Pygame - The Basics

Think of your favourite video game. Behind the scenes, there are thousands of lines of code working together to make the characters move, the graphics appear, and the sounds play. Now, if you wanted to create a game yourself, starting from zero might be super overwhelming, right?

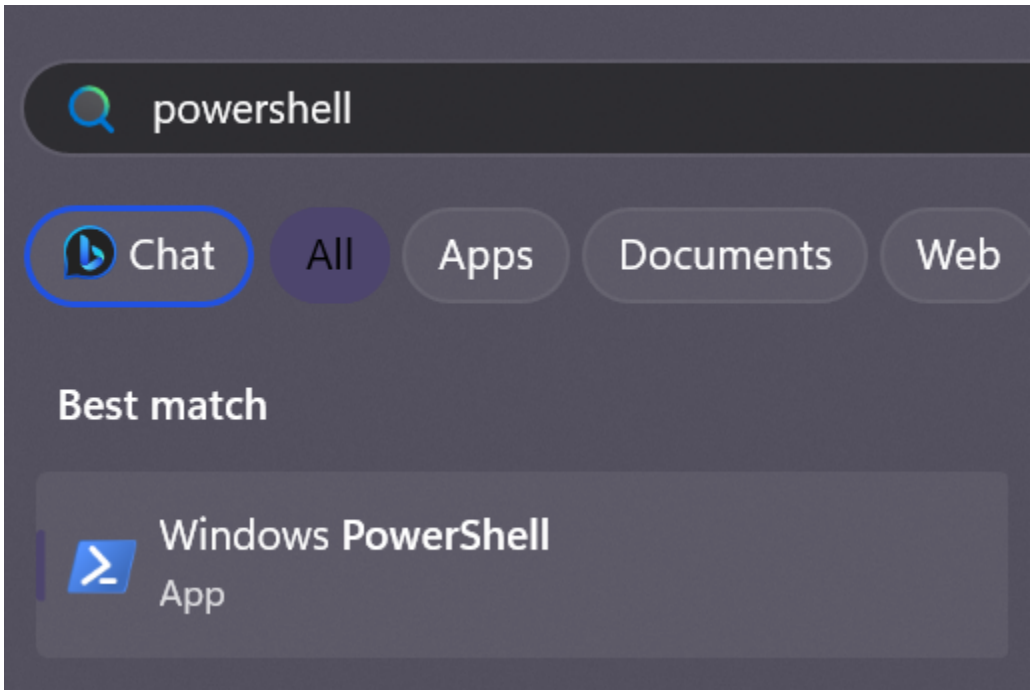
That's where Pygame comes in. It's a library for Python, which is a popular programming language. A library is like a toolkit filled with pre-made tools and shortcuts. Instead of you having to code every tiny detail, like how a character moves on the screen, Pygame provides you with tools to do that more easily.

I will be creating my pygame games locally, but you can also easily follow along using the pygame template on Replit.

Installing pygame

If you are using Replit, you can skip this section.

1. On Windows, search for PowerShell in your start menu and launch it



2. Copy and paste the following command into your console and press the enter key

```
py -m pip install -U pygame --user
```

Troubleshooting

If the installation command results in an error, make sure that python is installed correctly using the command:

```
py --version
```

This should show you the python version installed on your computer. It will look something like this:

```
PS C:\Users\Samuel> py --version
Python 3.11.5
```

If it does not, you do not have python installed correctly, or it is not accessible.

Next, make sure that pip is installed on your computer. Pip is a package manager for Python that allows you to install and manage additional libraries. You can use the command:

```
pip --version
```

This should show you the version of the pip that is installed on your pc.

```
PS C:\Users\Samuel> pip --version
pip 23.3.1 from C:\Users\Samuel\AppData\Roaming\Python
\Python311\site-packages\pip (python 3.11)
```

If it does not show you an output like that, you do not have pip installed on your pc.

Pygame - Key elements

The Game Window

Think of the Pygame game window as your personal digital playground or canvas. It's where all the action of your game happens. Here's a breakdown tailored for a teenager:

1. **Canvas**: The Pygame window is like the main screen or stage of your game. Everything you want the player to see, from characters to backgrounds to scores, will be displayed here.

