



GAME#1

"SHOOT THE FRUIT"



WHAT DO YOU THINK PYTHON LIBRARIES CAN DO?



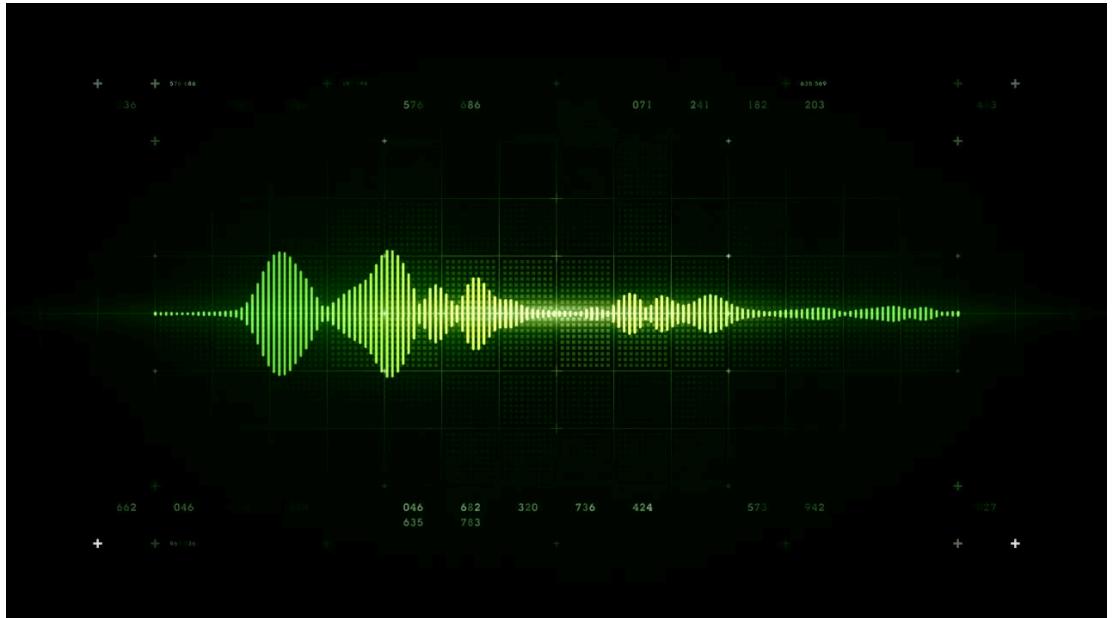
WHAT ARE PYTHON LIBRARIES?



WHAT DO YOU THINK PYTHON LIBRARIES CAN DO?



WHAT ARE PYTHON LIBRARIES?



