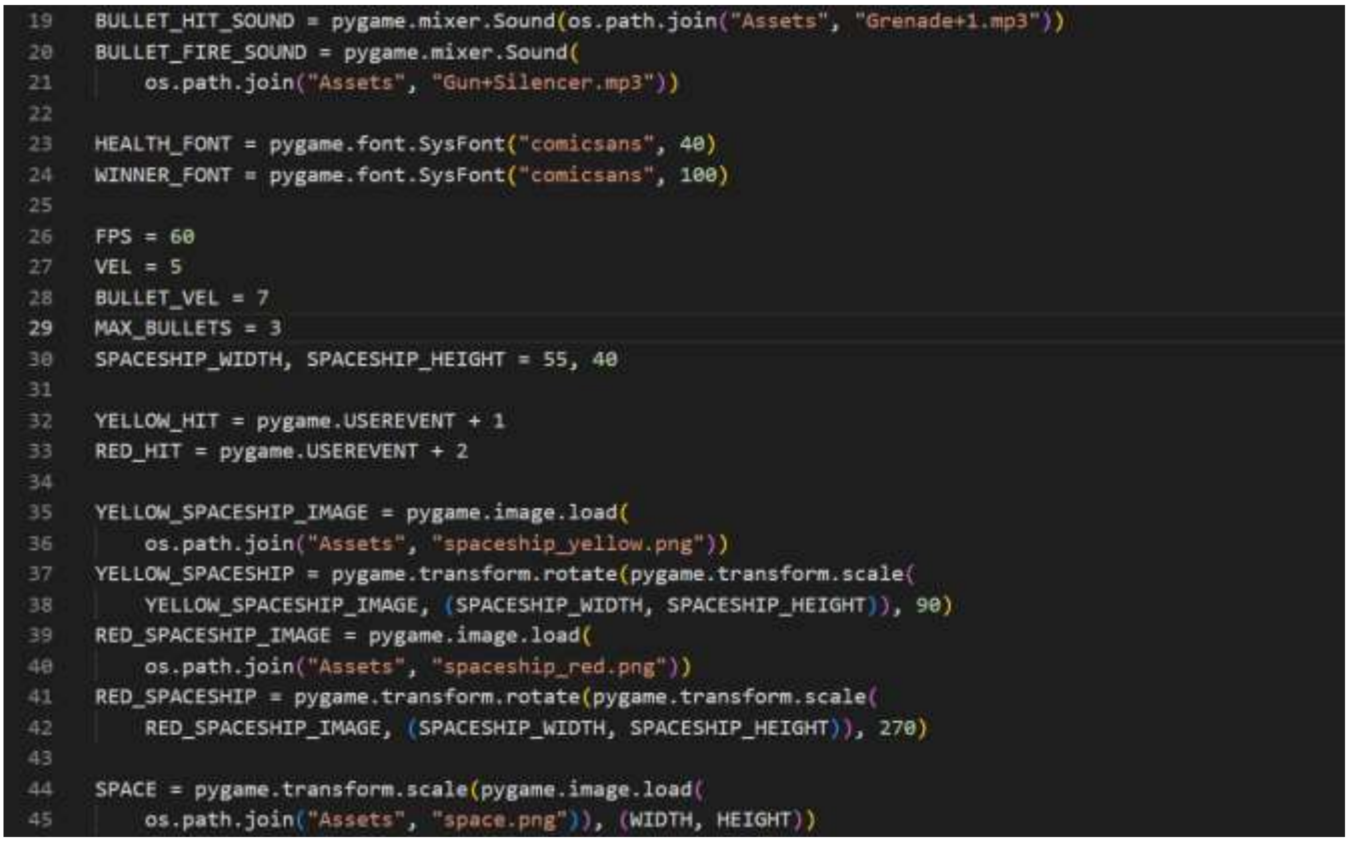
- The WIN variable is then given the Surface object provided by pygame.display.set_mode(). This enables the WIN variable to be used in code to refer to the game window.



```python
pygame.display.set_caption("Space Owners")

WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
RED = (255, 0, 0)
YELLOW = (255, 255, 0)

BORDER = pygame.Rect(WIDTH // 2 - 5, 0, 10, HEIGHT)
```

These lines of code provide the window caption, a few color constants, the size, and the location of the border that will divide the game screen into two equal halves.