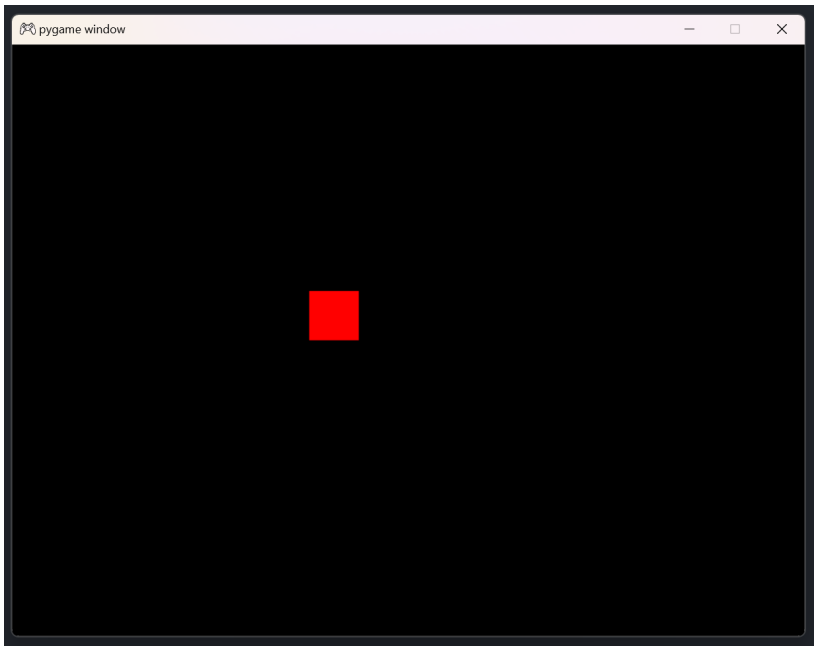
2. **Resolution**: This is the size of your window. Just like when you choose the screen resolution on a TV or monitor, in Pygame, you can set how big or small you want your game window to be.

3. **Title**: At the top of the window, there's a bar where you can put the name of your game. It's like the title of a book, letting players know what game they're playing. Our games icon will also be displayed here

4. **FPS (Frames Per Second)**: Imagine a flip book where each page is a picture. When you flip through it quickly, the pictures seem to move. Games work similarly. Each frame is a picture, and FPS determines how many of these pictures or 'frames' are shown every second. Higher FPS makes your game look smoother.

5. **Closing the Window**: Just like any other software window, there's an 'X' or close button. You can use Pygame to detect when someone clicks this and ensure your game shuts down smoothly.

In our first game, our window will be fairly basic and boring, but we will learn the basics for our upcoming projects!



The Game Loop

The game loop is the meat and potatoes of any pygame game. It is where we can write gameplay code, check for events, draw sprites and shapes and more.

Here's a breakdown of the Pygame game loop:

1. **Initialization**: Before the loop starts, the game sets up the basics. Here we will prep and configure all the components of our game that need to be in place before we can start playing

2. **Check for Events**: Once inside the loop, the game is always on the lookout for what the player is doing, like pressing keys, clicking the mouse, or

closing the game window. We can choose to react to certain events and handle them however we like

3. **Update the Game**: Depending on the player's actions or other factors, the game updates. This could mean moving a character, updating the score, or changing the game's state

