Cosmic Duel – Code Break Down

Game Overview: Cosmic Duel 🚀🌌

Cosmic Duel is a fast-paced, two-player space shooter game built using **Pygame**. Players take control of spaceships and engage in an intergalactic battle, dodging bullets and firing at their opponent while navigating a scrolling cosmic battlefield. The goal is to deplete the opponent's health by landing successful shots while avoiding incoming attacks.

Gameplay Features:

- **Two-player combat**: One player controls the **Yellow Spaceship**, and the other controls the **Red Spaceship**.
- **Bullet mechanics**: Each spaceship can fire a limited number of bullets, requiring strategic timing and positioning.
- **Dynamic background**: A continuously scrolling space backdrop enhances immersion.
- **Health system**: Players start with **100 health points**, and each hit reduces their health.
- **Victory conditions**: The game ends when a player's health reaches zero, displaying the winner.
- **Restart button with hover effect**: After a match, players can restart the game with a visually responsive button.

Table of Contents

Step 1: Set up the main window of the game & Title

Creating the Game Window

```
import pygame

# Game window settings
WIDTH, HEIGHT = 900, 500 # define the width and height of the game window
WIN = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Cosmic Duel") # set the title of the window
```

Explanation:

- WIN is the main surface where all the game elements will be drawn.
- pygame.display.set_mode((WIDTH, HEIGHT)) creates a window of the specified width and height.
- We use **all capital letters** for WIDTH and HEIGHT to indicate that these values are constants (a common convention in Python).
- The width and height are stored in a **tuple** because pygame.display.set_mode() requires a tuple as an argument.
- pygame.display.set_caption("Cosmic Duel") enables us to create a title for our games.

Why Does the Window Close Immediately?

If you run the code above, the window will briefly appear and then disappear. This happens because the program runs the script and immediately exits.

To keep the window open, we need a **game loop** that continuously updates the screen. We will use a while loop to accomplish this in the next step.

Step 2: Creating the game loop and quitting the game

Understanding the Game Loop

In most games, the game needs to continuously run until the player decides to quit. To achieve this, we use a **game loop** that keeps updating the game screen and checking for user inputs (such as key presses, mouse clicks, or quitting the game).

```
def main():
    # Main function to run the game
    run = True # variable to control the game loop
    while run:
        for event in pygame.event.get(): # loop through all events
```

```
            # check for quit event
            if event.type == pygame.QUIT:
                run = False # exit the game loop
    pygame.quit() # quit pygame when the game loop ends

if __name__ == "__main__":
    main() # call the main function to start the game
```

Explanation:

- run = True: This variable controls whether the game should continue running.
- while run: This loop ensures that the game runs continuously until run is set to False.
- pygame.event.get(): Retrieves all **events** (such as key presses, mouse clicks, or closing the window).
- if event.type == pygame.QUIT: This checks if the player clicks the close button (X on the window). If so, run is set to False, ending the loop.
- pygame.quit(): Once the loop ends, pygame.quit() properly shuts down the game and releases resources.
- When you run the code and press the x button on the top right conner you are able to quit the game.

What Are Pygame Events?

In Pygame, an **event** is anything that happens while the game is running, such as:

- Pressing a key (KEYDOWN, KEYUP)
- Moving the mouse (MOUSEMOTION)
- Clicking the mouse (MOUSEBUTTONDOWN, MOUSEBUTTONUP)
- Closing the game window (QUIT)

Pygame automatically tracks all events and stores them in an **event queue**. We access this queue using pygame.event.get(), which returns a list of all events that have occurred since the last update.

Common Pygame Events:

Event Type	**Description**
pygame.QUIT	Triggered when the user clicks the close button (X)
pygame.KEYDOWN	Triggered when a key is pressed
pygame.KEYUP	Triggered when a key is released

Event Type	Description
pygame.MOUSEBUTTONDOWN	Triggered when a mouse button is pressed
pygame.MOUSEBUTTONUP	Triggered when a mouse button is released
pygame.MOUSEMOTION	Triggered when the mouse is moved

In this step, we only handle the QUIT event, but we will later add more event handling for player controls and game mechanics.

Step 3: Drawing on the screen and using colors.

Filling the Background with a Color

To make the game visually appealing, we need to **draw elements** on the screen. The simplest way to start is by **filling the background with a solid color**.

```
WHITE = (255, 255, 255) # define color white

def draw_window(): # function to draw the game window
    WIN.fill(WHITE) # fill the window with white color
    pygame.display.update() # update the display
```

Explanation:

- WHITE = (255, 255, 255): Defines the **RGB color** for white.
- WIN.fill(WHITE): Fills the entire window (WIN) with the specified color.
- pygame.display.update(): Updates the display so that the changes are visible. If we don't call this, the screen won't refresh with the new color.
- After defining the draw_window function make sure you call it in the main function to ensure that it runs. (see full code in the section with full code)

Understanding Colors in Python (RGB Format)

Colors in Python (and Pygame) are usually defined using the **RGB (Red, Green, Blue)** color model. Each color component has a value between **0 and 255**, where:

- 0 means **no intensity** (black)
- 255 means **full intensity** (brightest)

Each color is represented as a tuple **(Red, Green, Blue)**:

Color Name	RGB Value
Black	(0, 0, 0)
White	(255, 255, 255)
Red	(255, 0, 0)
Green	(0, 255, 0)
Blue	(0, 0, 255)
Yellow	(255, 255, 0)
Cyan	(0, 255, 255)
Magenta	(255, 0, 255)
Gray	(128, 128, 128)

You can use **custom colors** by adjusting the RGB values.

Here is a snippet of the Full code so far:

```
import pygame

# Game window

WIDTH, HEIGHT = 900, 500 # define the width and height of the game window
WIN = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Cosmic Duel") # set the title of the window

WHITE = (255, 255, 255) # define color white

def draw_window(): # function to draw the game window
    WIN.fill(WHITE) # fill the window with white color
    pygame.display.update() # update the display

def main():
    # Main function to run the game
    run = True # variable to control the game loop
    while run:
        for event in pygame.event.get(): # loop through all events
            # check for quit event
            if event.type == pygame.QUIT:
```

```
                run = False # exit the game loop
        draw_window() # call the draw function
        pygame.display.update() # update the display

    pygame.quit() # quit pygame when the game loop ends

if __name__ == "__main__":
    main() # call the main function to start the game
```

Notice: The draw_window function is called in the main function to ensure it runs within the loop.

Step 4: Setting up the Frame rate.

What is Frame Rate?

The **frame rate** (frames per second, or FPS) determines how many times per second the screen updates. A higher FPS makes the game run smoother, while a lower FPS can make it appear laggy or choppy.

To maintain a consistent speed across different computers, we need to **control the frame rate** using pygame.time.Clock().

```
# Define Frame Rate
FPS = 60 # set the frames per second

def main():
    clock = pygame.time.Clock() # create a clock object to control the frame rate

    # Main function to run the game
    run = True # variable to control the game loop
    while run:
        clock.tick(FPS) # set the frame rate
```

Explanation:

- FPS = 60: This defines the maximum number of frames the game will process per second.
- clock = pygame.time.Clock(): Creates a **Clock** object that helps manage time.
- clock.tick(FPS): Ensures the game **runs at a consistent speed** by limiting the loop to **FPS** frames per second.

Why Do We Need to Set a Frame Rate?

Without a controlled frame rate, the game loop would run as fast as possible, consuming unnecessary CPU power and causing **inconsistent movement speeds** across different computers.

Key Reasons to Limit FPS:

1. **Smooth Gameplay** – Prevents the game from running too fast or too slow on different machines.
2. **Reduces CPU & GPU Usage** – Without an FPS cap, the game would try to update as many times per second as possible, overloading the processor.
3. **Consistent Motion** – If objects in the game move based on the loop speed, an unlimited FPS could cause movement to be **too fast on powerful PCs** and **too slow on weaker ones**.

Here is a snippet of the Full code so far:

```
import pygame

# Game window

WIDTH, HEIGHT = 900, 500 # define the width and height of the game window
WIN = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Cosmic Duel") # set the title of the window

# Define colors
WHITE = (255, 255, 255) # define color white

# Define Frame Rate
FPS = 60 # set the frames per second

def draw_window(): # function to draw the game window
    WIN.fill(WHITE) # fill the window with white color
    pygame.display.update() # update the display


def main():
    clock = pygame.time.Clock() # create a clock object to control the frame rate

    # Main function to run the game
    run = True # variable to control the game loop
    while run:
        clock.tick(FPS) # set the frame rate

        # Event handling
```

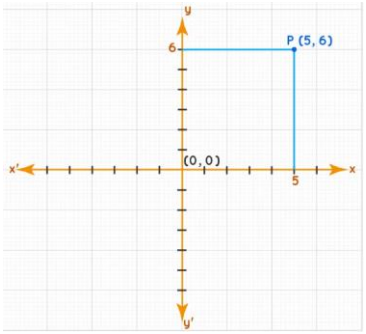
```
        for event in pygame.event.get(): # loop through all events
            # check for quit event
            if event.type == pygame.QUIT:
                run = False # exit the game loop
        draw_window()
        pygame.display.update() # update the display

    pygame.quit() # quit pygame when the game loop ends

if __name__ == "__main__":
    main() # call the main function to start the game
```

Step 5: Importing & drawing the images on the screen.

Understanding the Pygame Coordinate System

Before we import and draw images, it's important to understand how **coordinates** work in Pygame.

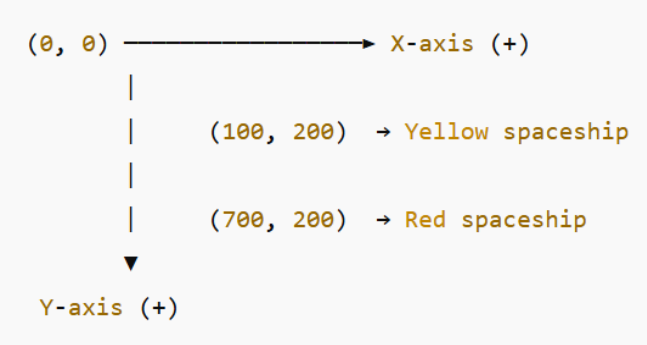
Normal math Cartesian plane looks like this.



Pygame uses a **Cartesian coordinate system** where:

- The **origin (0,0)** is at the **top-left corner** of the window.
- The **X-axis** increases **to the right**.
- The **Y-axis** increases **downwards** (opposite of standard graphs).