AI Music and voice generator



Autonomous driving



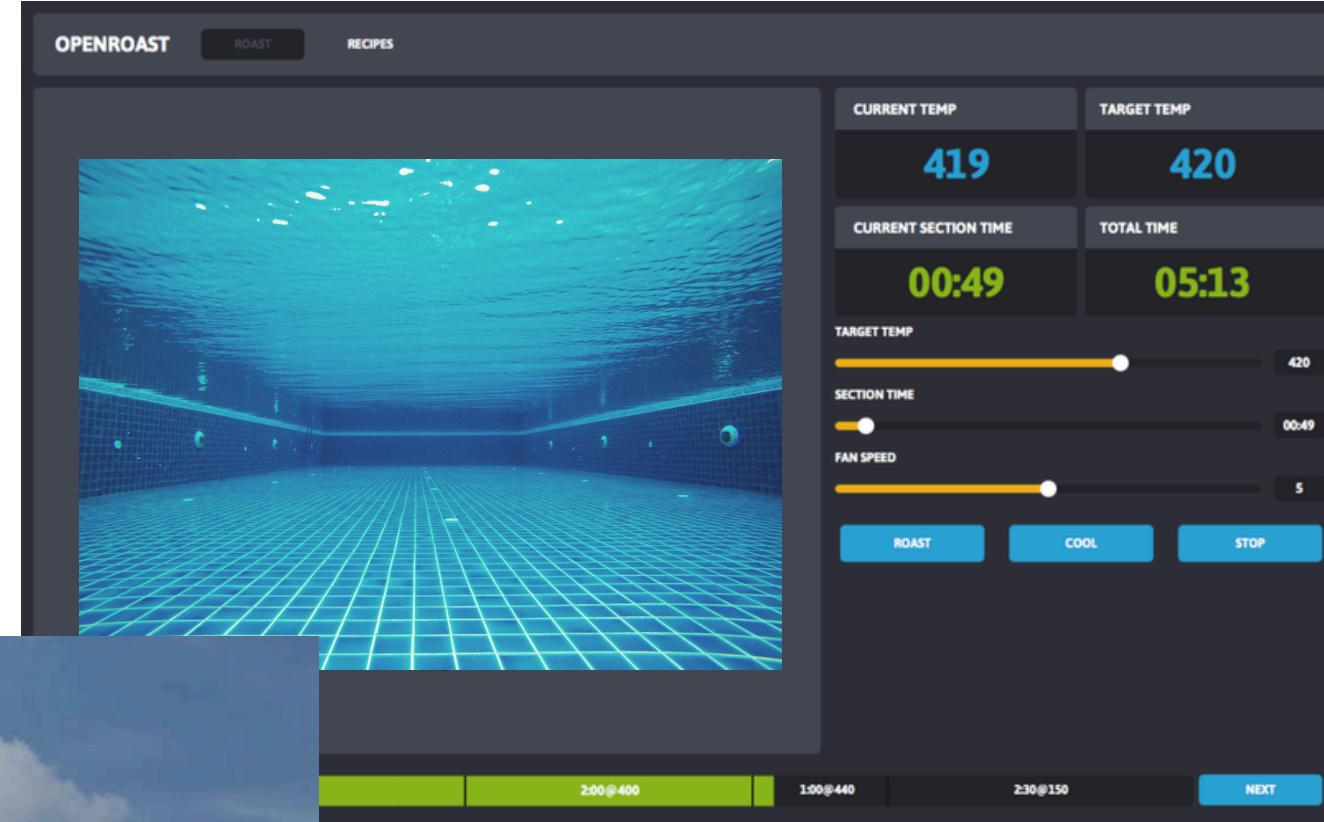
ChatGPT & LLMs



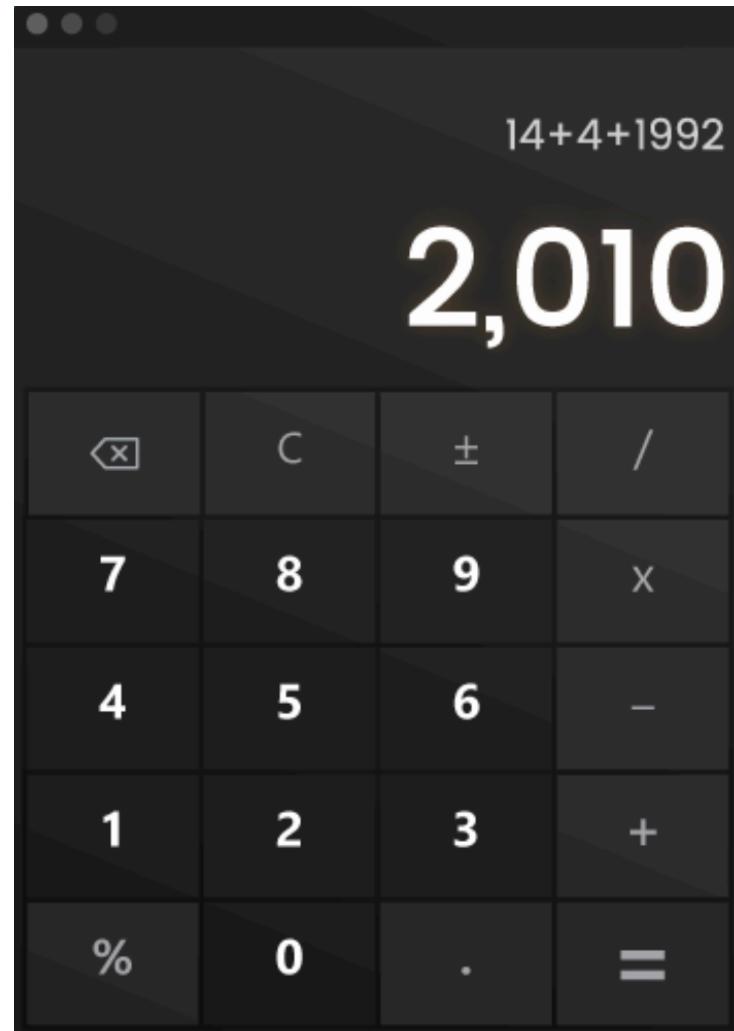
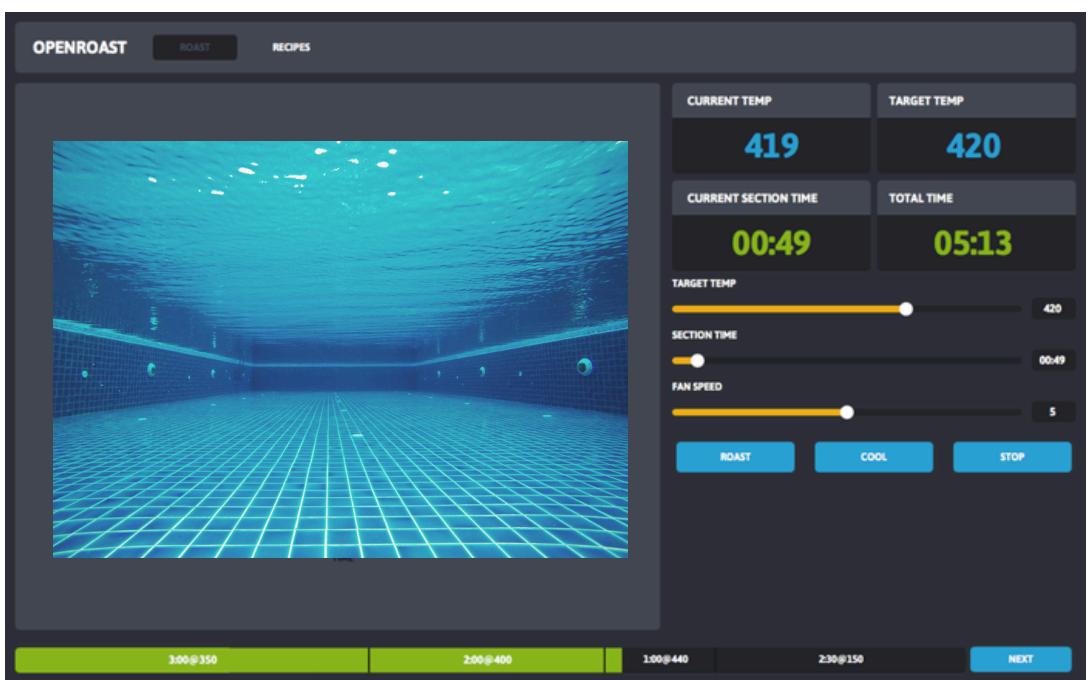
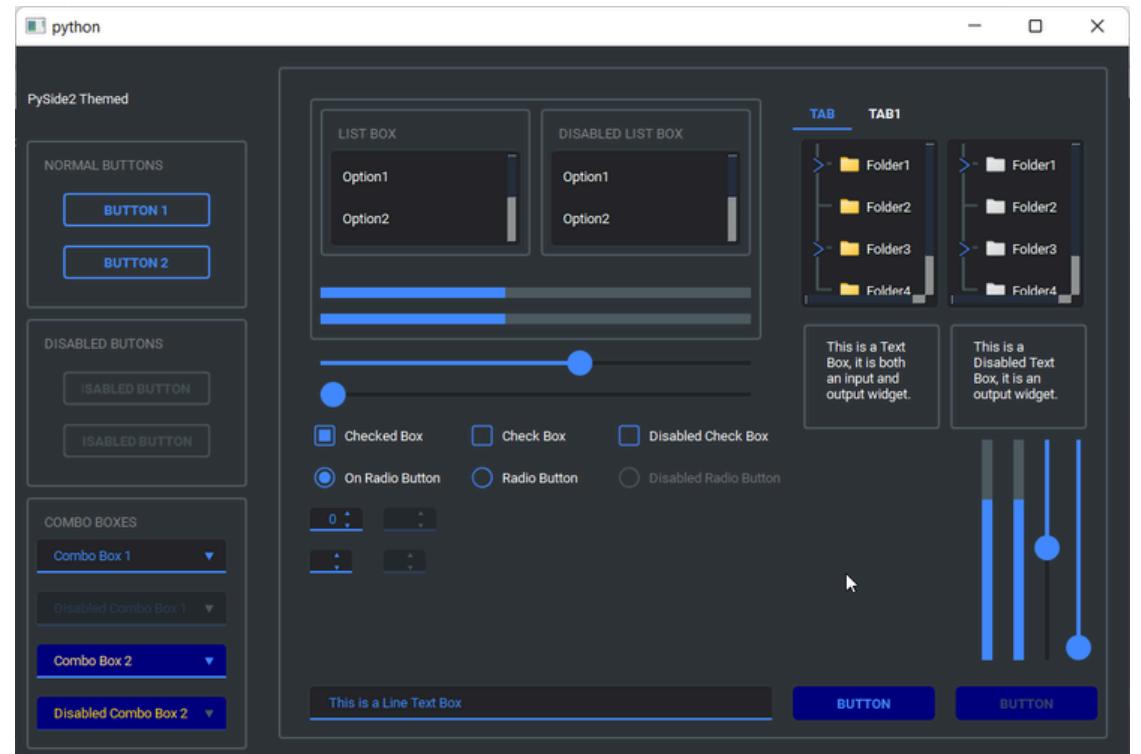
Photo and video generator