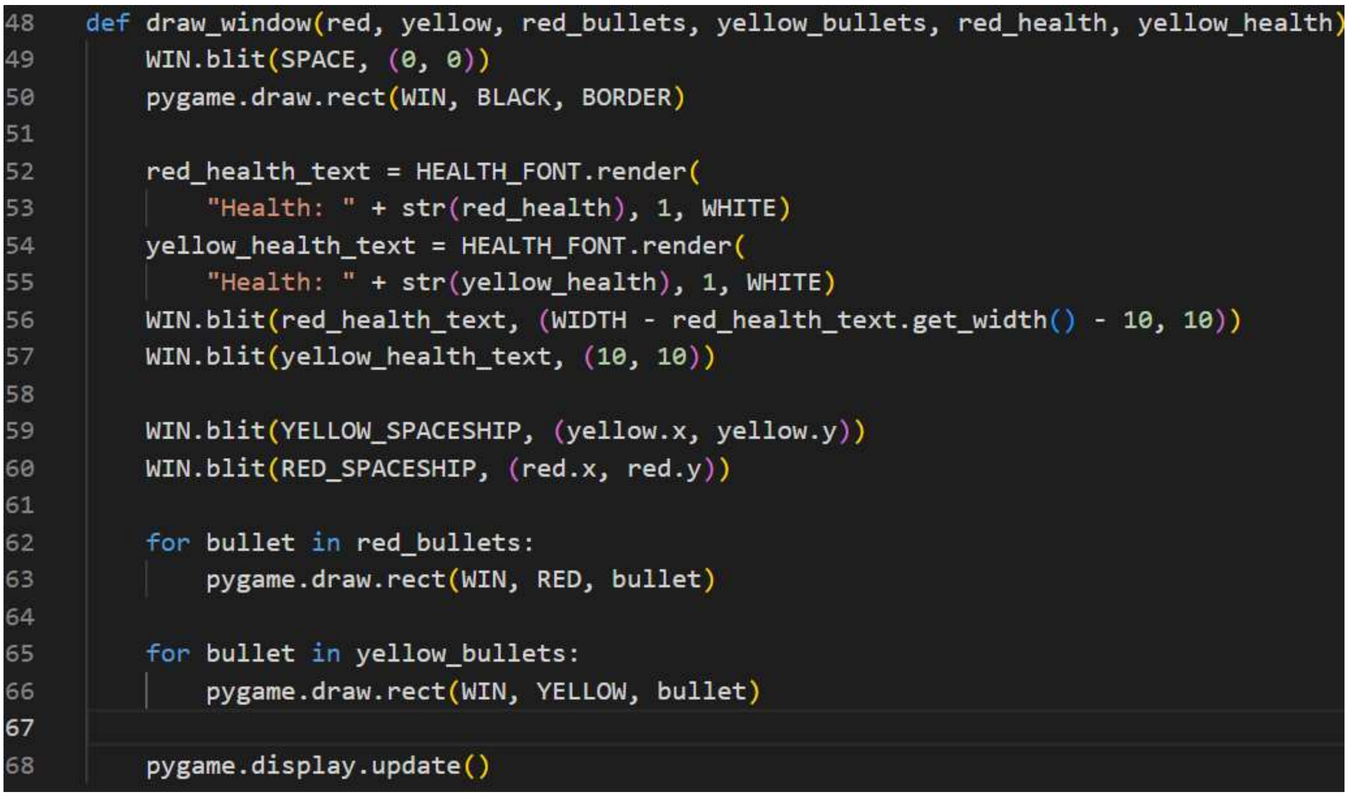
- The window caption is changed to "Space Owners" by calling pygame.display.set_caption("Space Owners"). This text can be seen in the game window's title bar.
- The RGB values for the hues WHITE, BLACK, RED, and YELLOW are represented as tuples.
- BORDER is a pygame.Rect object that specifies a 10 pixel wide by full screen high rectangle that is horizontally aligned. Each player is positioned on one side of this rectangle, which divides the game screen into two equal halves.



```
BULLET_HIT_SOUND = pygame.mixer.Sound(os.path.join("Assets", "Grenade+1.mp3"))
BULLET_FIRE_SOUND = pygame.mixer.Sound(
    os.path.join("Assets", "Gun+Silencer.mp3"))

HEALTH_FONT = pygame.font.SysFont("comicsans", 40)
WINNER_FONT = pygame.font.SysFont("comicsans", 100)

FPS = 60
VEL = 5
BULLET_VEL = 7
MAX_BULLETS = 3
SPACESHIP_WIDTH, SPACESHIP_HEIGHT = 55, 40

YELLOW_HIT = pygame.USEREVENT + 1
RED_HIT = pygame.USEREVENT + 2

YELLOW_SPACESHIP_IMAGE = pygame.image.load(
    os.path.join("Assets", "spaceship_yellow.png"))
YELLOW_SPACESHIP = pygame.transform.rotate(pygame.transform.scale(
    YELLOW_SPACESHIP_IMAGE, (SPACESHIP_WIDTH, SPACESHIP_HEIGHT)), 90)
RED_SPACESHIP_IMAGE = pygame.image.load(
    os.path.join("Assets", "spaceship_red.png"))
RED_SPACESHIP = pygame.transform.rotate(pygame.transform.scale(
    RED_SPACESHIP_IMAGE, (SPACESHIP_WIDTH, SPACESHIP_HEIGHT)), 270)

SPACE = pygame.transform.scale(pygame.image.load(
    os.path.join("Assets", "space.png")), (WIDTH, HEIGHT))
```

These lines of code define some constants and resources that are used in the game:

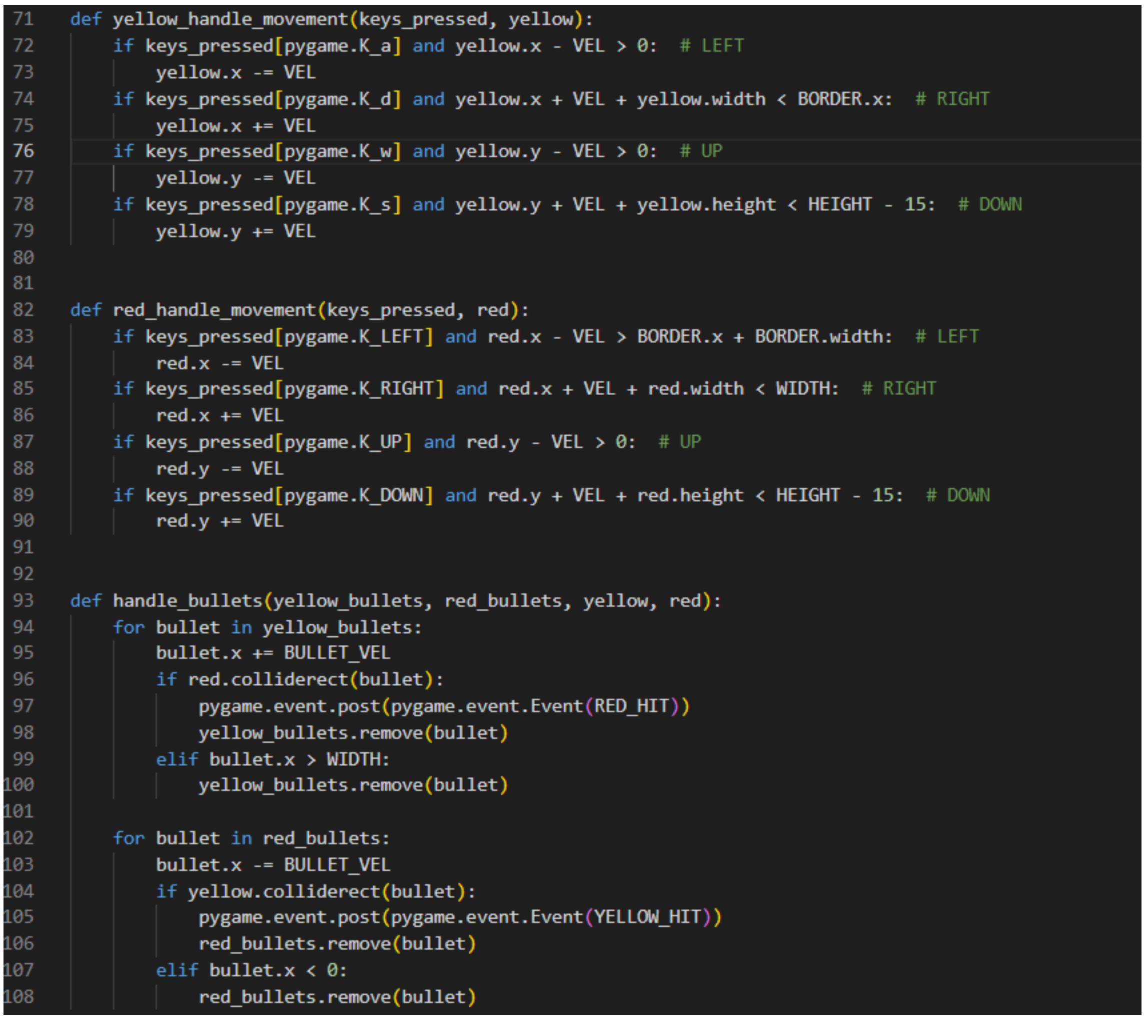
- The sounds for when a bullet strikes a spaceship and when a bullet is fired are respectively designated as BULLET_HIT_SOUND and BULLET_FIRE_SOUND.
- Fonts used to display the winner and the health are HEALTH_FONT and WINNER_FONT.
- FPS, VEL, BULLET_VEL, and MAX_BULLETS are variables that regulate the game's and the bullets' pace.
- The dimensions of the spacecraft images are SPACESHIP_WIDTH and SPACESHIP_HEIGHT.
- Yellow and red custom events are used to track conflicts between spacecraft and projectiles.
- The images of the yellow and red spaceships are YELLOW_SPACESHIP_IMAGE and RED_SPACESHIP_IMAGE, respectively.
- The scaled and rotated versions of the spacecraft photos are designated as YELLOW_SPACESHIP and RED_SPACESHIP, respectively.
- SPACE represents the game's backdrop image.



```python
def draw_window(red, yellow, red_bullets, yellow_bullets, red_health, yellow_health)
    WIN.blit(SPACE, (0, 0))
    pygame.draw.rect(WIN, BLACK, BORDER)

    red_health_text = HEALTH_FONT.render(
        "Health: " + str(red_health), 1, WHITE)
    yellow_health_text = HEALTH_FONT.render(
        "Health: " + str(yellow_health), 1, WHITE)
    WIN.blit(red_health_text, (WIDTH - red_health_text.get_width() - 10, 10))
    WIN.blit(yellow_health_text, (10, 10))

    WIN.blit(YELLOW_SPACESHIP, (yellow.x, yellow.y))
    WIN.blit(RED_SPACESHIP, (red.x, red.y))

    for bullet in red_bullets:
        pygame.draw.rect(WIN, RED, bullet)

    for bullet in yellow_bullets:
        pygame.draw.rect(WIN, YELLOW, bullet)

    pygame.display.update()
```

This function changes the game window by drawing the spacecraft, bullets, and health bars using a number of parameters.

- The spacecraft (YELLOW_SPACESHIP and RED_SPACESHIP) and backdrop picture (SPACE) are drawn onto the game window using the WIN.blit() method. The image to draw and the place at which to draw it are the two parameters required by the blit() function.
- The middle of the screen is surrounded by a black border thanks to the pygame.draw.rect() function (BORDER). The window to draw on, the color of the rectangle, the rectangle itself (formerly described as BORDER), and an optional border radius are the four inputs for the rect() function.
- The HEALTH_FONT font, which was defined previously in the code, is used to render the health bars. The health text and color are created on a surface using the render() function, and then it is blitted onto the game window (). The health bars on the screen's borders are centered using the get width() method.
- The bullets are drawn on the game window using for loops. Using pygame.draw.rect, a rectangle with the color of the appropriate spaceship is drawn on the game window for each bullet in red bullets or yellow bullets ().
- The game window is updated with all of the modifications made in this function by calling the pygame.display.update() function at the end.