Visual Representation of the Pygame Coordinate System:



A point **(100, 200)** means:

- Move **100 pixels to the right** (X).
- Move **200 pixels down** (Y).

```
import os # import the os module for file path handling

# Importing images
YELLOW_SPACESHIP_IMAGE =
pygame.image.load(os.path.join('Assets','spaceship_yellow.png')) # load yellow
spaceship image
RED_SPACESHIP_IMAGE =
pygame.image.load(os.path.join('Assets','spaceship_red.png'))

def draw_window(): # function to draw the game window
    WIN.fill(WHITE) # fill the window with white color
    WIN.blit(YELLOW_SPACESHIP_IMAGE, (100, 200)) # draw the yellow spaceship at
specified coordinates
    WIN.blit(RED_SPACESHIP_IMAGE, (700, 200)) # draw the red spaceship at
specified coordinates
    pygame.display.update() # update the display
```

Explanation:

1. Importing the os Module

- The os module is used to handle file paths in a way that works across **different operating systems** (Windows, Mac, Linux).

2. Importing Images

- pygame.image.load() loads an image from a specified file path and returns a **Surface object**, which represents the image.
- We use os.path.join() to construct the file path correctly, making sure it works on **all operating systems**.
- If you load an image like this:

```
pygame.image.load('Assets/spaceship_yellow.png')
```
This may work on **Windows** but might **fail** on **Mac/Linux** due to differences in how file paths are structured.

- A **better way** is:
```
pygame.image.load(os.path.join('Assets', 'spaceship_yellow.png'))
```
This ensures **compatibility** across all operating systems.

3. Drawing Images on the Screen (blit())

- The **blit() function** is used to **draw images onto the game window**.
- It **copies** an image onto another surface, making it **efficient** for rendering sprites and game objects. Syntax: WIN.blit(image, (x, y))

4. Updating the Display

- pygame.display.update() is used to **refresh** the screen so that the newly drawn images appear.

Here is a snippet of the Full code so far:

```
import pygame
import os # import the os module for file path handling

# Game window

WIDTH, HEIGHT = 900, 500 # define the width and height of the game window
WIN = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Cosmic Duel") # set the title of the window

# Define colors
WHITE = (255, 255, 255) # define color white

# Define Frame Rate
FPS = 60 # set the frames per second

# Importing images
YELLOW_SPACESHIP_IMAGE =
pygame.image.load(os.path.join('Assets','spaceship_yellow.png')) # load yellow
spaceship image
RED_SPACESHIP_IMAGE =
pygame.image.load(os.path.join('Assets','spaceship_red.png'))

def draw_window(): # function to draw the game window
    WIN.fill(WHITE) # fill the window with white color
```

```
    WIN.blit(YELLOW_SPACESHIP_IMAGE, (100, 200)) # draw the yellow spaceship at
specified coordinates
    WIN.blit(RED_SPACESHIP_IMAGE, (700, 200)) # draw the red spaceship at
specified coordinates
    pygame.display.update() # update the display

def main():
    clock = pygame.time.Clock() # create a clock object to control the frame rate

    # Main function to run the game
    run = True # variable to control the game loop
    while run:
        clock.tick(FPS) # set the frame rate

        # Event handling
        for event in pygame.event.get(): # loop through all events
            # check for quit event
            if event.type == pygame.QUIT:
                run = False # exit the game loop
        draw_window()
        pygame.display.update() # update the display

    pygame.quit() # quit pygame when the game loop ends

if __name__ == "__main__":
    main() # call the main function to start the game
```

when you run the game, this is what you see on the screen.

Important Note: Drawing Order Matters

- The order in which we draw things on the screen is very important.
- In **draw_window(),** if you place the images before **WIN.fill(WHITE)**, the white background will be drawn on top of the images.
- This means the images will be covered, and you will only see a white screen instead of the spaceships**.**

Step 6: Transforming and rotating the images.

In this step, we **resize** and **rotate** the spaceship images to fit our game properly.

We use **pygame.transform.scale()** to resize the images and **pygame.transform.rotate()** to rotate them.

Syntax:

pygame.transform.scale(surface, (width, height))

pygame.transform.rotate(surface, angle)

Parameters:

- surface: The image we want to resize.
- (width, height): The new size of the image. 55, 40
- angle: The number of degrees to rotate the image **counterclockwise**. 270, 90

```
# Transforming & Rotating the images
YELLOW_SPACESHIP =
pygame.transform.rotate(pygame.transform.scale(YELLOW_SPACESHIP_IMAGE, (55, 40)),
270) # scale and rotate the yellow spaceship image
RED_SPACESHIP =
pygame.transform.rotate(pygame.transform.scale(RED_SPACESHIP_IMAGE, (55, 40)),
90) # scale and rotate the red spaceship image
```

We are going to store the width and height of the spaceship images in separate variables and then use these variables when resizing the images. This makes our code more flexible, as we can easily change the spaceship size in one place. Additionally, we have updated the draw_window() function to use YELLOW_SPACESHIP and RED_SPACESHIP instead of YELLOW_SPACESHIP_IMAGE and RED_SPACESHIP_IMAGE to ensure we are drawing the correctly scaled and rotated images. (check code below)

Here is the code snippet so far:

```
import pygame
import os # import the pygame library and os module for file path handling

# Game window settings
WIDTH, HEIGHT = 900, 500 # define the width and height of the game window
WIN = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Cosmic Duel") # set the title of the window

# Define colors
```

```
WHITE = (255, 255, 255) # define color white

# Define Frame Rate
FPS = 60 # set the frames per second

# Importing images
YELLOW_SPACESHIP_IMAGE =
pygame.image.load(os.path.join('Assets','spaceship_yellow.png')) # load yellow
spaceship image
RED_SPACESHIP_IMAGE =
pygame.image.load(os.path.join('Assets','spaceship_red.png'))

# Variables for spaceship dimensions
SPACESHIP_WIDTH, SPACESHIP_HEIGHT = 55, 40 # define the width and height of the
space background

# Transforming & Rotating the images
YELLOW_SPACESHIP =
pygame.transform.rotate(pygame.transform.scale(YELLOW_SPACESHIP_IMAGE,
(SPACESHIP_WIDTH, SPACESHIP_HEIGHT)), 270) # scale and rotate the yellow
spaceship image
RED_SPACESHIP =
pygame.transform.rotate(pygame.transform.scale(RED_SPACESHIP_IMAGE,
(SPACESHIP_WIDTH, SPACESHIP_HEIGHT)), 90) # scale and rotate the red spaceship
image

def draw_window(): # function to draw the game window
    WIN.fill(WHITE) # fill the window with white color
    WIN.blit(YELLOW_SPACESHIP, (100, 200)) # draw the yellow spaceship at
specified coordinates
    WIN.blit(RED_SPACESHIP, (700, 200)) # draw the red spaceship at specified
coordinates
    pygame.display.update() # update the display


def main():
    clock = pygame.time.Clock() # create a clock object to control the frame rate

    # Main function to run the game
    run = True # variable to control the game loop
    while run:
        clock.tick(FPS) # set the frame rate

        # Event handling
        for event in pygame.event.get(): # loop through all events
```

```
            # check for quit event
            if event.type == pygame.QUIT:
                run = False # exit the game loop
        draw_window()
        pygame.display.update() # update the display

    pygame.quit() # quit pygame when the game loop ends

if __name__ == "__main__":
    main() # call the main function to start the game
```

Step 7: Moving the images (spaceships) around the window.

To move the spaceships, we need a way to track their **x** and **y** positions. We will use pygame.Rect to create rectangles representing the spaceships. These rectangles store their **position (x, y)** and **size (width, height)**, allowing us to update their positions dynamically.

```
def draw_window(red, yellow): # function to draw the game window
    WIN.fill(WHITE) # fill the window with white color
    WIN.blit(YELLOW_SPACESHIP, (yellow.x,yellow.y)) # draw the yellow spaceship
at specified coordinates
    WIN.blit(RED_SPACESHIP, (red.x, red.y)) # draw the red spaceship at specified
coordinates
    pygame.display.update() # update the display


def main():
    red = pygame.Rect(700, 300, SPACESHIP_WIDTH, SPACESHIP_HEIGHT) # create a
rectangle for the red spaceship
    yellow = pygame.Rect(100, 300, SPACESHIP_WIDTH, SPACESHIP_HEIGHT) # create a
rectangle for the yellow spaceship

    clock = pygame.time.Clock() # create a clock object to control the frame rate

    # Main function to run the game
    run = True # variable to control the game loop
    while run:
        clock.tick(FPS) # set the frame rate

        # Event handling
        for event in pygame.event.get(): # loop through all events
            # check for quit event
            if event.type == pygame.QUIT:
                run = False # exit the game loop
```

```
        yellow.x += 1 # move the yellow spaceship to the right

        draw_window(red , yellow) # call the draw_window function to draw the
game window
        pygame.display.update() # update the display

    pygame.quit() # quit pygame when the game loop ends
```

Understanding the Changes

1. **Using pygame.Rect for Spaceships**
    - We create a pygame.Rect object for each spaceship (red and yellow).
    - These rectangles hold the position (x, y) and size (width, height) of each spaceship.

```
    yellow = pygame.Rect(100, 300, SPACESHIP_WIDTH, SPACESHIP_HEIGHT)
```

This places the yellow spaceship at **(100, 300)** with the defined width and height.

2. **Modifying the draw_window() Function**
    - Instead of using fixed coordinates, we now use yellow.x, yellow.y and red.x, red.y to dynamically place the spaceships.
    - This allows movement by simply updating these values.
3. **Moving the Yellow Spaceship**
    - Inside the game loop, we increment yellow.x by **1** in each frame:

```
    yellow.x += 1
```

    - This moves the yellow spaceship **to the right** every frame.
4. **Calling draw_window(red, yellow)**
    - We pass both spaceship rectangles to draw_window() so they are drawn at their updated positions.