WHAT DO YOU THINK PYTHON LIBRARIES CAN DO?



WHAT ARE PYTHON LIBRARIES?



WHAT DO YOU THINK PYTHON LIBRARIES CAN DO?



WHAT ARE PYTHON LIBRARIES?



TOPICS

01

Introduction to Game Development

- How do games work?
- What is pygame and pygame zero?
- Installing pygame zero.

02

Drawing to the Screen

- How do we draw our game to the screen?
- How do we write text in our game?

03

Using the Mouse

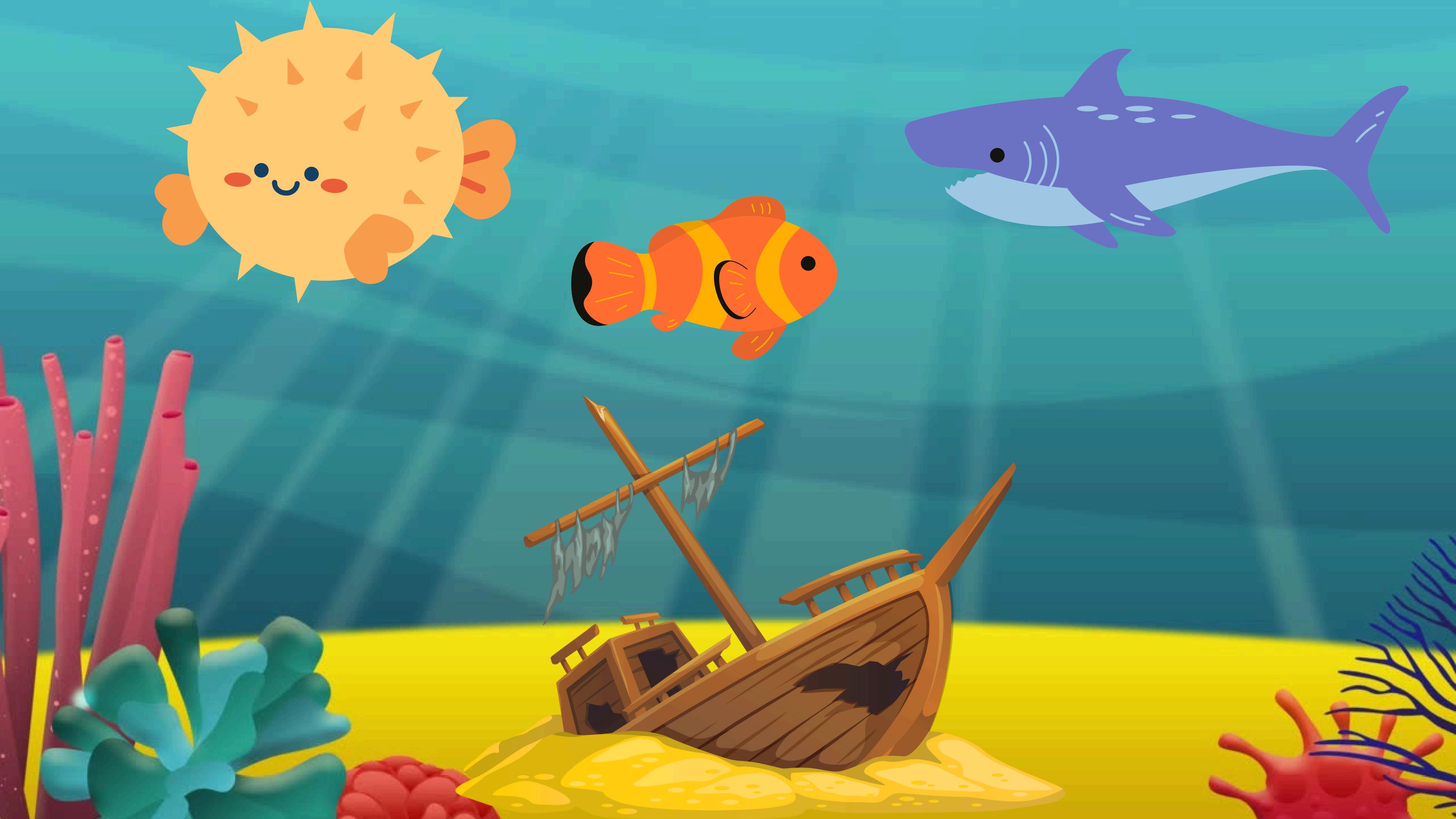
- How do we know when the player clicks the mouse and where?
- Building Shoot the fruit.

HOW DO GAMES WORK?













WHY PYGAME?