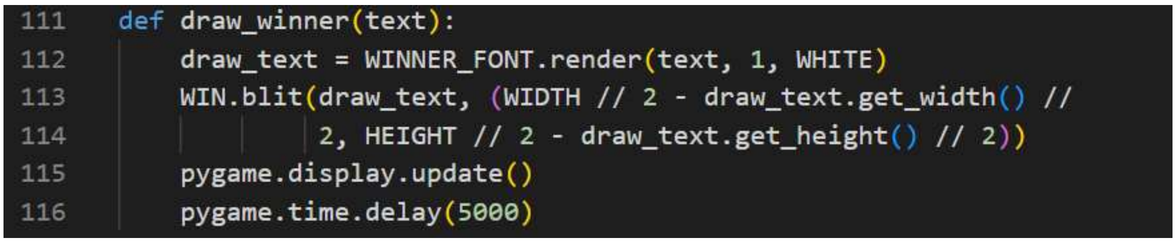
```python
def yellow_handle_movement(keys_pressed, yellow):
    if keys_pressed[pygame.K_a] and yellow.x - VEL > 0:  # LEFT
        yellow.x -= VEL
    if keys_pressed[pygame.K_d] and yellow.x + VEL + yellow.width < BORDER.x:  # RIGHT
        yellow.x += VEL
    if keys_pressed[pygame.K_w] and yellow.y - VEL > 0:  # UP
        yellow.y -= VEL
    if keys_pressed[pygame.K_s] and yellow.y + VEL + yellow.height < HEIGHT - 15:  # DOWN
        yellow.y += VEL


def red_handle_movement(keys_pressed, red):
    if keys_pressed[pygame.K_LEFT] and red.x - VEL > BORDER.x + BORDER.width:  # LEFT
        red.x -= VEL
    if keys_pressed[pygame.K_RIGHT] and red.x + VEL + red.width < WIDTH:  # RIGHT
        red.x += VEL
    if keys_pressed[pygame.K_UP] and red.y - VEL > 0:  # UP
        red.y -= VEL
    if keys_pressed[pygame.K_DOWN] and red.y + VEL + red.height < HEIGHT - 15:  # DOWN
        red.y += VEL


def handle_bullets(yellow_bullets, red_bullets, yellow, red):
    for bullet in yellow_bullets:
        bullet.x += BULLET_VEL
        if red.colliderect(bullet):
            pygame.event.post(pygame.event.Event(RED_HIT))
            yellow_bullets.remove(bullet)
        elif bullet.x > WIDTH:
            yellow_bullets.remove(bullet)

    for bullet in red_bullets:
        bullet.x -= BULLET_VEL
        if yellow.colliderect(bullet):
            pygame.event.post(pygame.event.Event(YELLOW_HIT))
            red_bullets.remove(bullet)
        elif bullet.x < 0:
            red_bullets.remove(bullet)
```

This code defines three functions: yellow_handle_movement, red_handle_movement, and handle_bullets.

- yellow_handle_movement takes two arguments: keys_pressed, a dictionary representing the state of all keyboard keys, and yellow, a pygame.Rect object representing the yellow spaceship. This function moves the yellow spaceship according to the state of the keys in the keys_pressed dictionary. If the pygame.K_a key is pressed and the yellow spaceship will still be on the screen after moving left by VEL pixels, the function moves the yellow spaceship left by VEL. Similarly, if the pygame.K_d key is pressed and the yellow spaceship will still be on the screen after moving right by VEL pixels, the function moves the yellow spaceship right by VEL. The same is true for the pygame.K_w and pygame.K_s keys, which move the yellow spaceship up and down, respectively.
- red_handle_movement is similar to yellow_handle_movement, but it moves the red spaceship instead. If the pygame.K_LEFT key is pressed and the red spaceship will still be on the screen after moving left by VEL pixels, the function moves the red spaceship left by VEL. Similarly, if the pygame.K_RIGHT key is pressed and the red spaceship will still be on the screen after moving

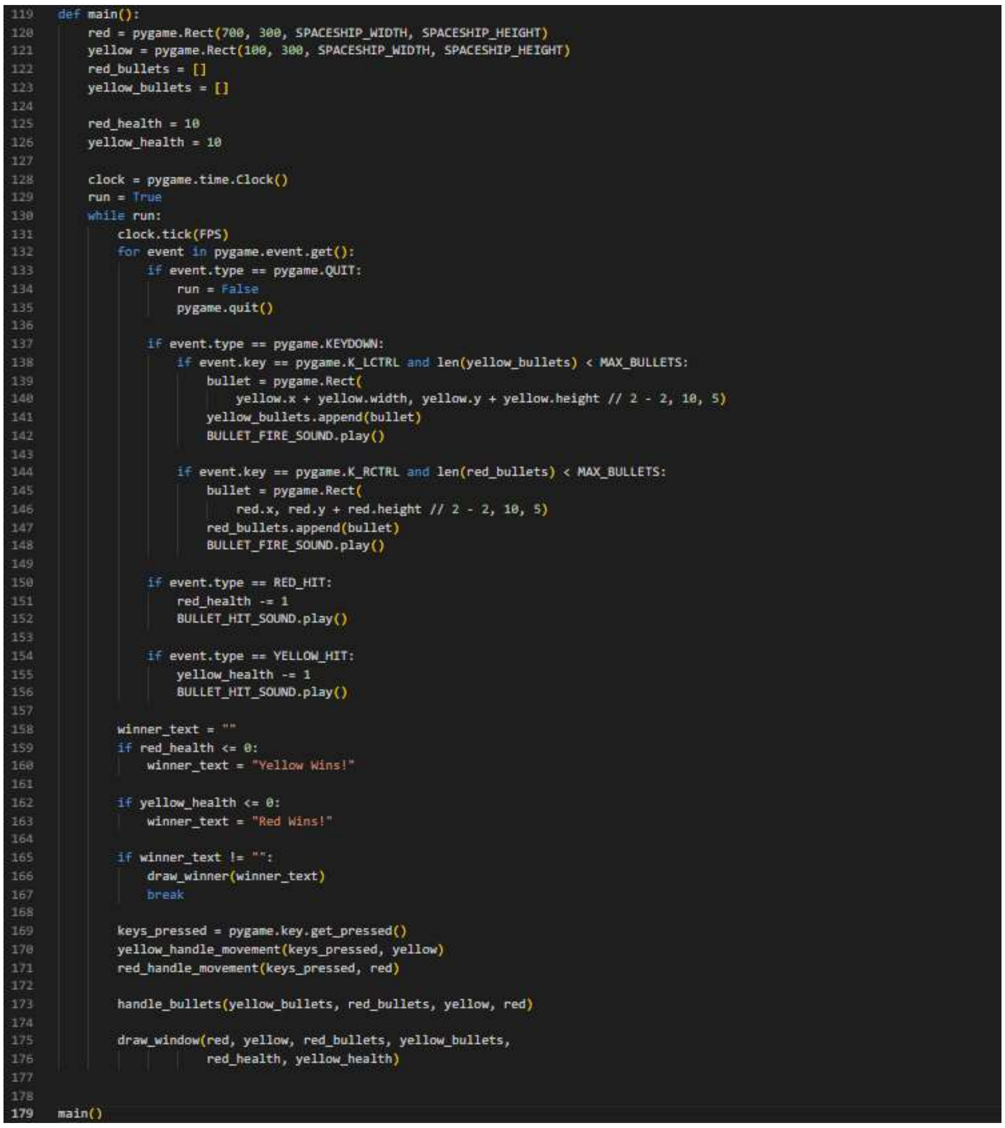
right by VEL pixels, the function moves the red spaceship right by VEL. The same is true for the pygame.K_UP and pygame.K_DOWN keys, which move the red spaceship up and down, respectively.