Allow player-controlled movement

we will allow **player-controlled movement** using keyboard input, so players can move their spaceships freely. The code below focuses on the movement of the yellow spaceship.

```python
# velocity variables for the spaceships
VEL = 5


def draw_window(red, yellow): # function to draw the game window
    WIN.fill(WHITE) # fill the window with white color
    WIN.blit(YELLOW_SPACESHIP, (yellow.x,yellow.y)) # draw the yellow spaceship
at specified coordinates
    WIN.blit(RED_SPACESHIP, (red.x, red.y)) # draw the red spaceship at specified
coordinates
    pygame.display.update() # update the display


def main():
    red = pygame.Rect(700, 300, SPACESHIP_WIDTH, SPACESHIP_HEIGHT) # create a
rectangle for the red spaceship
    yellow = pygame.Rect(100, 300, SPACESHIP_WIDTH, SPACESHIP_HEIGHT) # create a
rectangle for the yellow spaceship

    clock = pygame.time.Clock() # create a clock object to control the frame rate

    # Main function to run the game
    run = True # variable to control the game loop
    while run:
        clock.tick(FPS) # set the frame rate

        # Event handling
        for event in pygame.event.get(): # loop through all events
            # check for quit event
            if event.type == pygame.QUIT:
                run = False # exit the game loop

         # Movement handling
        keys = pygame.key.get_pressed() # get the state of all keys
        if keys[pygame.K_a]: # move yellow spaceship left
            yellow.x -= VEL
        if keys[pygame.K_d]:
            yellow.x += VEL # move yellow spaceship right
        if keys[pygame.K_w]: # move yellow spaceship up
            yellow.y -= VEL
        if keys[pygame.K_s]: # move yellow spaceship down
            yellow.y += VEL

        draw_window(red , yellow) # call the draw_window function to draw the
game window
```

```
        pygame.display.update() # update the display

    pygame.quit() # quit pygame when the game loop ends

if __name__ == "__main__":
    main() # call the main function to start the game
```

Understanding the Changes

1. **Adding a Velocity Variable (VEL)**
    - VEL = 5 sets the speed of movement.
    - This determines how many pixels the spaceship moves per frame.
2. **Checking Keyboard Input (pygame.key.get_pressed())**
    - We use pygame.key.get_pressed() to check which keys are currently pressed.
    - This returns a list of all keys, where True means a key is pressed.
3. **Handling Movement Controls**
    - **A Key (K_a) → Move Left**

```
    if keys[pygame.K_a]: # move yellow spaceship left
            yellow.x -= VEL
```

    - **D Key (K_d) → Move Right**

```
    if keys[pygame.K_d]:
            yellow.x += VEL # move yellow spaceship right
```

To handle the movement of the red spaceship just add this code.

```
# check for red spaceship movement
        if keys[pygame.K_LEFT]:
            red.x -= VEL # move red spaceship left
        if keys[pygame.K_RIGHT]:
            red.x += VEL # move red spaceship right
        if keys[pygame.K_UP]: # move red spaceship up
            red.y -= VEL
        if keys[pygame.K_DOWN]: # move red spaceship down
            red.y += VEL
```

To clean up our code we are going to put the yellow and red spaceship movement handlings in separate functions. Check code below.

Here is a snippet of the code so far.

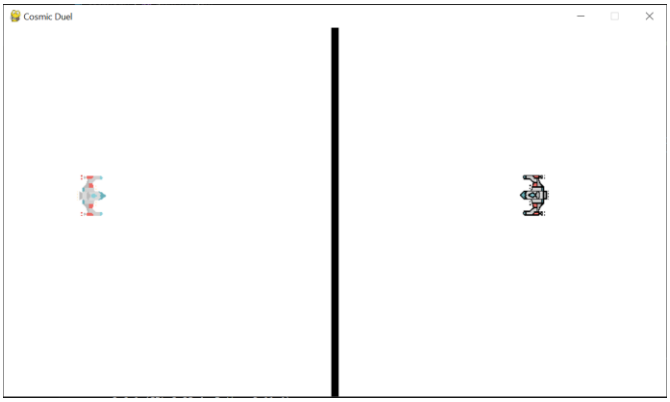
```
import pygame
import os # import the pygame library and os module for file path handling

# Game window settings
WIDTH, HEIGHT = 900, 500 # define the width and height of the game window
WIN = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Cosmic Duel") # set the title of the window

# Define colors
WHITE = (255, 255, 255) # define color white

# Define Frame Rate
FPS = 60 # set the frames per second

# Importing images
YELLOW_SPACESHIP_IMAGE =
pygame.image.load(os.path.join('Assets','spaceship_yellow.png')) # load yellow
spaceship image
RED_SPACESHIP_IMAGE =
pygame.image.load(os.path.join('Assets','spaceship_red.png'))

# Variables for spaceship dimensions
SPACESHIP_WIDTH, SPACESHIP_HEIGHT = 55, 40 # define the width and height of the
space background

# Transforming & Rotating the images
YELLOW_SPACESHIP =
pygame.transform.rotate(pygame.transform.scale(YELLOW_SPACESHIP_IMAGE,
(SPACESHIP_WIDTH, SPACESHIP_HEIGHT)), 270) # scale and rotate the yellow
spaceship image
RED_SPACESHIP =
pygame.transform.rotate(pygame.transform.scale(RED_SPACESHIP_IMAGE,
(SPACESHIP_WIDTH, SPACESHIP_HEIGHT)), 90) # scale and rotate the red spaceship
image

# velocity variables for the spaceships
VEL = 5


def draw_window(red, yellow): # function to draw the game window
    WIN.fill(WHITE) # fill the window with white color
    WIN.blit(YELLOW_SPACESHIP, (yellow.x,yellow.y)) # draw the yellow spaceship
at specified coordinates
```

```
    WIN.blit(RED_SPACESHIP, (red.x, red.y)) # draw the red spaceship at specified
coordinates
    pygame.display.update() # update the display

def handle_yellow_movement(keys, yellow): # function to handle the movement of
the yellow spaceship
     # check for yellow spaceship movement
        if keys[pygame.K_a]: # move yellow spaceship left
            yellow.x -= VEL
        if keys[pygame.K_d]:
            yellow.x += VEL # move yellow spaceship right
        if keys[pygame.K_w]: # move yellow spaceship up
            yellow.y -= VEL
        if keys[pygame.K_s]: # move yellow spaceship down
            yellow.y += VEL

def handle_red_movement(keys, red): # function to handle the movement of the red
spaceship
    # check for red spaceship movement
        if keys[pygame.K_LEFT]: # move red spaceship left
            red.x -= VEL
        if keys[pygame.K_RIGHT]: # move red spaceship right
            red.x += VEL
        if keys[pygame.K_UP]: # move red spaceship up
            red.y -= VEL
        if keys[pygame.K_DOWN]: # move red spaceship down
            red.y += VEL
def main():
    red = pygame.Rect(700, 300, SPACESHIP_WIDTH, SPACESHIP_HEIGHT) # create a
rectangle for the red spaceship
    yellow = pygame.Rect(100, 300, SPACESHIP_WIDTH, SPACESHIP_HEIGHT) # create a
rectangle for the yellow spaceship

    clock = pygame.time.Clock() # create a clock object to control the frame rate

    # Main function to run the game
    run = True # variable to control the game loop
    while run:
        clock.tick(FPS) # set the frame rate

        # Event handling
        for event in pygame.event.get(): # loop through all events
            # check for quit event
            if event.type == pygame.QUIT:
                run = False # exit the game loop
```

```
        # Movement handling
        keys = pygame.key.get_pressed() # get the state of all keys
        handle_yellow_movement(keys, yellow) # call the function
        handle_red_movement(keys, red) # call the function
        draw_window(red , yellow) # call the function
        pygame.display.update() # update the display

    pygame.quit() # quit pygame when the game loop ends

if __name__ == "__main__":
    main() # call the main function to start the game
```

Step 8: Drawing a Border to Split the Screen

In this step, we add a **black border** in the middle of the screen to separate the left and right players (yellow and red spaceships). This helps visually divide the play area and ensures that each player has their own side.




```
# Define colors
WHITE = (255, 255, 255) # define color white
BLACK = (0, 0, 0) # define color black

# black border in the middle of the window
BORDER = pygame.Rect(WIDTH//2 - 5, 0, 10, HEIGHT) # create a rectangle for the
border in the middle of the window

# Define Frame Rate
FPS = 60 # set the frames per second

# Importing images
```

```
YELLOW_SPACESHIP_IMAGE =
pygame.image.load(os.path.join('Assets','spaceship_yellow.png')) # load yellow
spaceship image
RED_SPACESHIP_IMAGE =
pygame.image.load(os.path.join('Assets','spaceship_red.png'))

# Variables for spaceship dimensions
SPACESHIP_WIDTH, SPACESHIP_HEIGHT = 55, 40 # define the width and height of the
space background

# Transforming & Rotating the images
YELLOW_SPACESHIP =
pygame.transform.rotate(pygame.transform.scale(YELLOW_SPACESHIP_IMAGE,
(SPACESHIP_WIDTH, SPACESHIP_HEIGHT)), 270) # scale and rotate the yellow
spaceship image
RED_SPACESHIP =
pygame.transform.rotate(pygame.transform.scale(RED_SPACESHIP_IMAGE,
(SPACESHIP_WIDTH, SPACESHIP_HEIGHT)), 90) # scale and rotate the red spaceship
image

# velocity variables for the spaceships
VEL = 5

def draw_window(red, yellow): # function to draw the game window
    WIN.fill(WHITE) # fill the window with white color
    pygame.draw.rect(WIN, BLACK, BORDER) # draw the border in the middle of the
window
    WIN.blit(YELLOW_SPACESHIP, (yellow.x,yellow.y)) # draw the yellow spaceship
at specified coordinates
    WIN.blit(RED_SPACESHIP, (red.x, red.y)) # draw the red spaceship at specified
coordinates
    pygame.display.update() # update the display
```

Understanding the Changes

1. **Defining the Border (BORDER)**
    - We create a **rectangle** that represents the border using pygame.Rect().

- **Syntax:** pygame.Rect(x, y, width, height)

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

- This makes the border **10 pixels wide** and as tall as the screen.

Why Use WIDTH // 2 – 5 as the value for x?

Let's break it down:

- WIDTH // 2 → Finds the **middle** of the screen.
- - 5 → Moves the border **5 pixels left** to center it properly.
- y = 0 → The border starts from the **top of the screen**.
- 10 → The **width of the border**.

Visual Representation

Imagine the screen is 900 pixels wide (`WIDTH = 900`):

```
0 px              450 px              900 px
|-----------------/-----------------/
(left side)     (middle)          (right side)
```

By default, if we place the border at WIDTH // 2, it would start at **450 px**. Since the border is **10 pixels wide**, it would cover 450 px to 460 px. This means the center of the border is actually **455 px**. To **center it properly**, we move it **5 pixels left** (-5), so it starts at **445 px** and covers **445 px to 455 px**:

```
0 px              445 px    455 px              900 px
|-----------------/▓▓▓▓▓▓▓/-----------------/
(left side)     (border)       (right side)
```

2. Drawing the Border in draw_window()

- **pygame.draw.rect(WIN, BLACK, BORDER)** fills the rectangle with black color.
- **`WIN`** → The window (where the border is drawn).
- **`BLACK`** → The color of the border.
- **`BORDER`** → The rectangle we defined (`pygame.Rect(x, y, width, height)`).

why did we use pygame.draw.rect () instead of win.blit () ?

blit() is used for drawing images (surfaces), while **draw.rect()** is used for drawing shapes like rectangles.