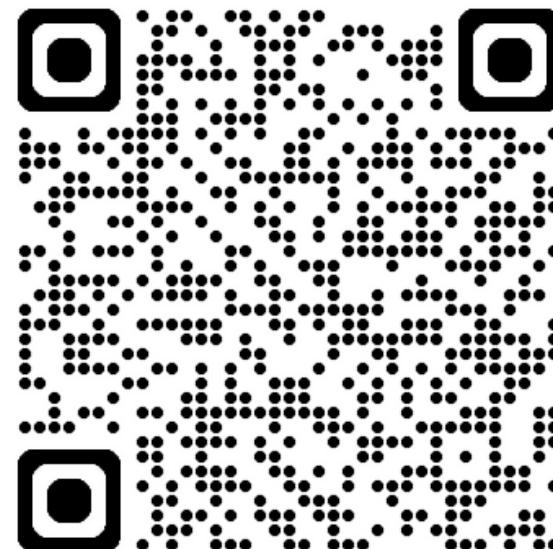


George
R.R. Martin

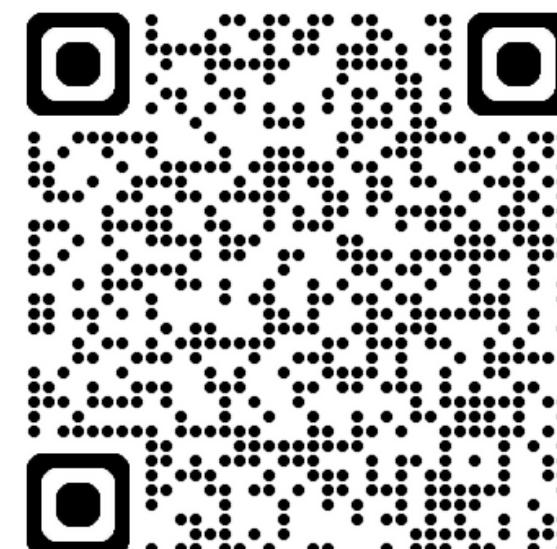


WHY PYGAME?

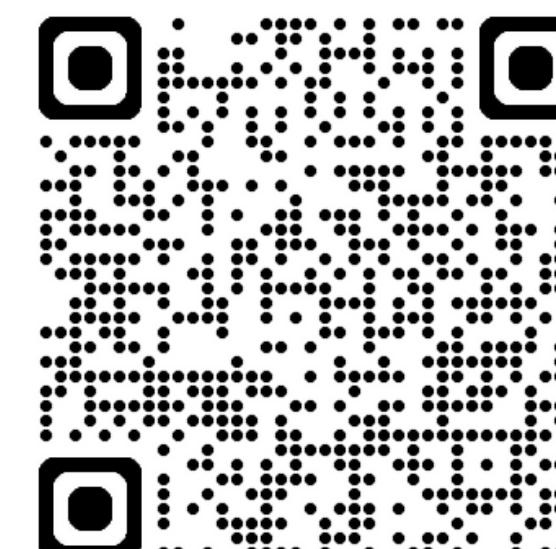
Pygame is a python library used for game development, it enables us to develop 2D games similar to what we saw in the previous video and can be used with other libraries to make 3D games.



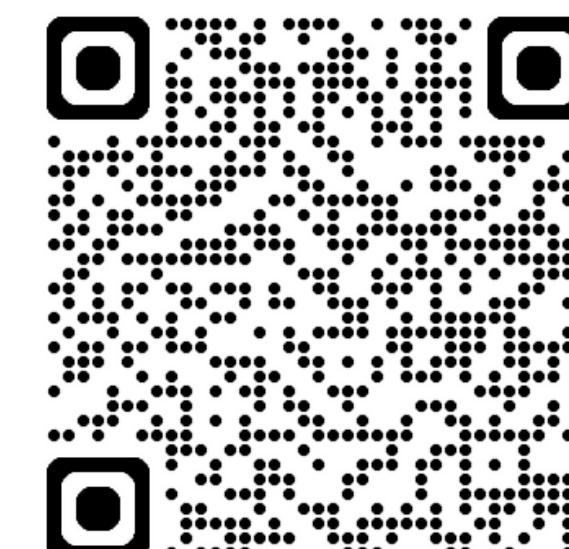
[itch.io pygame games](#)



[Steam pygame games](#)



[Indie Pygame developer](#)



[Games on Pygame site.](#)

Pygame is used to create games that can be sold on steam or itch.io.

Some developers host their games on their own websites, while others work at game studios.

You can check the games on pygame site to see examples of finished games and thier source codes.



PYGAME ZERO

Pygame zero is a version of pygame that is easier to use, it helps us by implementing the game loop and providing the functions we need to make our game.

Pygame provides the basic tools, pygame zero helps us build the game.

In order to develop our game we modify the functions in pygame zero game loop to run our game.

To install pygame zero open a terminal in VS code and run the command:

pip install pgzero

Use the following code in a python script to test your installation:

```
import pgzrun

def draw():
    screen.fill('teal')

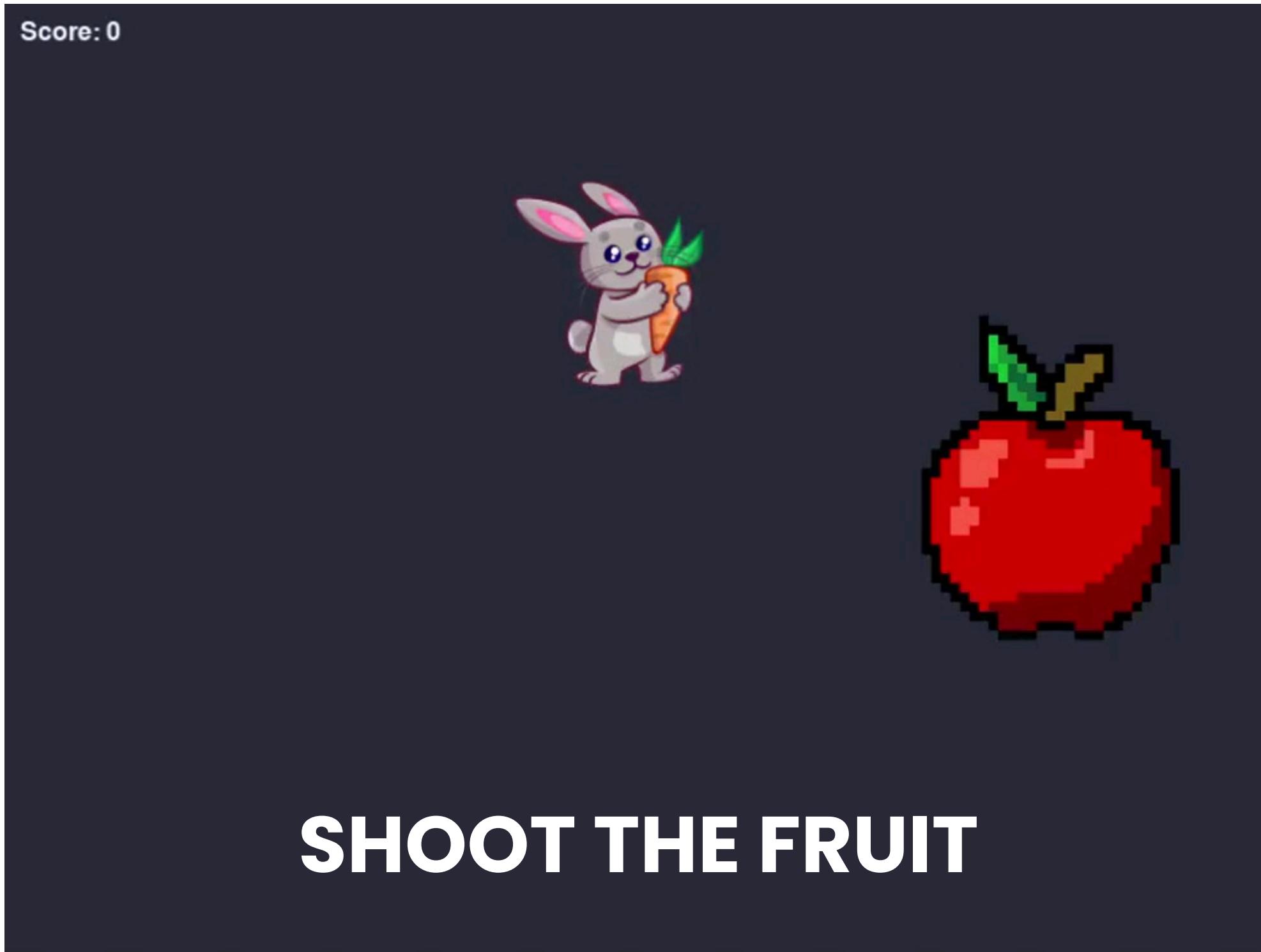
pgzrun.go()
```