- handle_bullets takes four arguments: yellow_bullets, a list of pygame.Rect objects representing the bullets fired by the yellow spaceship; red_bullets, a list of pygame.Rect objects representing the bullets fired by the red spaceship; yellow, a pygame.Rect object representing the yellow spaceship; and red, a pygame.Rect object representing the red spaceship. This function moves each bullet in yellow_bullets to the right by BULLET_VEL pixels, and each bullet in red_bullets to the left by BULLET_VEL pixels. If a bullet in yellow_bullets collide with the red spaceship, the function generates a RED_HIT event and removes the bullet from yellow_bullets. If a bullet in red_bullets collide with the yellow spaceship, the function generates a YELLOW_HIT event and removes the bullet from red_bullets. If a bullet goes off the screen, the function removes it from the appropriate list.



```python
def draw_winner(text):
    draw_text = WINNER_FONT.render(text, 1, WHITE)
    WIN.blit(draw_text, (WIDTH // 2 - draw_text.get_width() //
                2, HEIGHT // 2 - draw_text.get_height() // 2))
    pygame.display.update()
    pygame.time.delay(5000)
```

- The draw_winner function displays the winner text on the screen and updates the display. It takes a single parameter text which is the string to be displayed as the winner.
- First, the function renders the text with the WINNER_FONT font and color WHITE. The rendered text is then displayed on the center of the screen by calculating the x and y coordinates based on the width and height of the text surface. Finally, the function updates the display and waits for 5 seconds before returning.



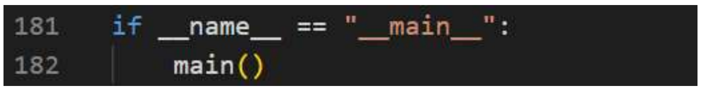
```python
def main():
    red = pygame.Rect(700, 300, SPACESHIP_WIDTH, SPACESHIP_HEIGHT)
    yellow = pygame.Rect(100, 300, SPACESHIP_WIDTH, SPACESHIP_HEIGHT)
    red_bullets = []
    yellow_bullets = []

    red_health = 10
    yellow_health = 10

    clock = pygame.time.Clock()
    run = True
    while run:
        clock.tick(FPS)
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                run = False
                pygame.quit()

            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_LCTRL and len(yellow_bullets) < MAX_BULLETS:
                    bullet = pygame.Rect(
                        yellow.x + yellow.width, yellow.y + yellow.height // 2 - 2, 10, 5)
                    yellow_bullets.append(bullet)
                    BULLET_FIRE_SOUND.play()

                if event.key == pygame.K_RCTRL and len(red_bullets) < MAX_BULLETS:
                    bullet = pygame.Rect(
                        red.x, red.y + red.height // 2 - 2, 10, 5)
                    red_bullets.append(bullet)
                    BULLET_FIRE_SOUND.play()

            if event.type == RED_HIT:
                red_health -= 1
                BULLET_HIT_SOUND.play()

            if event.type == YELLOW_HIT:
                yellow_health -= 1
                BULLET_HIT_SOUND.play()

        winner_text = ""
        if red_health <= 0:
            winner_text = "Yellow Wins!"

        if yellow_health <= 0:
            winner_text = "Red Wins!"

        if winner_text != "":
            draw_winner(winner_text)
            break

        keys_pressed = pygame.key.get_pressed()
        yellow_handle_movement(keys_pressed, yellow)
        red_handle_movement(keys_pressed, red)

        handle_bullets(yellow_bullets, red_bullets, yellow, red)

        draw_window(red, yellow, red_bullets, yellow_bullets,
                    red_health, yellow_health)


main()
```

This code defines the main game loop for a 2-player space shooter game using the Pygame library in Python.

- The first few lines define the initial positions and dimensions of the spaceships (red and yellow), and the initial lists for the bullets fired by each player. The initial health for both players is set to 10.
- A clock object is created to regulate the game speed by ensuring that the game loop runs at a consistent frame rate. The variable 'run' is initialized to True to allow the loop to run.
- Inside the loop, the clock is ticked and events are handled using a for loop to check each event in the Pygame event queue. If the 'QUIT' event is detected (typically when the user clicks the close button on the game window), the loop is exited and Pygame is quit.
- The loop also checks for keyboard input events to detect when the players have pressed the 'LEFT CTRL' or 'RIGHT CTRL' keys to fire bullets. The code checks if the maximum number of bullets

have already been fired by the player before appending a new bullet to the corresponding bullet list. A bullet object is created as a Pygame Rect object with appropriate dimensions and positions, and added to the appropriate bullet list. A sound effect is played when a bullet is fired.