Step 9: Prevent Spaceships from Moving Outside the Game Window and Past the Middle Screen.

```
def handle_yellow_movement(keys, yellow): # function to handle the movement of
the yellow spaceship
     # check for yellow spaceship movement
        if keys[pygame.K_a] and yellow.x - VEL > 0:  # Move left, stop at the
left edge
            yellow.x -= VEL
        if keys[pygame.K_d] and yellow.x + VEL + yellow.width < BORDER.x:  # Move
right, stop at the border
            yellow.x += VEL
        if keys[pygame.K_w] and yellow.y - VEL > 0:  # Move up, stop at the top
edge
            yellow.y -= VEL
        if keys[pygame.K_s] and yellow.y + VEL + yellow.height < HEIGHT - 10:  #
Move down, stop at bottom edge
            yellow.y += VEL

def handle_red_movement(keys, red): # function to handle the movement of the red
spaceship
    # check for red spaceship movement
        if keys[pygame.K_LEFT] and red.x - VEL > BORDER.x + BORDER.width:  # Move
left, stop at the border
            red.x -= VEL
        if keys[pygame.K_RIGHT] and red.x + VEL + red.width < WIDTH:  # Move
right, stop at right edge
            red.x += VEL
        if keys[pygame.K_UP] and red.y - VEL > 0:  # Move up, stop at the top
edge
            red.y -= VEL
        if keys[pygame.K_DOWN] and red.y + VEL + red.height < HEIGHT - 10:  #
Move down, stop at the bottom edge
            red.y += VEL
```

Condition	Purpose
yellow.x - VEL > 0	Prevents the yellow spaceship from moving past the left edge.
yellow.x + VEL + yellow.width < BORDER.x	Stops the yellow spaceship at the middle border.
red.x - VEL > BORDER.x + BORDER.width	Stops the red spaceship at the middle border.
red.x + VEL + red.width < WIDTH	Prevents the red spaceship from moving past the right edge.
yellow.y - VEL > 0 / red.y - VEL > 0	Prevents both spaceships from moving above the screen.
yellow.y + VEL + yellow.height < HEIGHT - 10 / red.y + VEL + red.height < HEIGHT - 10	Prevents both spaceships from moving below the screen.

Step 10: Implementing Bullet Firing Mechanism

To allow spaceships to fire projectiles, we need to define a mechanism to shoot bullets when a specific key is pressed. This includes setting up a list to store bullets, creating bullet objects upon key press, and ensuring they move in the correct direction.

1. **Define Bullet Properties**
    - Define bullet velocity (BULLET_VEL = 7).
    - Limit the number of bullets a spaceship can fire at a time (MAX_BULLET = 5).
    - Define bullet dimensions (BULLET_WIDTH = 10, BULLET_HEIGHT = 5).
2. **Handling Bullet Firing**
    - Listen for key press events (KEYDOWN).
    - If the K_LCTRL key is pressed and the yellow spaceship has fewer than MAX_BULLET, create a bullet at the spaceship's position and add it to the yellow_bullets list.
    - If the K_RCTRL key is pressed and the red spaceship has fewer than MAX_BULLET, create a bullet at the spaceship's position and add it to the red_bullets list.
    - Remove bullets that go out of bounds.

Step 11: Handling Bullet Movement and Collisions

Once bullets are fired, they should move across the screen and be able to collide with the opposing spaceship.

1. **Bullet Movement**
    - Loop through all bullets in yellow_bullets and move them to the right.
    - Loop through all bullets in red_bullets and move them to the left.
2. **Collision Detection**
    - Check if a bullet from yellow_bullets collides with the red spaceship using colliderect().
    - If a collision occurs, trigger the RED_HIT event and remove the bullet.
    - Check if a bullet from red_bullets collides with the yellow spaceship.
    - If a collision occurs, trigger the YELLOW_HIT event and remove the bullet.
3. **Updating the Game State**
    - Continuously update bullet positions.
    - Call handle_bullets(yellow_bullets, red_bullets, yellow, red) inside the game loop.
    - Update the display using draw_window() to render the bullets along with the spaceships.

Here is the code snippet so far. I have highlighted in blue the new code entered.

```
import pygame
import os # import the pygame library and os module for file path handling

# Game window settings
WIDTH, HEIGHT = 900, 500 # define the width and height of the game window
WIN = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Cosmic Duel") # set the title of the window

# Define colors
WHITE = (255, 255, 255) # define color white
BLACK = (0, 0, 0) # define color black
RED = (255, 0, 0) # define color red
YELLOW = (255, 255, 0) # define color yellow

# black border in the middle of the window
BORDER = pygame.Rect(WIDTH//2 - 5, 0, 10, HEIGHT) # create a rectangle for the
border in the middle of the window
```

```
# Define Frame Rate
FPS = 60 # set the frames per second

# Importing images
YELLOW_SPACESHIP_IMAGE =
pygame.image.load(os.path.join('Assets','spaceship_yellow.png')) # load yellow
spaceship image
RED_SPACESHIP_IMAGE =
pygame.image.load(os.path.join('Assets','spaceship_red.png'))

# Variables for spaceship dimensions
SPACESHIP_WIDTH, SPACESHIP_HEIGHT = 55, 40 # define the width and height of the
space background

# Transforming & Rotating the images
YELLOW_SPACESHIP =
pygame.transform.rotate(pygame.transform.scale(YELLOW_SPACESHIP_IMAGE,
(SPACESHIP_WIDTH, SPACESHIP_HEIGHT)), 270) # scale and rotate the yellow
spaceship image
RED_SPACESHIP =
pygame.transform.rotate(pygame.transform.scale(RED_SPACESHIP_IMAGE,
(SPACESHIP_WIDTH, SPACESHIP_HEIGHT)), 90) # scale and rotate the red spaceship
image

# velocity variables for the spaceships
VEL = 5

# Bullets
BULLET_VEL = 7 # set the velocity of the bullets
MAx_BULLET = 5 # set the maximum velocity of the bullets
BULLET_WIDTH, BULLET_HEIGHT = 10, 5 # define the width and height of the bullets

# Define events for bullet hit detection
YELLOW_HIT = pygame.USEREVENT + 1 # define a custom event for yellow spaceship
hit
RED_HIT = pygame.USEREVENT + 2 # define a custom event for red spaceship hit

def draw_window(red, yellow, red_bullets,yellow_bullets): # function to draw the
game window
    WIN.fill(WHITE) # fill the window with white color
    pygame.draw.rect(WIN, BLACK, BORDER) # draw the border in the middle of the
window
    WIN.blit(YELLOW_SPACESHIP, (yellow.x,yellow.y)) # draw the yellow spaceship
at specified coordinates
```

```
    WIN.blit(RED_SPACESHIP, (red.x, red.y)) # draw the red spaceship at specified
coordinates

    # draw the bullets
    for bullet in yellow_bullets: # loop through all yellow bullets
        pygame.draw.rect(WIN, YELLOW, bullet) # draw yellow bullets
    for bullet in red_bullets: # loop through all red bullets
        pygame.draw.rect(WIN, RED, bullet) # draw red bullets

    pygame.display.update() # update the display

def handle_yellow_movement(keys, yellow): # function to handle the movement of
the yellow spaceship
     # check for yellow spaceship movement
        if keys[pygame.K_a] and yellow.x - VEL > 0:  # Move left, stop at the
left edge
            yellow.x -= VEL
        if keys[pygame.K_d] and yellow.x + VEL + yellow.width < BORDER.x:  # Move
right, stop at the border
            yellow.x += VEL
        if keys[pygame.K_w] and yellow.y - VEL > 0:  # Move up, stop at the top
edge
            yellow.y -= VEL
        if keys[pygame.K_s] and yellow.y + VEL + yellow.height < HEIGHT - 10:  #
Move down, stop at bottom edge
            yellow.y += VEL

def handle_red_movement(keys, red): # function to handle the movement of the red
spaceship
    # check for red spaceship movement
        if keys[pygame.K_LEFT] and red.x - VEL > BORDER.x + BORDER.width:  # Move
left, stop at the border
            red.x -= VEL
        if keys[pygame.K_RIGHT] and red.x + VEL + red.width < WIDTH:  # Move
right, stop at right edge
            red.x += VEL
        if keys[pygame.K_UP] and red.y - VEL > 0:  # Move up, stop at the top
edge
            red.y -= VEL
        if keys[pygame.K_DOWN] and red.y + VEL + red.height < HEIGHT - 10:  #
Move down, stop at the bottom edge
            red.y += VEL

def handle_bullets(yellow_bullets, red_bullets, yellow, red): # function to
handle bullet movement and collision
```

```
    for bullet in yellow_bullets: # loop through all yellow bullets
        bullet.x += BULLET_VEL # move the bullet to the right
        if red.colliderect(bullet):
            pygame.event.post(pygame.event.Event(RED_HIT)) # post a custom event
if the red spaceship is hit
            yellow_bullets.remove(bullet) # remove the bullet if it collides with
the red spaceship

    for bullet in red_bullets:
        bullet.x -= BULLET_VEL # move the bullet to the left
        if yellow.colliderect(bullet):
            pygame.event.post(pygame.event.Event(YELLOW_HIT)) # post a custom
event if the yellow spaceship is hit
            red_bullets.remove(bullet) # remove the bullet if it collides with
the yellow spaceship

def main():
    red = pygame.Rect(700, 200, SPACESHIP_WIDTH, SPACESHIP_HEIGHT) # create a
rectangle for the red spaceship
    yellow = pygame.Rect(100, 200, SPACESHIP_WIDTH, SPACESHIP_HEIGHT) # create a
rectangle for the yellow spaceship
    yellow_bullets = [] # list to store bullets
    red_bullets = [] # list to store bullets
    clock = pygame.time.Clock() # create a clock object to control the frame rate

    # Main function to run the game
    run = True # variable to control the game loop
    while run:
        clock.tick(FPS) # set the frame rate

        # Event handling
        for event in pygame.event.get(): # loop through all events
            # check for quit event
            if event.type == pygame.QUIT:
                run = False # exit the game loop

            # Handle bullet firing
            if event.type == pygame.KEYDOWN:
                # yellow spaceship firing bullets with left control key
                if event.key == pygame.K_LCTRL and len(yellow_bullets) <
MAx_BULLET: # check if the left control key is pressed
                    if len(yellow_bullets) < 5: # limit the number of bullets to
5
                        bullet = pygame.Rect(yellow.x + yellow.width, yellow.y +
yellow.height//2 - BULLET_HEIGHT//2, BULLET_WIDTH, BULLET_HEIGHT)
```

```
                        yellow_bullets.append(bullet)
                    elif bullet.x > WIDTH: # remove the bullet if it goes out of
bounds
                        yellow_bullets.remove(bullet)

                # red spaceship firing bullets with right control key
                if event.key == pygame.K_RCTRL and len(red_bullets) < MAx_BULLET:
                    if len(red_bullets) < 5:
                        bullet = pygame.Rect(red.x, red.y + red.height//2 -
BULLET_HEIGHT//2, BULLET_WIDTH, BULLET_HEIGHT)
                        red_bullets.append(bullet)
                    elif bullet.x < 0: # remove the bullet if it goes out of
bounds
                        red_bullets.remove(bullet)



         # Movement handling
        keys = pygame.key.get_pressed() # get the state of all keys
        handle_yellow_movement(keys, yellow) # call the function to handle yellow
spaceship movement
        handle_red_movement(keys, red) # call the function to handle red
spaceship movement

        handle_bullets(yellow_bullets, red_bullets, yellow, red) # call the
function to handle bullet movement and collision
        draw_window(red , yellow, red_bullets, yellow_bullets) # call the
draw_window function to draw the game window


    pygame.quit() # quit pygame when the game loop ends

if __name__ == "__main__":
    main() # call the main function to start the game
```

Step 12: Loading a background image

Previously, the game window had a plain **white background**. Now, we will replace it with a **space-themed image** to enhance the visual appeal of the game.