[Pygame Zero Documentation](#)



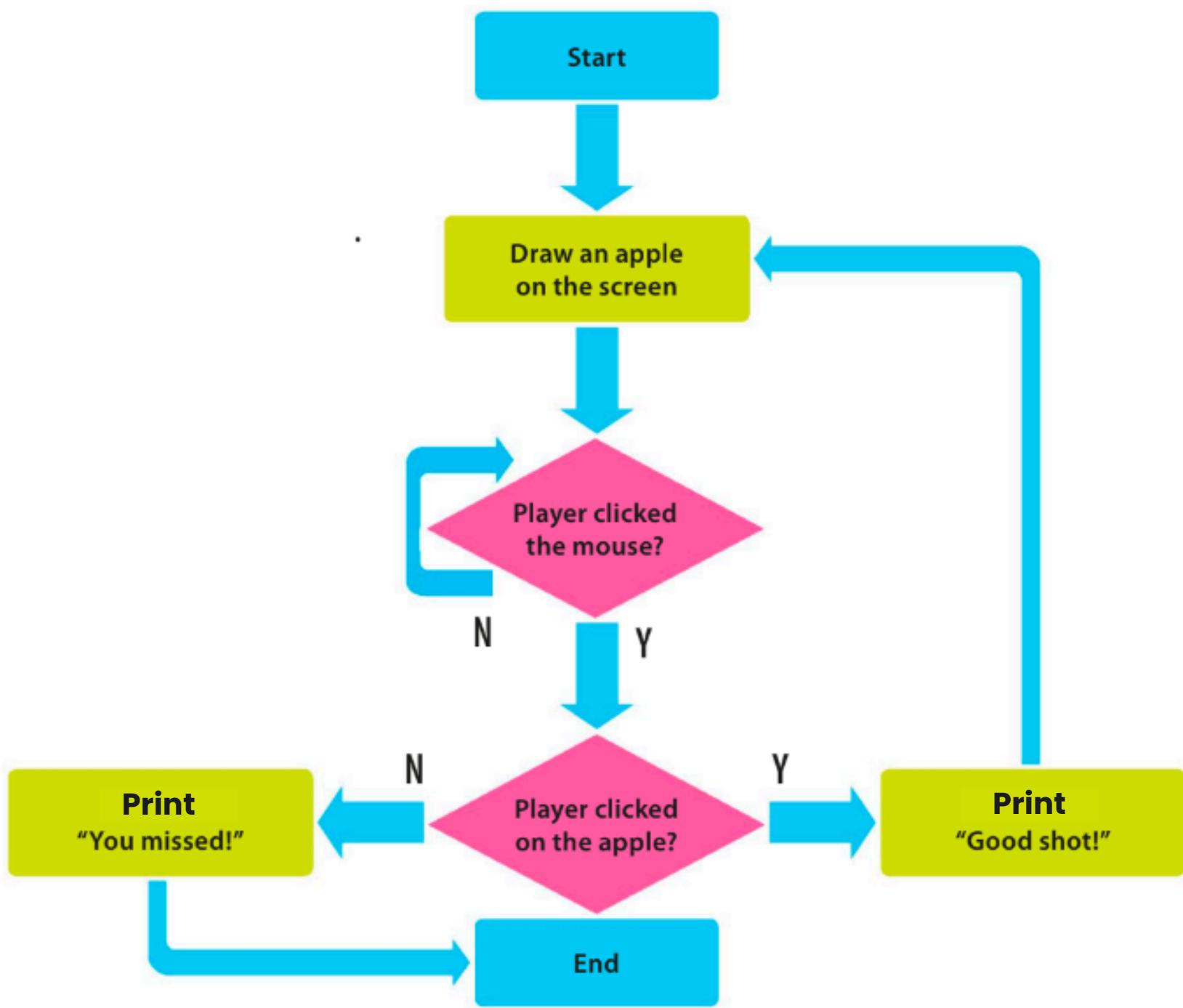
CAN YOU GUESS WHAT WE'LL CREATE TODAY?



HOW DO WE MAKE SHOOT THE FRUIT?