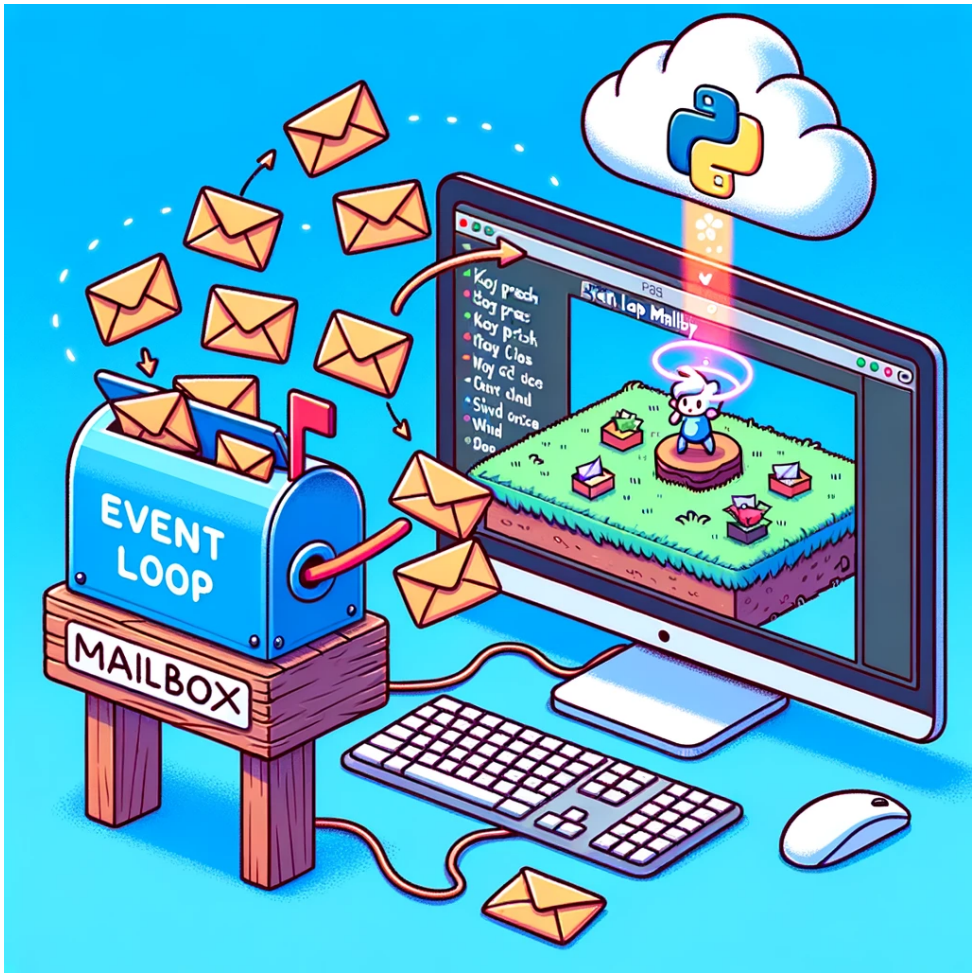
4. **Draw on the Screen**: After updating, the game displays the latest situation on the screen. Everything gets drawn again to show the newest changes. It's like how each scene in an episode shows you the current story

5. **Repeat**: Once we finish dealing with one frame, we are ready to deal with the next and so on. Steps 2-4 keep repeating rapidly, again and again, making the game run smoothly

6. **Exit**: Eventually, when you decide to close the game (or if the game ends), the loop stops, and the game shuts down

In short, the Pygame game loop is the heart of the game. It keeps everything moving, responding, and updating, ensuring that the game experience is dynamic and engaging. Just like how episodes flow in a series, the game loop ensures the game flows seamlessly for the player.

The Event Loop



Imagine you're playing a video game where you control a character. Every time you press a button on your controller—like the jump button or the move left stick—something happens in the game, right? That's because the game is constantly watching for your actions, or "events", and then decides what to do when it sees you've done something.

In Pygame, which is a set of tools to help you make games in Python, events work the same way. The game keeps an eye out for things like key presses, mouse clicks, or even the game window closing. Here's how it happens:

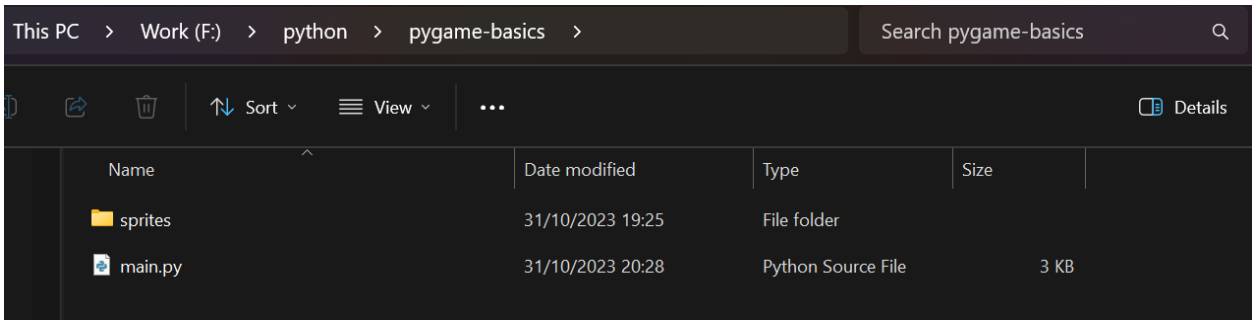
1. **Listening for Events:** Pygame has something like a mailbox for events. Every frame of the game (many times per second), Pygame checks this mailbox to see what's new. Maybe it's a "space bar pressed" or a "mouse moved" event

2. **Handling Events:** Once Pygame sees an event, it needs to decide what to do with it. This is where you come in as the programmer. You write the code that says, "Hey, when the spacebar is pressed, make the character jump."

3. **Writing Event Loops:** To make all this happen, you write what's called an "event loop" in your game code. It's a loop because it runs over and over again, checking for new events each time. Inside this loop, you'll have a bunch of if statements or a match case to handle different events

Let's get started

1. Create a new python project. Choose a folder where you want your project to live and create a file called *main.py* in it
2. Then create a folder in the same directory called sprites. This is where we will keep our game sprites



3. Open the main.py file using your code editor of choice
4. Let set up the basic structure of our game

```
def main():
    print("Hello World")


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

5. Next, import the pygame library at the top of the file

```
import pygame
```

6. Let now write the key elements of any pygame project into the main function

```
import sys
import pygame

def main():
    # Initialization
    pygame.init()
    screen = pygame.display.set_mode((600, 400))

    running = True

    # Game loop
    while running:
        # Event loop
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False

    # Cleanup
    pygame.quit()
    sys.exit(0)
```

> **Challenge!**
>
> Listen to the *MOUSEBUTTONDOWN* and print "Mouse clicked!" when it occurs. Then, remove some of the hard-coded values we have used to global variables at the top of the file underneath the import statements

7. We now want to set the background colour of our game. Add the following code to the bottom of our game loop. First we set the screen's fill colour to black, and then we update the display

```
  screen.fill((0, 0, 0))
  pygame.display.update()
```

> **Challenge!**
>
> Can you turn the background colour White, Green, Red, Blue and Yellow?

8. Let's create a basic player. In this game, it will be a simple square. Declare it in the main function just above the game loop

```
player = pygame.Rect((300, 250, 50, 50))
```

9. Then let draw the player inside the game loop. When you launch the game now, you should see a red rectangle on the screen

```
pygame.draw.rect(screen, (255, 0, 0), player)
```

> **Challenge!**
>
> Find out what the magic numbers in the above lines of code do. Change them and restart the game to find out

10. Next, we need to know how to read player input. Pygame provides a way of easily checking which keys are pressed down every frame. The *get_pressed* function returns a list of all the pressed keys in this frame

```
pressed_keys = pygame.key.get_pressed()
```

11. You can check if a specific key has been pressed using the pressed_keys variable

```
if pressed_keys[pygame.K_UP]:
            print("The up arrow key was pressed!")
```

> **Challenge!**
>
> Check for input of all the arrow keys and move the player using the `player.move_ip(<x_movement>, <y_movement>)`

12. Lastly, we want to be able to control the FPS (Frames per second) our game runs at. First, we need to initialize a clock object at the top of the main function

```
fpsClock = pygame.time.Clock()
```

13. Then, as the very last line in our game loop, we can tick the clock with a target FPS value

```
fpsClock.tick(60)
```

Challenge!

Change the target FPS value using a global variable. What do you notice happens to our player's speed when you increase and decrease the FPS value?

Next Steps

In the next tutorial, we will focus on Sprites. We will add a player sprite, an exciting background and some more gameplay!