- Before we can use the image, we need to **load it** and **resize** it to fit the game window dimensions.
  SPACE_BACKGROUND = pygame.image.load(os.path.join('Assets', 'space.png'))
  SPACE_BACKGROUND = pygame.transform.scale(SPACE_BACKGROUND, (WIDTH, HEIGHT))
- Draw the Background Image in the Game Window. In the draw_window () function
  We replaced WIN.fill(WHITE) (# Old code: Fills the screen with white) with
  WIN.blit(SPACE_BACKGROUND, (0, 0)) # New code: Draws the background image

```
# Importing images
YELLOW_SPACESHIP_IMAGE =
pygame.image.load(os.path.join('Assets','spaceship_yellow.png')) # load yellow
spaceship image
RED_SPACESHIP_IMAGE =
pygame.image.load(os.path.join('Assets','spaceship_red.png'))
SPACE_BACKGROUND = pygame.image.load(os.path.join('Assets', space_background.jpg
')) # load space background image

# Variables for spaceship dimensions
SPACESHIP_WIDTH, SPACESHIP_HEIGHT = 55, 40 # define the width and height of the
space background

# Transforming & Rotating the images
YELLOW_SPACESHIP =
pygame.transform.rotate(pygame.transform.scale(YELLOW_SPACESHIP_IMAGE,
(SPACESHIP_WIDTH, SPACESHIP_HEIGHT)), 270) # scale and rotate the yellow
spaceship image
RED_SPACESHIP =
pygame.transform.rotate(pygame.transform.scale(RED_SPACESHIP_IMAGE,
(SPACESHIP_WIDTH, SPACESHIP_HEIGHT)), 90) # scale and rotate the red spaceship
image
SPACE_BACKGROUND = pygame.transform.scale(SPACE_BACKGROUND, (WIDTH, HEIGHT)) #
scale the space background image to fit the window

# velocity variables for the spaceships
VEL = 5

# Bullets
BULLET_VEL = 7 # set the velocity of the bullets
MAx_BULLET = 5 # set the maximum velocity of the bullets
BULLET_WIDTH, BULLET_HEIGHT = 10, 5 # define the width and height of the bullets
```

```
# Define events for bullet hit detection
YELLOW_HIT = pygame.USEREVENT + 1 # define a custom event for yellow spaceship
hit
RED_HIT = pygame.USEREVENT + 2 # define a custom event for red spaceship hit

def draw_window(red, yellow, red_bullets,yellow_bullets): # function to draw the
game window
    WIN.blit(SPACE_BACKGROUND, (0, 0)) # draw the space background at coordinates
(0, 0)
    pygame.draw.rect(WIN, BLACK, BORDER) # draw the border in the middle of the
window
    WIN.blit(YELLOW_SPACESHIP, (yellow.x,yellow.y)) # draw the yellow spaceship
at specified coordinates
    WIN.blit(RED_SPACESHIP, (red.x, red.y)) # draw the red spaceship at specified
coordinates

    # draw the bullets
    for bullet in yellow_bullets: # loop through all yellow bullets
        pygame.draw.rect(WIN, YELLOW, bullet) # draw yellow bullets
    for bullet in red_bullets: # loop through all red bullets
        pygame.draw.rect(WIN, RED, bullet) # draw red bullets

    pygame.display.update() # update the display
```

Step 13: Implementing a Horizontally Scrolling Background

In this step, horizontal scrolling for the background has been implemented to create a dynamic space environment. This enhances the visual experience by making it appear as though the spaceships are moving through an endless space scene.

1. **Background Image Adjustment:**
    - The background image has been resized to be significantly larger than the game window. This allows for a seamless scrolling effect when shifting the background horizontally.
    - SPACE_BACKGROUND is scaled to (5000, 3000), ensuring the image covers a wide range before looping.
2. **Background Movement Implementation:**
    - A new variable bg_x is introduced to track the horizontal position of the background.
    - A velocity variable BG_VEL = 0.5 controls the speed at which the background scrolls.
    - In the game loop, bg_x is decremented to shift the background leftward.

- When bg_x reaches -900, it resets to 0 to create an illusion of continuous movement.

3. **Modifications in draw_window() Function:**
    - The background is drawn twice: once at (bg_x, 0) and again at (bg_x + WIDTH, 0). This ensures a smooth transition when resetting the background.
    - This technique prevents abrupt visual gaps and maintains the illusion of seamless scrolling.

Code Changes

Initializing Background Movement Variables:

```
def main():
    bg_x = 0  # Initial Y position of the background
    BG_VEL = 0.5 # Speed at which the background moves down
```

Updating Background Position in main() Loop:

```
while run:
        clock.tick(FPS) # set the frame rate

        # Handle background movement
        bg_x -= BG_VEL
        if bg_x <= - 900:  # Reset the background position when it moves out of
the window
            bg_x = 0
```

Modifications in draw_window() to Display Scrolling Background:

```
def draw_window(red, yellow, red_bullets,yellow_bullets, bg_x): # function to
draw the game window
    WIN.blit(SPACE_BACKGROUND, (bg_x, 0)) # draw the space background at
coordinates (0, 0)
    WIN.blit(SPACE_BACKGROUND, (bg_x + WIDTH, 0)) # draw the space background
again to create a scrolling effect
```

Here is a snippet of the full code so far. I have highlighted in blue the changes in the code.

```
import pygame
import os # import the pygame library and os module for file path handling

# Game window settings
WIDTH, HEIGHT = 920, 500 # define the width and height of the game window
WIN = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Cosmic Duel") # set the title of the window

# Define colors
WHITE = (255, 255, 255) # define color white
BLACK = (0, 0, 0) # define color black
RED = (255, 0, 0) # define color red
YELLOW = (255, 255, 0) # define color yellow

# black border in the middle of the window
BORDER = pygame.Rect(WIDTH//2 - 5, 0, 10, HEIGHT) # create a rectangle for the
border in the middle of the window

# Define Frame Rate
FPS = 60 # set the frames per second

# Importing images
YELLOW_SPACESHIP_IMAGE =
pygame.image.load(os.path.join('Assets','spaceship_yellow.png')) # load yellow
spaceship image
RED_SPACESHIP_IMAGE =
pygame.image.load(os.path.join('Assets','spaceship_red.png'))
SPACE_BACKGROUND =
pygame.image.load(os.path.join('Assets','space_background.jpg')) # load space
background image

# Variables for spaceship dimensions
SPACESHIP_WIDTH, SPACESHIP_HEIGHT = 55, 40 # define the width and height of the
space background

# Transforming & Rotating the images
YELLOW_SPACESHIP =
pygame.transform.rotate(pygame.transform.scale(YELLOW_SPACESHIP_IMAGE,
(SPACESHIP_WIDTH, SPACESHIP_HEIGHT)), 270) # scale and rotate the yellow
spaceship image
RED_SPACESHIP =
pygame.transform.rotate(pygame.transform.scale(RED_SPACESHIP_IMAGE,
```

```
(SPACESHIP_WIDTH, SPACESHIP_HEIGHT)), 90) # scale and rotate the red spaceship
image
SPACE_BACKGROUND = pygame.transform.scale(SPACE_BACKGROUND, (5000, 3000)) # scale
the space background image to fit the window

# velocity variables for the spaceships
VEL = 5

# Bullets
BULLET_VEL = 7 # set the velocity of the bullets
MAx_BULLET = 5 # set the maximum velocity of the bullets
BULLET_WIDTH, BULLET_HEIGHT = 10, 5 # define the width and height of the bullets

# Define events for bullet hit detection
YELLOW_HIT = pygame.USEREVENT + 1 # define a custom event for yellow spaceship
hit
RED_HIT = pygame.USEREVENT + 2 # define a custom event for red spaceship hit

def draw_window(red, yellow, red_bullets,yellow_bullets, bg_x): # function to
draw the game window
    WIN.blit(SPACE_BACKGROUND, (bg_x, 0)) # draw the space background at
coordinates (0, 0)
    WIN.blit(SPACE_BACKGROUND, (bg_x + WIDTH, 0)) # draw the space background
again to create a scrolling effect

    pygame.draw.rect(WIN, BLACK, BORDER) # draw the border in the middle of the
window
    WIN.blit(YELLOW_SPACESHIP, (yellow.x,yellow.y)) # draw the yellow spaceship
at specified coordinates
    WIN.blit(RED_SPACESHIP, (red.x, red.y)) # draw the red spaceship at specified
coordinates

    # draw the bullets
    for bullet in yellow_bullets: # loop through all yellow bullets
        pygame.draw.rect(WIN, YELLOW, bullet) # draw yellow bullets
    for bullet in red_bullets: # loop through all red bullets
        pygame.draw.rect(WIN, RED, bullet) # draw red bullets

    pygame.display.update() # update the display

def handle_yellow_movement(keys, yellow): # function to handle the movement of
the yellow spaceship
     # check for yellow spaceship movement
        if keys[pygame.K_a] and yellow.x - VEL > 0:  # Move left, stop at the
left edge
```

```
            yellow.x -= VEL
        if keys[pygame.K_d] and yellow.x + VEL + yellow.width < BORDER.x:  # Move
right, stop at the border
            yellow.x += VEL
        if keys[pygame.K_w] and yellow.y - VEL > 0:  # Move up, stop at the top
edge
            yellow.y -= VEL
        if keys[pygame.K_s] and yellow.y + VEL + yellow.height < HEIGHT - 10:  #
Move down, stop at bottom edge
            yellow.y += VEL

def handle_red_movement(keys, red): # function to handle the movement of the red
spaceship
    # check for red spaceship movement
        if keys[pygame.K_LEFT] and red.x - VEL > BORDER.x + BORDER.width:  # Move
left, stop at the border
            red.x -= VEL
        if keys[pygame.K_RIGHT] and red.x + VEL + red.width < WIDTH:  # Move
right, stop at right edge
            red.x += VEL
        if keys[pygame.K_UP] and red.y - VEL > 0:  # Move up, stop at the top
edge
            red.y -= VEL
        if keys[pygame.K_DOWN] and red.y + VEL + red.height < HEIGHT - 10:  #
Move down, stop at the bottom edge
            red.y += VEL

def handle_bullets(yellow_bullets, red_bullets, yellow, red): # function to
handle bullet movement and collision
    for bullet in yellow_bullets: # loop through all yellow bullets
        bullet.x += BULLET_VEL # move the bullet to the right
        if red.colliderect(bullet):
            pygame.event.post(pygame.event.Event(RED_HIT)) # post a custom event
if the red spaceship is hit
            yellow_bullets.remove(bullet) # remove the bullet if it collides with
the red spaceship

    for bullet in red_bullets:
        bullet.x -= BULLET_VEL # move the bullet to the left
        if yellow.colliderect(bullet):
            pygame.event.post(pygame.event.Event(YELLOW_HIT)) # post a custom
event if the yellow spaceship is hit
            red_bullets.remove(bullet) # remove the bullet if it collides with
the yellow spaceship
```

```
def main():
    bg_x = 0  # Initial Y position of the background
    BG_VEL = 0.5 # Speed at which the background moves down

    red = pygame.Rect(700, 200, SPACESHIP_WIDTH, SPACESHIP_HEIGHT) # create a
rectangle for the red spaceship
    yellow = pygame.Rect(100, 200, SPACESHIP_WIDTH, SPACESHIP_HEIGHT) # create a
rectangle for the yellow spaceship
    yellow_bullets = [] # list to store bullets
    red_bullets = [] # list to store bullets
    clock = pygame.time.Clock() # create a clock object to control the frame rate

    # Main function to run the game
    run = True # variable to control the game loop
    while run:
        clock.tick(FPS) # set the frame rate

        # Handle background movement
        bg_x -= BG_VEL
        if bg_x <= - 900:  # Reset the background position when it moves out of
the window
            bg_x = 0

        # Event handling
        for event in pygame.event.get(): # loop through all events
            # check for quit event
            if event.type == pygame.QUIT:
                run = False # exit the game loop

            # Handle bullet firing
            if event.type == pygame.KEYDOWN:
                # yellow spaceship firing bullets with left control key
                if event.key == pygame.K_LCTRL and len(yellow_bullets) <
MAx_BULLET: # check if the left control key is pressed
                    if len(yellow_bullets) < 5: # limit the number of bullets to
5
                        bullet = pygame.Rect(yellow.x + yellow.width, yellow.y +
yellow.height//2 - BULLET_HEIGHT//2, BULLET_WIDTH, BULLET_HEIGHT)
                        yellow_bullets.append(bullet)
                    elif bullet.x > WIDTH: # remove the bullet if it goes out of
bounds
                        yellow_bullets.remove(bullet)

                # red spaceship firing bullets with right control key
                if event.key == pygame.K_RCTRL and len(red_bullets) < MAx_BULLET:
```

```
                    if len(red_bullets) < 5:
                        bullet = pygame.Rect(red.x, red.y + red.height//2 -
BULLET_HEIGHT//2, BULLET_WIDTH, BULLET_HEIGHT)
                        red_bullets.append(bullet)
                    elif bullet.x < 0: # remove the bullet if it goes out of
bounds
                        red_bullets.remove(bullet)



         # Movement handling
        keys = pygame.key.get_pressed() # get the state of all keys
        handle_yellow_movement(keys, yellow) # call the function to handle yellow
spaceship movement
        handle_red_movement(keys, red) # call the function to handle red
spaceship movement

        handle_bullets(yellow_bullets, red_bullets, yellow, red) # call the
function to handle bullet movement and collision
        draw_window(red , yellow, red_bullets, yellow_bullets,bg_x) # call the
draw_window function to draw the game window


    pygame.quit() # quit pygame when the game loop ends

if __name__ == "__main__":
    main() # call the main function to start the game
```