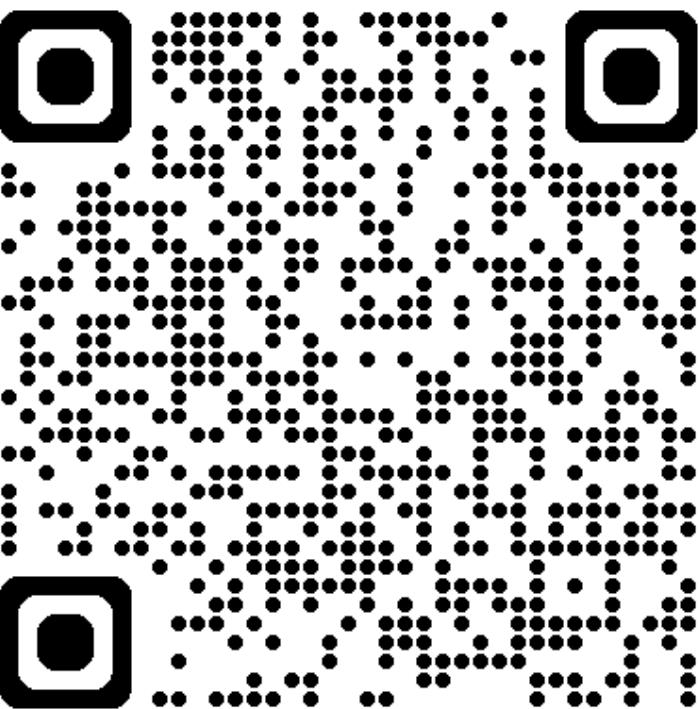
HOW DO WE MAKE SHOOT THE FRUIT?



ACTIVITY#1

Try the following code

```
import pgzrun  
  
color = input("Enter the color: ")  
  
def draw():  
    screen.clear()  
    screen.fill(color)  
  
def update():  
    #This allows the window to update when we move it,  
    #even if it's empty.  
    pass  
  
pgzrun.go()
```

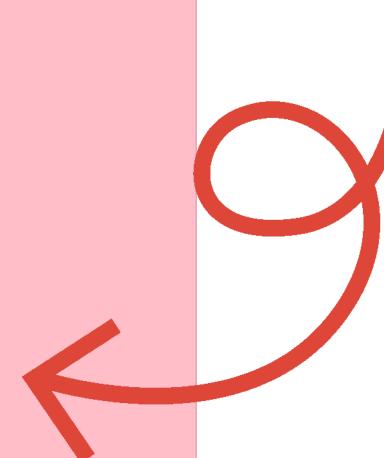
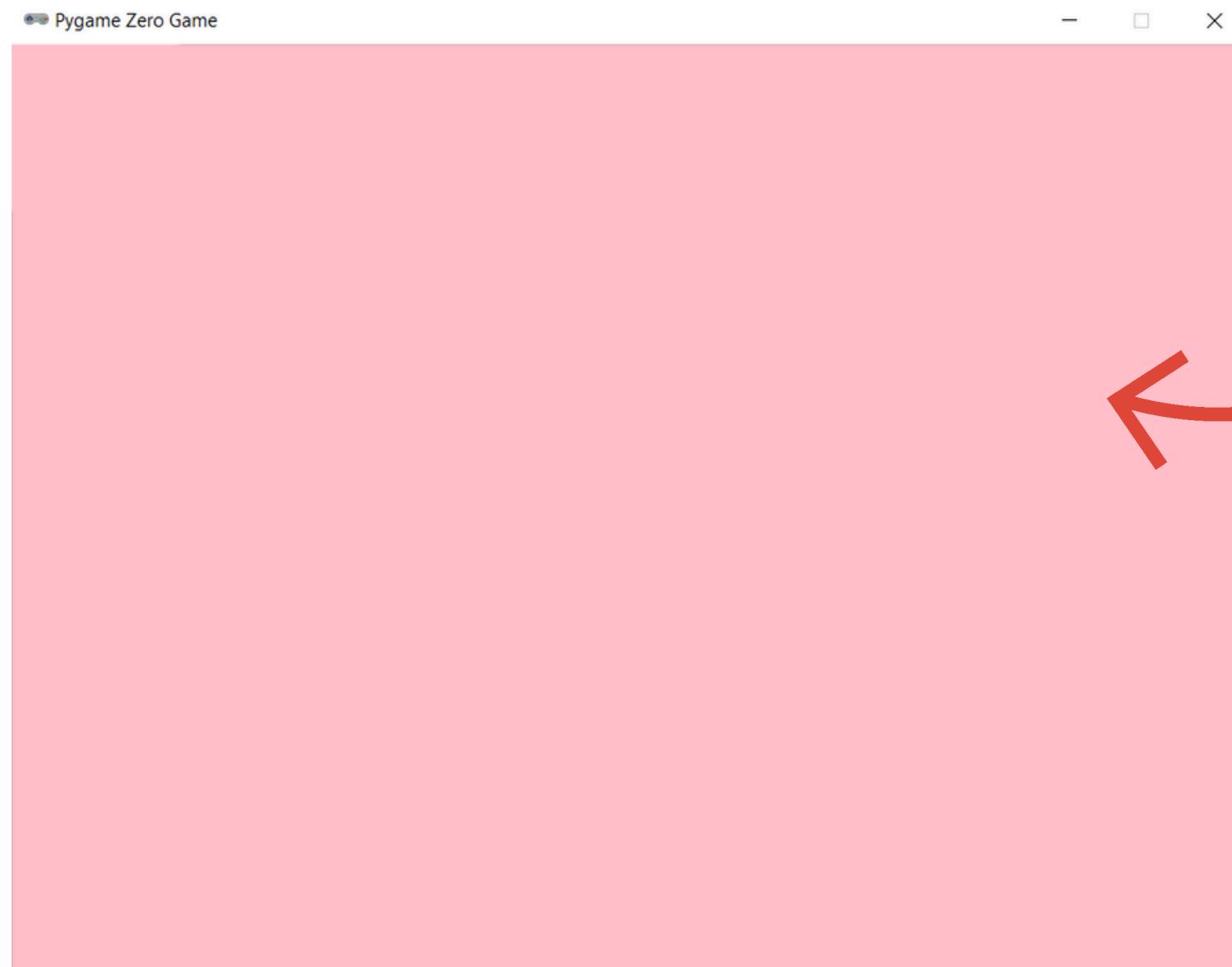


The colors available in
pygame zero



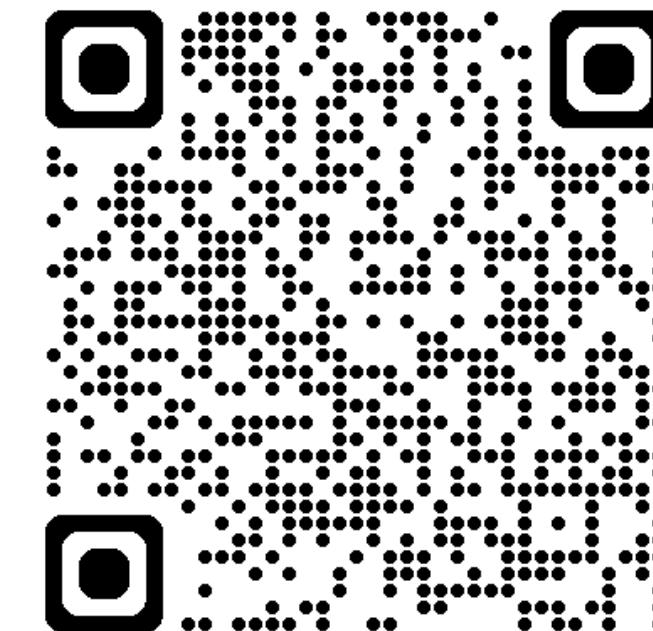
ACTIVITY#1

OUTPUT:



**Background color
should match the
color you picked**

#HINT: you can try different
colors using this website.