- The code also listens for 'RED_HIT' and 'YELLOW_HIT' events that indicate that a player has been hit by a bullet fired by the other player. The code reduces the health of the corresponding player by 1, and plays a sound effect.
- The game loop also checks if either player's health has reached 0, in which case the winner is determined and the game loop is exited. The winner text is drawn using the 'draw_winner' function.
- Finally, the code checks for keyboard input to move the spaceships and calls the 'handle_bullets' function to update the position of the bullets and check for collisions with the opposing player's spaceship. The game window is then updated with the new positions of the spaceships and bullets, as well as the updated health of both players.



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

If the module is being executed as the main program or if it is being imported as a module into another program, it is determined by the line if __name__ == "__main__". If so, the main() function, which houses the main game loop and manages the game logic, is invoked. By doing this, you can make sure that the code in main() is only called when the file is run as the main program and not when it is imported as a module into another program.

5. **Game Testing and Debugging**:

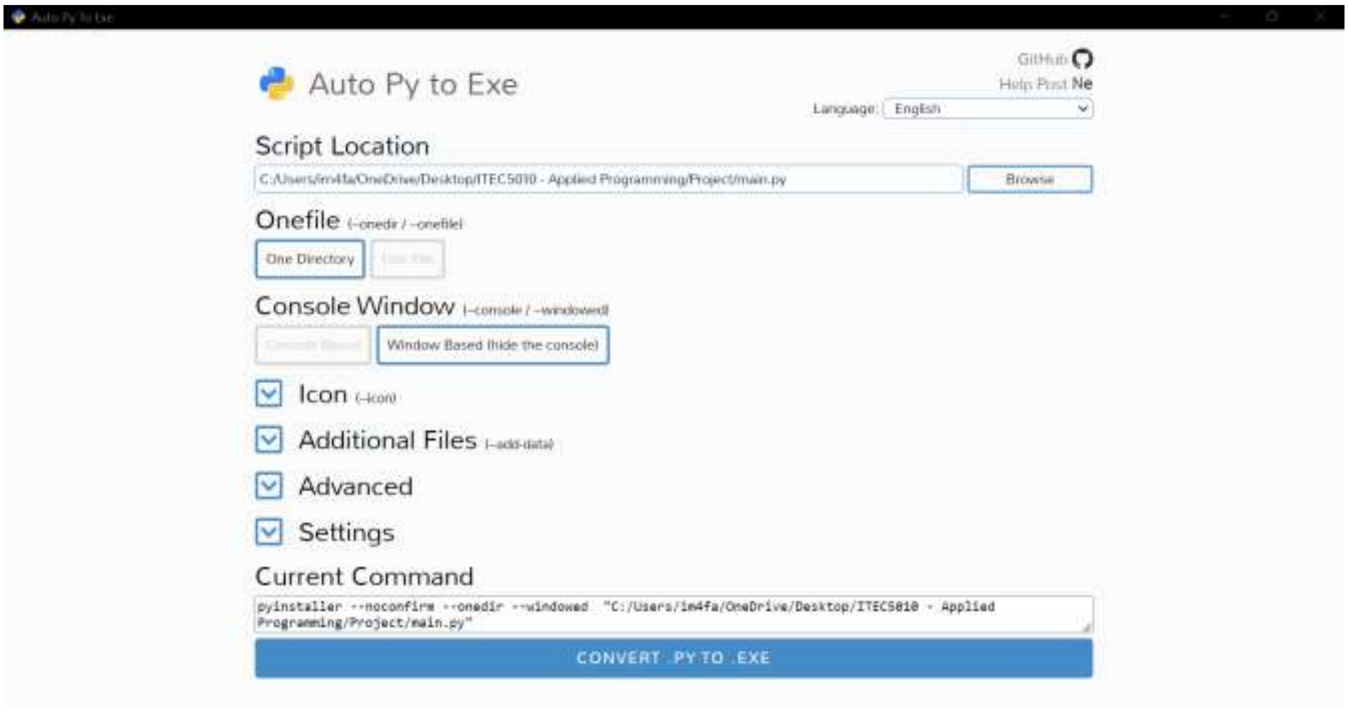
A key component of game development is testing and debugging, especially when creating multiplayer games like a Python P2P space shooter. It's crucial to thoroughly test the game on both local and P2P networks to make sure it runs well.

This application can produce a top-notch and entertaining game for players by employing the following testing and debugging techniques:

- Manual testing utilizing the VSCode debugger to find and correct faults and defects.
- Automated testing tools like Pytest or unittest utilised to speed up the testing process and reduce the probability of human mistake.

- Ultimately, feedback from players who have tested the game aided in identifying problems that may have been missed and improving the game before it was released.

6. **Game Deployment**:



Auto Py to Exe

Language: English

Script Location

C:/Users/im4fa/OneDrive/Desktop/ITEC5010 - Applied Programming/Project/main.py

Onefile (--onedir / --onefile): One Directory

Console Window (--console / --windowed): Window Based (hide the console)

Icon (--icon)

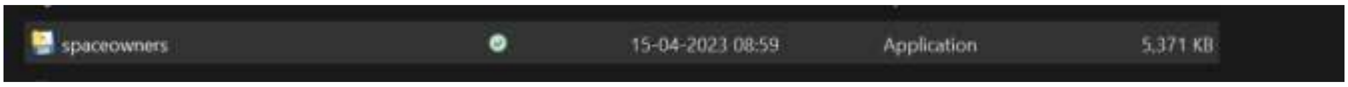
Additional Files (--add-data)

Advanced

Settings

Current Command

```
pyinstaller --noconfirm --onedir --windowed  "C:/Users/im4fa/OneDrive/Desktop/ITEC5010 - Applied Programming/Project/main.py"
```



spaceowners	15-04-2023 08:59	Application	5,371 KB

- The.py file has been converted into an executable program for app deployment using the Auto Py to Exe tool.
- Python scripts can be turned into standalone executable files for Windows operating systems using the software program Autopy to exe.
- A single executable file including the Python script, all of its dependencies and resources, including libraries and assets like graphics, sounds, and data files, is created by the tool using PyInstaller. This enables users to run the Python script independently of a Python interpreter or any other third-party dependencies [4].
- With the help of its user-friendly GUI, Autopy to exe makes it easier to convert Python scripts into executable files.