Step 14: Implementing the Health System

In this step, we introduce a **health system** to track the spaceship's health. Each spaceship starts with **full health (100 points)** and loses **10 points** every time it gets hit by a bullet. When a spaceship's health reaches **zero**, the game declares the **opponent as the winner**.

1. Adding Health Variables

First, we need two variables to track the health of both spaceships in the main () function:

```
    red_health = 100 # initial health of the red spaceship
    yellow_health = 100 # initial health of the yellow spaceship
```

- red_health stores the health of the **red spaceship**.
- yellow_health stores the health of the **yellow spaceship**.

- Both start at **100 points** (this is our chosen max health).
- When a spaceship is hit, its health decreases by **10 points per hit**.

2. Displaying Health on the Screen

The players need to see their spaceship's health, so we **render the health as text** and display it at the top of the screen. This code is added in the draw () function.

```
# draw the health bars for both spaceships
    red_health_text = FONT.render(f"Health: {red_health}", 1, WHITE) # render the
health text for the red spaceship
    yellow_health_text = FONT.render(f"Health: {yellow_health}", 1, WHITE) #
render the health text for the yellow spaceship
    WIN.blit(red_health_text, (WIDTH - red_health_text.get_width() - 10, 10)) #
draw the health text for the red spaceship
    WIN.blit(yellow_health_text, (10, 10)) # draw the health text for the yellow
spaceship
```

Breaking Down FONT.render(f"Health: {red_health}", 1, WHITE)

This function **creates a text surface** that will be displayed on the screen. Let's analyze its parameters:

f"Health: {red_health}"

- This is an **f-string**, meaning the variable red_health will be inserted dynamically.
- If red_health = 90, the text will display:

The number 1

- This **controls anti-aliasing**, which makes text **smoother and easier to read**.
- 1 (or True) **turns anti-aliasing ON**, making the text look smooth.
- 0 (or False) **turns anti-aliasing OFF**, making the text look pixelated.

WHITE

- This defines the **color of the text**.
- WHITE is a variable we defined earlier as (255, 255, 255), meaning **pure white**.

Placing the Health on the Screen

```
WIN.blit(red_health_text, (WIDTH - red_health_text.get_width() - 10, 10)) # draw
the health text for the red spaceship
    WIN.blit(yellow_health_text, (10, 10)) # draw the health text for the yellow
spaceship
```

Red's Health (Top-Right Corner)

- red_health_text.get_width() gets the **width** of the text.
- WIDTH - red_health_text.get_width() - 10 moves the text to the **right** side.
- The 10 offsets it slightly from the right edge.
- 10 in (WIDTH - text_width - 10, 10) sets it **10 pixels from the top**.

Yellow's Health (Top-Left Corner)

- (10, 10) means the text starts **10 pixels from the left** and **10 pixels from the top**.

3. Handling the Health Reduction

Now we listen for the **RED_HIT** and **YELLOW_HIT** events and reduce the spaceship's health. This is in the main () inside the forloop. (for event in pygame.event.get())

```
# Handle hit events
            if event.type == YELLOW_HIT: # check if the yellow spaceship is hit
                yellow_health -= 10 # reduce the health of the red spaceship

            if event.type == RED_HIT: # check if the red spaceship is hit
                red_health -= 10 # reduce the health of the yellow spaceship
```

How This Works

pygame.event.get()

- This **retrieves all game events** (such as key presses, collisions, and our custom events).

Checking for RED_HIT

- If event.type == RED_HIT, it means the **red spaceship got hit**.
- We reduce red_health by **10 points**.

Checking for YELLOW_HIT

- If event.type == YELLOW_HIT, it means the **yellow spaceship got hit**.
- We reduce yellow_health by **10 points**.

4. Checking for Game Over

When a spaceship's health reaches **zero**, the game should **declare the winner**.

```
# Check for game over conditions
        # if either spaceship's health is less than or equal to 0, end the game
and display the winner
```

```
        winner_text = ""
        if yellow_health <= 0:
            winner_text= "Red wins!"

        if red_health <= 0:
            winner_text="Yellow wins!"
```

5. Displaying the Winner and Stopping the Game

Finally, we check if there is a winner and **end the game**.

```
if winner_text != "":
            handle_game_over(winner_text) # call the function to handle game over
            break # exit the game loop
```

Breaking It Down

if winner_text != ""

- If winner_text is **not empty**, that means **we have a winner**.

draw_winner(winner_text)

- This function displays the **winning message** on the screen.

break

- This **exits the game loop**, stopping the game.

6. Handling_game_over () function

```
def handle_game_over(winner_text): # function to handle game over conditions
    draw_text = WINNER_FONT.render(winner_text, 1, WHITE) # render the winner
text
    WIN.blit(draw_text, (WIDTH//2 - draw_text.get_width()//2, HEIGHT/2 -
draw_text.get_height()//2)) # draw the winner text at the center of the window
    pygame.display.update() # update the display
    pygame.time.delay(5000) # delay for 5 seconds before quitting the game
```

def handle_game_over(winner_text):

- This defines a **function** named handle_game_over().
- It takes **one parameter**, winner_text, which stores the message to display (e.g., "Red Wins!" or "Yellow Wins!").
- **`WINNER_FONT.render(winner_text, 1, WHITE)`**
    - This **creates a text surface** that will display the winner's message.

- **Parameters explained:**
  - `winner_text`: The text to display (e.g., `"Red Wins!"`).
  - `1`: Enables **anti-aliasing**, which makes the text smooth.
  - `WHITE`: Sets the text color to **white** `(255, 255, 255)`.
- The result is stored in `draw_text`, which is a **pygame Surface** (image of the text).

Centering the Text

1. **X-position: WIDTH//2 - draw_text.get_width()//2**
   - WIDTH//2 finds the **center of the screen**.
   - draw_text.get_width()//2 **halves the text width** to center it properly.
   - Subtracting ensures the text starts **at the correct centered position**.
2. **Y-position: HEIGHT/2 - draw_text.get_height()//2**
   - HEIGHT/2 finds the **vertical center** of the screen.
   - draw_text.get_height()//2 adjusts the position so the text is fully centered.

Example Calculation (if screen is 800x600 pixels):

- WIDTH//2 = 800//2 = 400
- draw_text.get_width()//2 = 100//2 = 50 (if text is 100 pixels wide)
- **Final X position:** 400 - 50 = 350 (text starts at pixel 350)

Note: I also moved pygame.quit() and main() into handle_game_over() to ensure a cleaner and more structured game loop. By placing them inside the function, the game now properly quits or restarts only when a game-over condition is met.

```
   while run:
        clock.tick(FPS) # set the frame rate

        # Handle background movement
        bg_x -= BG_VEL
        if bg_x <= - 900:  # Reset the background position when it moves out of
the window
            bg_x = 0

        # Event handling
        for event in pygame.event.get(): # loop through all events
            # check for quit event
            if event.type == pygame.QUIT:
                run = False # exit the game loop
                pygame.quit() # quit pygame when the game loop ends

            # Handle bullet firing
            if event.type == pygame.KEYDOWN:
                # yellow spaceship firing bullets with left control key
```

```
                if event.key == pygame.K_LCTRL and len(yellow_bullets) <
MAx_BULLET: # check if the left control key is pressed
                    if len(yellow_bullets) < 5: # limit the number of bullets to
5
                        bullet = pygame.Rect(yellow.x + yellow.width, yellow.y +
yellow.height//2 - BULLET_HEIGHT//2, BULLET_WIDTH, BULLET_HEIGHT)
                        yellow_bullets.append(bullet)
                    elif bullet.x > WIDTH: # remove the bullet if it goes out of
bounds
                        yellow_bullets.remove(bullet)

                # red spaceship firing bullets with right control key
                if event.key == pygame.K_RCTRL and len(red_bullets) < MAx_BULLET:
                    if len(red_bullets) < 5:
                        bullet = pygame.Rect(red.x, red.y + red.height//2 -
BULLET_HEIGHT//2, BULLET_WIDTH, BULLET_HEIGHT)
                        red_bullets.append(bullet)
                    elif bullet.x < 0: # remove the bullet if it goes out of
bounds
                        red_bullets.remove(bullet)

            # Handle hit events
            if event.type == YELLOW_HIT: # check if the yellow spaceship is hit
                yellow_health -= 10 # reduce the health of the red spaceship

            if event.type == RED_HIT: # check if the red spaceship is hit
                red_health -= 10 # reduce the health of the yellow spaceship

        # Check for game over conditions
        # if either spaceship's health is less than or equal to 0, end the game
and display the winner
        winner_text = ""
        if yellow_health <= 0:
            winner_text= "Red wins!"

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

        if winner_text != "":
           handle_game_over(winner_text) # call the function to handle game over
           break # exit the game loop

         # Movement handling
        keys = pygame.key.get_pressed() # get the state of all keys
```

```
        handle_yellow_movement(keys, yellow) # call the function to handle yellow
spaceship movement
        handle_red_movement(keys, red) # call the function to handle red
spaceship movement

        handle_bullets(yellow_bullets, red_bullets, yellow, red) # call the
function to handle bullet movement and collision
        draw_window(red , yellow, red_bullets,
yellow_bullets,bg_x,red_health,yellow_health) # call the draw_window function to
draw the game window


    main() # restart the game if the game loop ends

if __name__ == "__main__":
    main() # call the main function to start the game
```