The colors available in
pygame zero



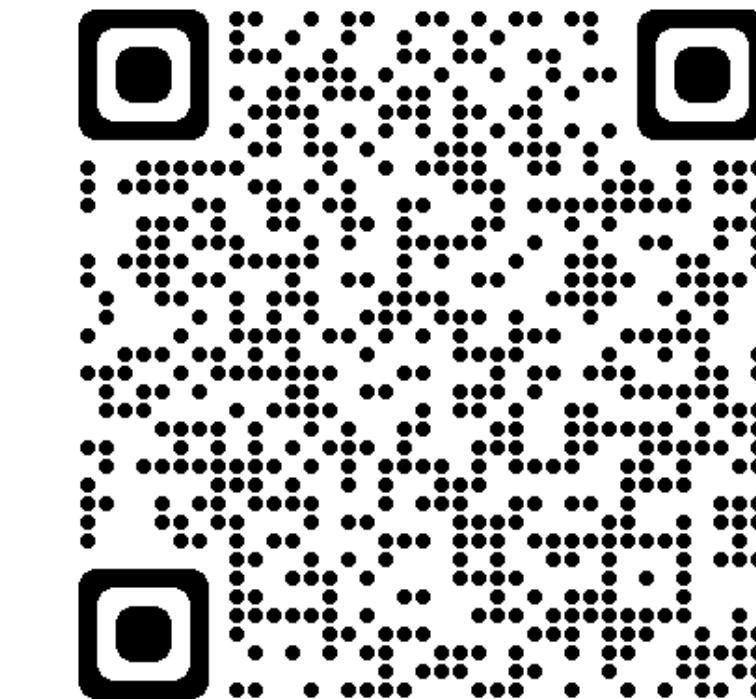
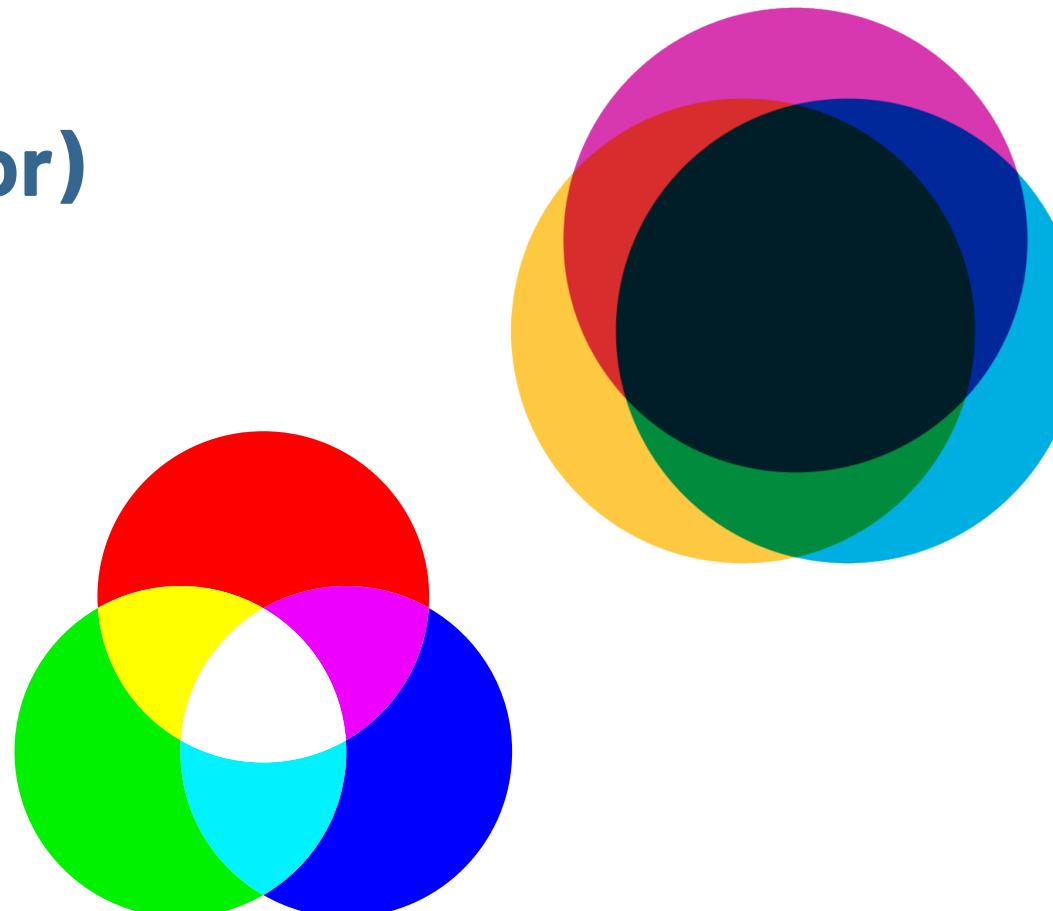
ACTIVITY#2

Modify your code and try

```
import pgzrun
#Change the lines in yellow
color = eval(input("Enter the color as a tuple (e.g., (255, 0, 0)): "))
#The eval() function transforms the input from a string to a tuple.
def draw():
    screen.clear()
    screen.fill(color)

def update():
    pass

pgzrun.go()
```