7. **Game Maintenance and Updates**:

Here are some steps that will be considered for maintaining and updating the game:

- **Monitor game performance**: Check the game's performance data frequently to spot any problems that need be fixed, such latency or server outages.
- **Fix bugs**: Keep an eye out for issues in the game and fix them as soon as you find them. Analyzing game logs, player comments, or using automated testing tools can all be used to achieve this.
- **Release updates**: Deliver frequent game updates that include new features, bug fixes, and enhancements. This may help retain current players and draw in new ones.
- **Manage game servers**: Maintain the game servers to make sure they are reliable and able to handle the traffic associated with the game. This include taking care of backups, maintaining server hardware, and putting security measures in place.
- **Communicate with players**: Inform players on a regular basis about upcoming updates, maintenance windows, and bug fixes. Social media, email, and in-game announcements can all be used for this.
- **Test updates:** Updates should be rigorously tested to verify that they are stable so not to bring any new bugs or problems before being released.

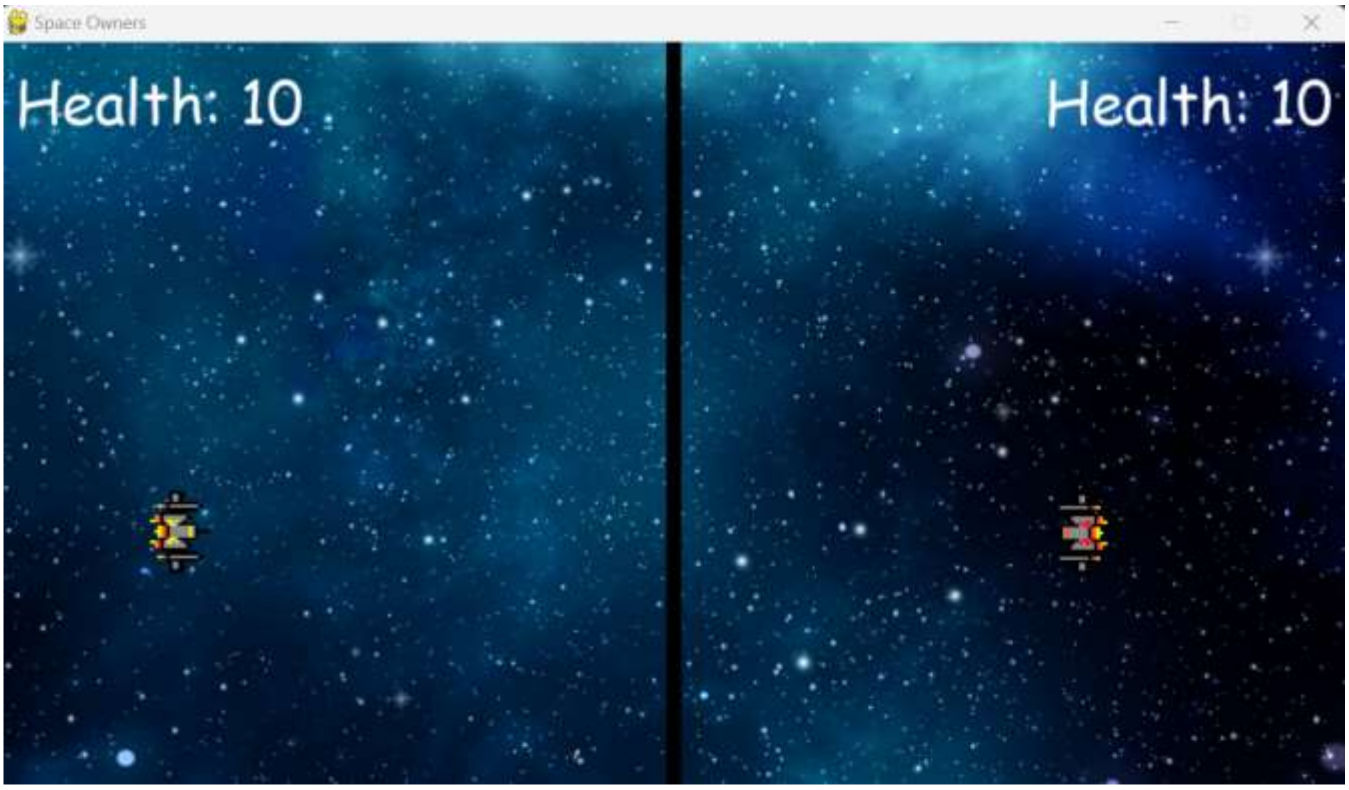
CHALLENGES:

The process of developing this gaming program was great. But the following are some of the difficulties encountered on the way to this entertaining space combat:

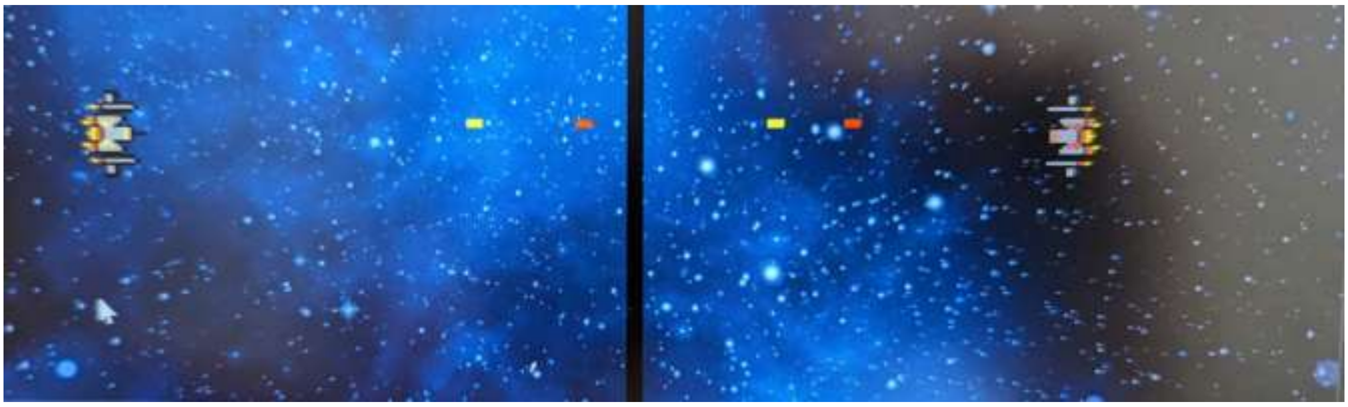
1. Designing an enjoyable game experience that is balanced for both players was one of the biggest challenges. Careful consideration must be given to aspects like the potency of weaponry, the resilience of ships, and the complexity of enemy encounters when balancing the game.
2. Implementing dependable network synchronization presented another difficulty. Any network communication latency in a two-person game might result in delays, making it challenging to coordinate game variables and player actions.
3. A two-player space shooting game in Python was difficult to test because it had to work properly for both players under a variety of hardware and network situations.

OBSERVATIONS:

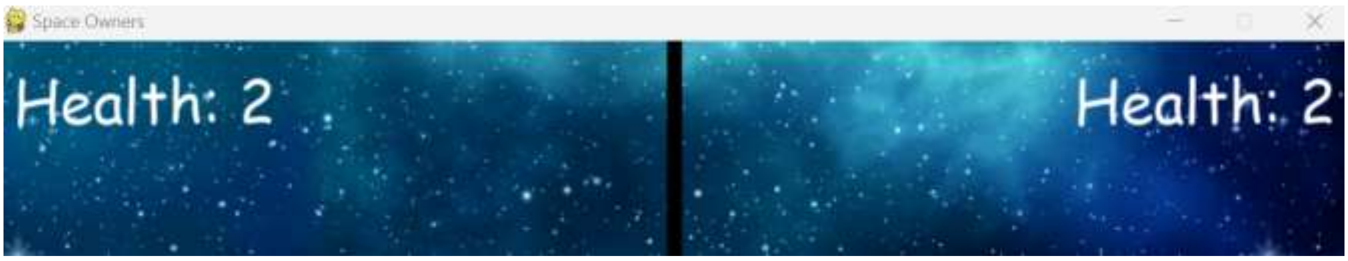
1. **Main GUI**



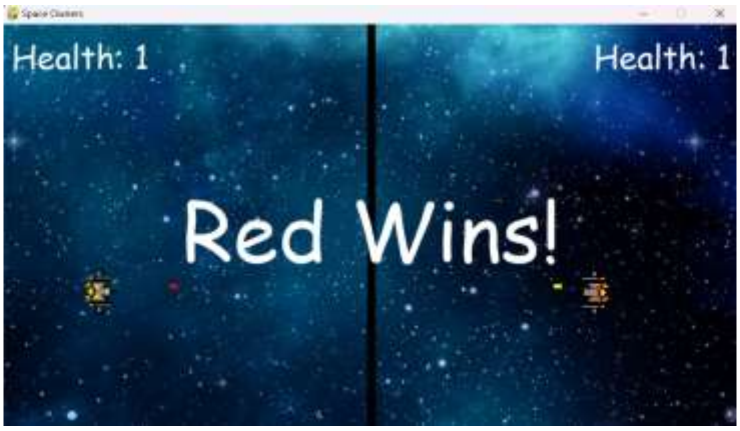
2. **Live Combat**:



3. **Lives counter**:



4. **Result**:



FUTURE WORK:

Future development in this space shooting game has a number of promising directions.

- One area is enhancing network synchronization to guarantee fluid gameplay in a variety of network environments. This can entail enhancing peer finding and connection management, streamlining network communication protocols, and lowering latency.
- Improving the game's audio and visual components to give players a more immersive experience is another area of future work. To enhance the overall visual of the game, this can entail incorporating more sophisticated graphics, animations, and sound effects. Also, giving opposing ships more sophisticated AI might give the game a new depth. Players may have more difficult and interesting gaming by developing cleverer enemy AI.
- More game features will be added in the future to improve the game's replay value and longevity. For instance, expanding the selection of available weaponry, power-ups, and ship customization options can give gamers greater diversity and flexibility in their gameplay.
- The competitive parts of the game can be improved, players can be encouraged to keep playing, and the game can be made more fun all around by incorporating a ranking or matchmaking system based on player performance.

REFERENCES:

[1] "Python for Engineers," Python for Engineers, Feb. 28, 2023. https://new.pythonforengineers.com/

[2] R. Python, "Python Tutorials – Real Python." https://realpython.com/

[3] "Make Tech Easier," Make Tech Easier, Apr. 14, 2023. https://www.maketecheasier.com/

[4] F. Shadhin, "Convert a Python Project to an Executable (.exe) File," Medium, Sep. 08, 2021. https://python.plainenglish.io/convert-a-python-project-to-an-executable-exe-file-175080da4485