Step 15: Adding Sound Effects

a) Start by importing this module

```
pygame.mixer.init() # initialize the mixer module for sound playback
```

b) load in a mixer sounds.

```
# load in mixer for sound effects
BACKGROUND_SOUND =
pygame.mixer.music.load(os.path.join('Assets','background_music.mp3')) # load
background music
BULLET_HIT_SOUND =
pygame.mixer.Sound(os.path.join('Assets','Bullet_Hit.mp3')) # load bullet hit
sound
BULLET_FIRE_SOUND =
pygame.mixer.Sound(os.path.join('Assets','Bullet_Fire.mp3')) # load bullet
fire sound
```

c) Play the sound in the main () function

```
def main():
    # background music setup
    pygame.mixer.music.play(-1)  # Play background music in an infinite loop
```

```
    pygame.mixer.music.set_volume(0.5)  # Set the volume of the background
music

    bg_x = 0  # Initial Y position of the background
    BG_VEL = 0.5 # Speed at which the background moves down

    red = pygame.Rect(700, 200, SPACESHIP_WIDTH, SPACESHIP_HEIGHT) # create a
rectangle for the red spaceship
    yellow = pygame.Rect(100, 200, SPACESHIP_WIDTH, SPACESHIP_HEIGHT) #
create a rectangle for the yellow spaceship
    yellow_bullets = [] # list to store bullets
    red_bullets = [] # list to store bullets
    clock = pygame.time.Clock() # create a clock object to control the frame
rate

    red_health = 100 # initial health of the red spaceship
    yellow_health = 100 # initial health of the yellow spaceship

    # Main function to run the game
    run = True # variable to control the game loop
    while run:
        clock.tick(FPS) # set the frame rate

        # Handle background movement
        bg_x -= BG_VEL
        if bg_x <= - 900:  # Reset the background position when it moves out
of the window
            bg_x = 0

        # Event handling
        for event in pygame.event.get(): # loop through all events
            # check for quit event
            if event.type == pygame.QUIT:
                run = False # exit the game loop
                pygame.quit() # quit pygame when the game loop ends

            # Handle bullet firing
            if event.type == pygame.KEYDOWN:
                # yellow spaceship firing bullets with left control key
                if event.key == pygame.K_LCTRL and len(yellow_bullets) <
MAx_BULLET: # check if the left control key is pressed
                    if len(yellow_bullets) < 5: # limit the number of bullets
to 5
                        bullet = pygame.Rect(yellow.x + yellow.width,
yellow.y + yellow.height//2 - BULLET_HEIGHT//2, BULLET_WIDTH, BULLET_HEIGHT)
```

```
                        yellow_bullets.append(bullet)
                        BULLET_FIRE_SOUND.play() # play the bullet fire sound

                    elif bullet.x > WIDTH: # remove the bullet if it goes out
of bounds
                        yellow_bullets.remove(bullet)


                # red spaceship firing bullets with right control key
                if event.key == pygame.K_RCTRL and len(red_bullets) <
MAx_BULLET:
                    if len(red_bullets) < 5:
                        bullet = pygame.Rect(red.x, red.y + red.height//2 -
BULLET_HEIGHT//2, BULLET_WIDTH, BULLET_HEIGHT)
                        red_bullets.append(bullet)
                        BULLET_FIRE_SOUND.play() # play the bullet fire sound

                    elif bullet.x < 0: # remove the bullet if it goes out of
bounds
                        red_bullets.remove(bullet)

            # Handle hit events
            if event.type == YELLOW_HIT: # check if the yellow spaceship is
hit
                yellow_health -= 10 # reduce the health of the red spaceship
                BULLET_HIT_SOUND.play() # play the bullet hit sound

            if event.type == RED_HIT: # check if the red spaceship is hit
                red_health -= 10 # reduce the health of the yellow spaceship
                BULLET_HIT_SOUND.play() # play the bullet hit sound

        # Check for game over conditions
        # if either spaceship's health is less than or equal to 0, end the
game and display the winner
        winner_text = ""
        if yellow_health <= 0:
            winner_text= "Red wins!"

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

        if winner_text != "":
           handle_game_over(winner_text) # call the function to handle game
over
           break # exit the game loop
```

```
        # Movement handling
        keys = pygame.key.get_pressed() # get the state of all keys
        handle_yellow_movement(keys, yellow) # call the function to handle
yellow spaceship movement
        handle_red_movement(keys, red) # call the function to handle red
spaceship movement

        handle_bullets(yellow_bullets, red_bullets, yellow, red) # call the
function to handle bullet movement and collision
        draw_window(red , yellow, red_bullets,
yellow_bullets,bg_x,red_health,yellow_health) # call the draw_window function
to draw the game window

    pygame.mixer.music.stop() # stop the background music when the game loop
ends

    main() # restart the game if the game loop ends
```

STEP 16: Adding a Restart Button

Let's implement a Restart button that the player can click it to restart the game.

a. Create a function called `draw_restart_button()` & Define Button Size and Position

```
# Restart button variables
button_width, button_height = 220, 70
button_x = WIDTH//2 - button_width//2 - 5  # x position to center the button
button_y = HEIGHT//2 + 100 - 50  # y position to center the button
```

b. Create and Draw the Button

```
# Draw restart button
restart_button = pygame.Rect(button_x, button_y, button_width, button_height)
pygame.draw.rect(WIN, RED, restart_button)
```

c. Add "Restart" Text to the Button

```
# Restart button text variables
button_text = BUTTON_FONT.render("Restart", True, BLACK)
text_x = restart_button.x + (restart_button.width - button_text.get_width())
// 2
text_y = restart_button.y + (restart_button.height -
button_text.get_height()) // 2

# Draw button text
```

```
WIN.blit(button_text, (text_x, text_y))
pygame.display.update()
```

d. Wait for the Player to Click the Button

```
# Wait for user to click restart button
waiting = True
while waiting:
      for event in pygame.event.get():
          if event.type == pygame.QUIT:
              pygame.quit()
              exit()
          if event.type == pygame.MOUSEBUTTONDOWN:
               if restart_button.collidepoint(event.pos):
                  waiting = False
```

e. Call the function in handle_game_over ()

```
def handle_game_over(winner_text):
    pygame.mixer.music.stop()
    ViCTORY_SOUND.play()

    draw_text = WINNER_FONT.render(winner_text, 1, WHITE)
    WIN.blit(draw_text, (WIDTH//2 - draw_text.get_width()//2, HEIGHT//2 -
draw_text.get_height()//2))
    draw_restart_button()
```

HANDLING BUGS

I realized that bullets were not being properly removed when they went off-screen. This caused the spaceships to stop firing ammo after a certain point. To fix this, I adjusted the code to ensure that bullets are removed from the list as soon as they leave the screen. (The fix is highlighted in blue in the code below).

```
def handle_bullets(yellow_bullets, red_bullets, yellow, red): # function to
handle bullet movement and collision
    for bullet in yellow_bullets: # loop through all yellow bullets
        bullet.x += BULLET_VEL # move the bullet to the right
        if bullet.x > WIDTH:  # Remove bullet when it goes off screen
            yellow_bullets.remove(bullet)
        elif red.colliderect(bullet):
            pygame.event.post(pygame.event.Event(RED_HIT)) # post a custom event
if the red spaceship is hit
            yellow_bullets.remove(bullet) # remove the bullet if it collides with
the red spaceship
```

```
    for bullet in red_bullets:
        bullet.x -= BULLET_VEL # move the bullet to the left
        if bullet.x < 0:  # Remove bullet when it goes off screen
            red_bullets.remove(bullet)
        elif yellow.colliderect(bullet):
            pygame.event.post(pygame.event.Event(YELLOW_HIT)) # post a custom
event if the yellow spaceship is hit
            red_bullets.remove(bullet) # remove the bullet if it collides with
the yellow spaceship
```

I also added a victory sound effect when a player wins and a hover over effect on the restart button.

Here is the full complete code.

```
import pygame
import os

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

# Game window settings
WIDTH, HEIGHT = 920, 500
WIN = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Cosmic Duel")

# Fonts
FONT = pygame.font.SysFont('comicsans', 20)
WINNER_FONT = pygame.font.SysFont('comicsans', 60)
BUTTON_FONT = pygame.font.SysFont('comicsans', 40)

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

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

# Frame Rate
FPS = 60

# Sounds
```

```
BACKGROUND_SOUND = pygame.mixer.music.load(os.path.join('Assets',
'background_music.mp3'))
BULLET_HIT_SOUND = pygame.mixer.Sound(os.path.join('Assets', 'Bullet_Hit.mp3'))
BULLET_FIRE_SOUND = pygame.mixer.Sound(os.path.join('Assets', 'Bullet_Fire.mp3'))
ViCTORY_SOUND = pygame.mixer.Sound(os.path.join('Assets',
'Victory_Sound_Effect.mp3'))

# Images
YELLOW_SPACESHIP_IMAGE = pygame.image.load(os.path.join('Assets',
'spaceship_yellow.png'))
RED_SPACESHIP_IMAGE = pygame.image.load(os.path.join('Assets',
'spaceship_red.png'))
SPACE_BACKGROUND = pygame.image.load(os.path.join('Assets',
'space_background.jpg'))

SPACESHIP_WIDTH, SPACESHIP_HEIGHT = 55, 40

YELLOW_SPACESHIP = pygame.transform.rotate(
    pygame.transform.scale(YELLOW_SPACESHIP_IMAGE, (SPACESHIP_WIDTH,
SPACESHIP_HEIGHT)), 270)
RED_SPACESHIP = pygame.transform.rotate(
    pygame.transform.scale(RED_SPACESHIP_IMAGE, (SPACESHIP_WIDTH,
SPACESHIP_HEIGHT)), 90)
SPACE_BACKGROUND = pygame.transform.scale(SPACE_BACKGROUND, (5000, 3000))

# Velocities
VEL = 5
BULLET_VEL = 7
MAx_BULLET = 5
BULLET_WIDTH, BULLET_HEIGHT = 10, 5

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

def draw_window(red, yellow, red_bullets, yellow_bullets, bg_x, red_health,
yellow_health):
    WIN.blit(SPACE_BACKGROUND, (bg_x, 0))
    WIN.blit(SPACE_BACKGROUND, (bg_x + WIDTH, 0))

    red_health_text = FONT.render(f"Health: {red_health}", 1, WHITE)
    yellow_health_text = FONT.render(f"Health: {yellow_health}", 1, WHITE)
    WIN.blit(red_health_text, (WIDTH - red_health_text.get_width() - 10, 10))
    WIN.blit(yellow_health_text, (10, 10))
```

```
    pygame.draw.rect(WIN, BLACK, BORDER)
    WIN.blit(YELLOW_SPACESHIP, (yellow.x, yellow.y))
    WIN.blit(RED_SPACESHIP, (red.x, red.y))

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

    pygame.display.update()

def draw_restart_button():
    # Restart button variables
    button_width, button_height = 220, 70
    button_x = WIDTH//2 - button_width//2 - 5  # Center the button
    button_y = HEIGHT//2 + 100 - 50  # Position the button

    restart_button = pygame.Rect(button_x, button_y, button_width, button_height)

    waiting = True
    while waiting:
        WIN.fill((0, 0, 0), restart_button)  # Clear previous button

        mouse_x, mouse_y = pygame.mouse.get_pos()
        if restart_button.collidepoint((mouse_x, mouse_y)):
            button_color = (255, 100, 100)  # Lighter red when hovered
        else:
            button_color = RED  # Default color

        pygame.draw.rect(WIN, button_color, restart_button)

        # Restart button text
        button_text = BUTTON_FONT.render("Restart", True, BLACK)
        text_x = restart_button.x + (restart_button.width -
button_text.get_width()) // 2
        text_y = restart_button.y + (restart_button.height -
button_text.get_height()) // 2
        WIN.blit(button_text, (text_x, text_y))

        pygame.display.update()

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

```
            if event.type == pygame.MOUSEBUTTONDOWN:
                if restart_button.collidepoint(event.pos):
                    waiting = False



def handle_game_over(winner_text):
    pygame.mixer.music.stop()
    ViCTORY_SOUND.play()

    draw_text = WINNER_FONT.render(winner_text, 1, WHITE)
    WIN.blit(draw_text, (WIDTH//2 - draw_text.get_width()//2, HEIGHT//2 -
draw_text.get_height()//2))
    draw_restart_button()

def main():
    pygame.mixer.music.play(-1)
    pygame.mixer.music.set_volume(0.5)

    bg_x = 0
    BG_VEL = 0.5

    red = pygame.Rect(700, 200, SPACESHIP_WIDTH, SPACESHIP_HEIGHT)
    yellow = pygame.Rect(100, 200, SPACESHIP_WIDTH, SPACESHIP_HEIGHT)
    yellow_bullets = []
    red_bullets = []
    clock = pygame.time.Clock()

    red_health = 100
    yellow_health = 100

    run = True
    while run:
        clock.tick(FPS)

        bg_x -= BG_VEL
        if bg_x <= -900:
            bg_x = 0

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

            if event.type == pygame.KEYDOWN:
```

```
                if event.key == pygame.K_LCTRL and len(yellow_bullets) <
MAx_BULLET:
                    bullet = pygame.Rect(yellow.x + yellow.width, yellow.y +
yellow.height//2 - BULLET_HEIGHT//2, BULLET_WIDTH, BULLET_HEIGHT)
                    yellow_bullets.append(bullet)
                    BULLET_FIRE_SOUND.play()

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

            if event.type == YELLOW_HIT:
                yellow_health -= 10
                BULLET_HIT_SOUND.play()

            if event.type == RED_HIT:
                red_health -= 10
                BULLET_HIT_SOUND.play()

        winner_text = ""
        if yellow_health <= 0:
            winner_text = "Red wins!"
        if red_health <= 0:
            winner_text = "Yellow wins!"

        if winner_text:
            handle_game_over(winner_text)
            return main()

        keys = pygame.key.get_pressed()
        if keys[pygame.K_a] and yellow.x - VEL > 0:
            yellow.x -= VEL
        if keys[pygame.K_d] and yellow.x + VEL + yellow.width < BORDER.x:
            yellow.x += VEL
        if keys[pygame.K_w] and yellow.y - VEL > 0:
            yellow.y -= VEL
        if keys[pygame.K_s] and yellow.y + VEL + yellow.height < HEIGHT - 10:
            yellow.y += VEL

        if keys[pygame.K_LEFT] and red.x - VEL > BORDER.x + BORDER.width:
            red.x -= VEL
        if keys[pygame.K_RIGHT] and red.x + VEL + red.width < WIDTH:
            red.x += VEL
```

```
        if keys[pygame.K_UP] and red.y - VEL > 0:
            red.y -= VEL
        if keys[pygame.K_DOWN] and red.y + VEL + red.height < HEIGHT - 10:
            red.y += VEL

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

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

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

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

Convert the .py to exe

1. **Navigate to Your Game's Directory from your terminal**

Use the command prompt or terminal to navigate to the folder containing your **main.py** (or the main script of your game).

For example: *cd C:\Users\YourName\Games\CosmicDuel*

2. **Activate Your Virtual Environment**

If your virtual environment is in `my_env`, activate it:

For example *C:\users\YourName\Virtual\my_env\Scripts\activate*

Make sure you use the correct command to activate your virtual environment:

For **Windows (Command Prompt - cmd)**: my_env\Scripts\activate

For **Windows (PowerShell)**: my_env\Scripts\Activate.ps1

For **macOS/Linux**: source my_env/bin/activate

3. **Install Dependencies (If Not Installed)**

```
pip install pygame
pip install Pillow
```

4. **Run PyInstaller**

Now, run the following command to generate the .exe file:

```
pyinstaller --onefile --windowed --hidden-import pygame --icon=icon.ico main.py
```

If your game depends on external files (like images, sounds, or fonts), PyInstaller might not include them automatically. Instead use the command below:

```
pyinstaller --onefile --windowed --hidden-import pygame --add-data
"Assets;Assets" --icon=icon.ico main.py
```

Explanation of Flags:

- **--onefile**: Bundles everything into a single .exe file.
- **--windowed**: Prevents a console window from appearing (useful for GUI-based apps like Pygame).
- **--hidden-import pygame** → Ensures PyInstaller includes pygame.
- **--add-data "Assets;Assets"** → Ensures the Assets folder is bundled correctly.
- **--icon=icon.ico**: (Optional) Adds a custom icon for your game. Replace icon.ico with the actual icon file. My file is known as icon.ico

(On Windows, use ; as a separator, but on Mac/Linux, use : instead.)

If you don't have an icon and you have a png or jpg file you can use the convert_image_to_icon.py script I have provided to convert the file to ico (just make sure to replace icon.png with your file name.

5. **Locate the Executable**
   After running the command, **PyInstaller** creates several folders. Your executable will be inside the **dist** folder:

```
CosmicDuel
├── build/
├── dist/
│   ├── main.exe  <-- Your final game executable!
├── main.py
├── assets/
├── game_icon.ico
├── main.spec
```

The file **dist/main.exe** is your game's **executable**.

6. **Test the Executable**

Run the .exe file from the dist folder to make sure everything works correctly.

EXE Troubleshooting and Error Handling

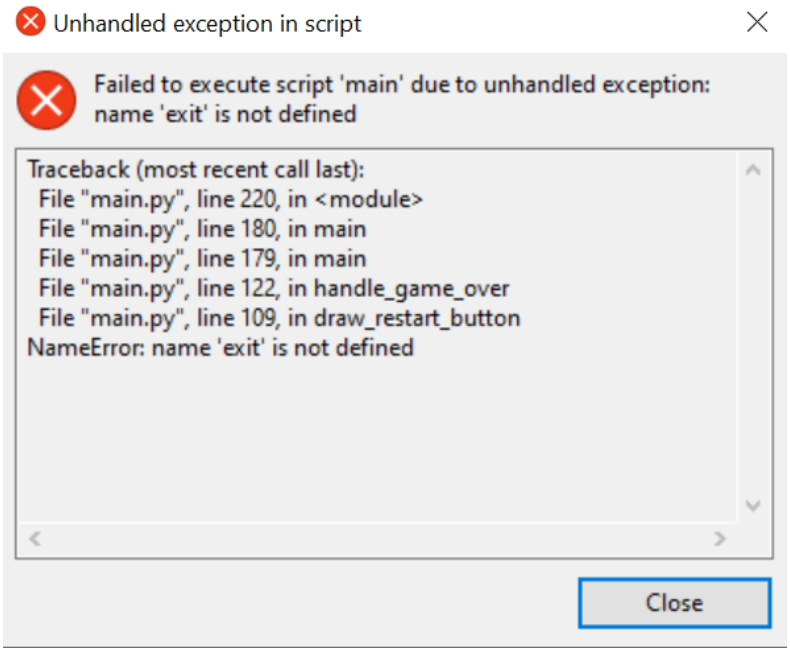
Handling Missing Assets

If you did step 4 above and you are still missing the assets folder in your dist directory.

Easy Fix: Manually copy your **Assets** folder into dist/ (next to main.exe).

Handling EXIT Problems

After doing the steps above I opened the main.exe file and everything was working great but when I tried to quit the game I got the error message below.



After a bit of research, I found out that; The error is happening because exit is **not a built-in function in Python** when running as an executable.

Why does this happen?

- In **VSCode (or normal Python)**, exit() is available because the interactive shell (REPL) defines it.
- In **a PyInstaller .exe**, exit is **not automatically available**, causing the NameError.

Fix: Use sys.exit() Instead

1. At the top of your script (main.py), import sys:

```
import sys
```

2. Define a function called exit ():

```
# Exit function to ensure proper exit
def exit():
    sys.exit()
```

3. Call the function in the main () while loop: (as highlighted below)

```
while run:
        clock.tick(FPS)

        bg_x -= BG_VEL
        if bg_x <= -900:
            bg_x = 0

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

4. Rebuild the .exe:

```
pyinstaller --onefile --windowed --hidden-import pygame --add-data
"Assets;Assets" --icon=icon.ico main.py
```

Distribute Your Game (Optional)

- **ZIP Method**: Compress the dist folder and share it with players.
- **Installer (Optional)**: Use software like **Inno Setup** to create a proper installer.
- **Itch.io/Steam (Optional)**: If you want to publish your game, platforms like Itch.io and Steam allow you to distribute indie games.