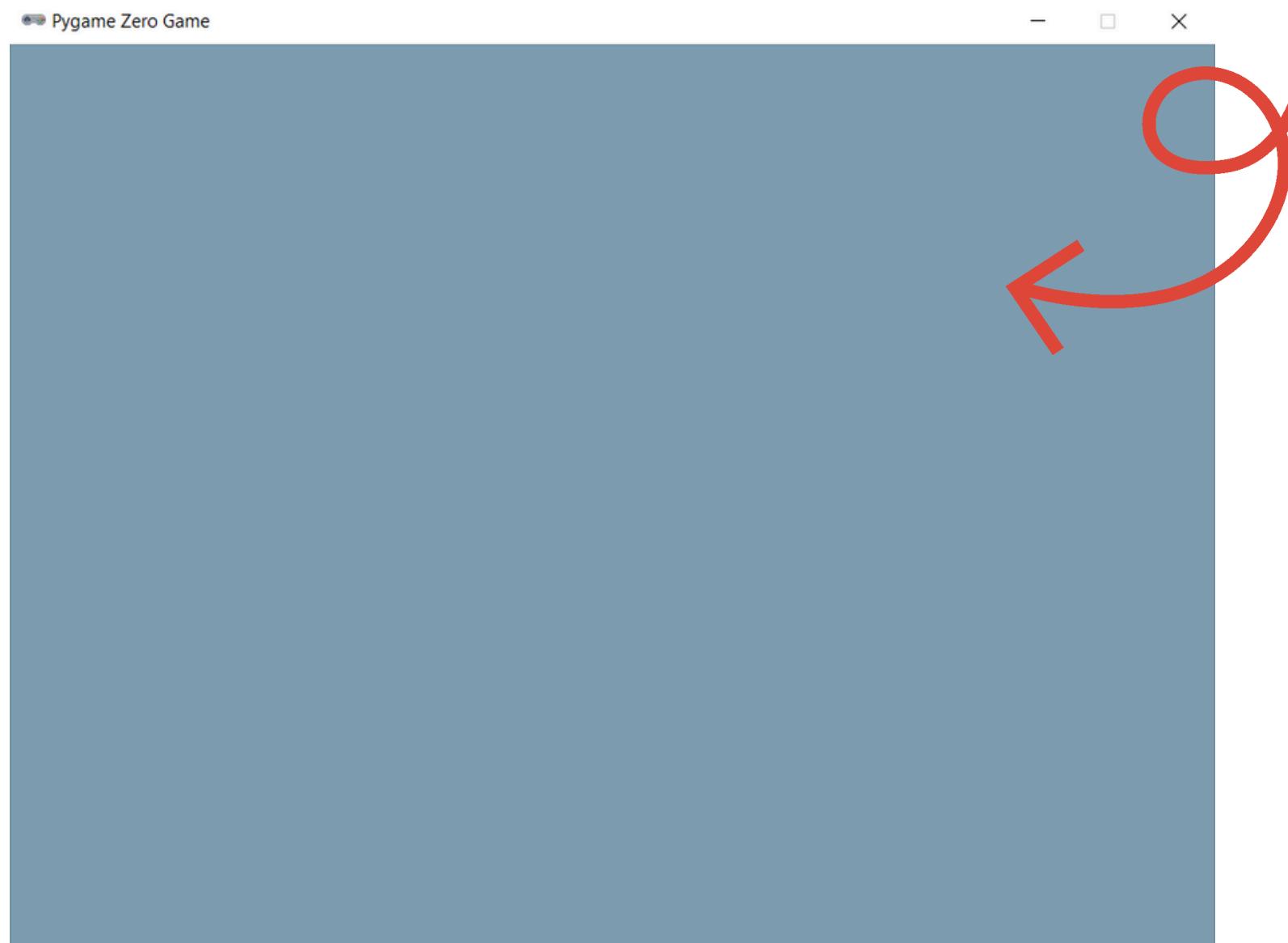


Use this website to determine the
RGB values of colors



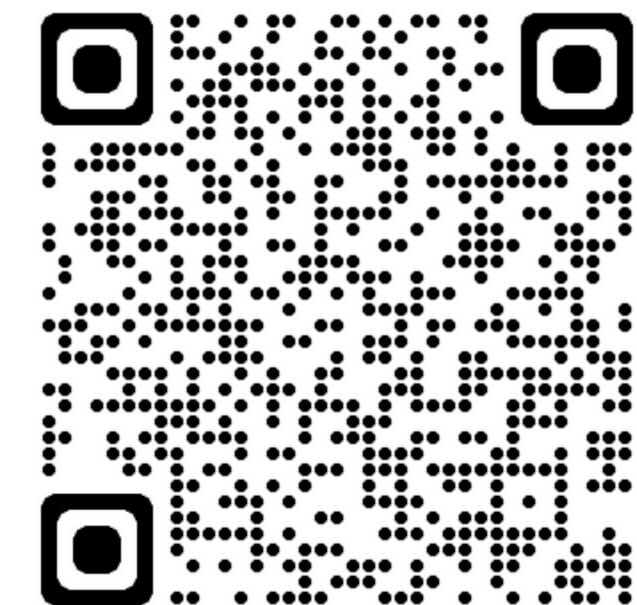
ACTIVITY#2

OUTPUT:



**Background color
should match the
color you picked**

#HINT: you can try different
colors using this website.



[Use this website to determine the
RGB values of colors](#)



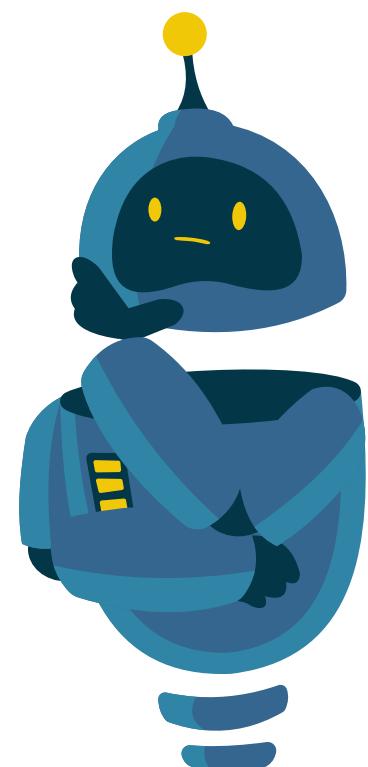
DRAW()

The **draw()** function runs in our main game loop to draw all the elements in our game.

It is responsible for filling the background color in the previous example and any function that is meant to draw anything to our game screen should go inside the **draw()** function.

Syntax:

```
def draw():
    #The different functions used to draw anything to our screen
```



ACTIVITY#3

Modify your code and try

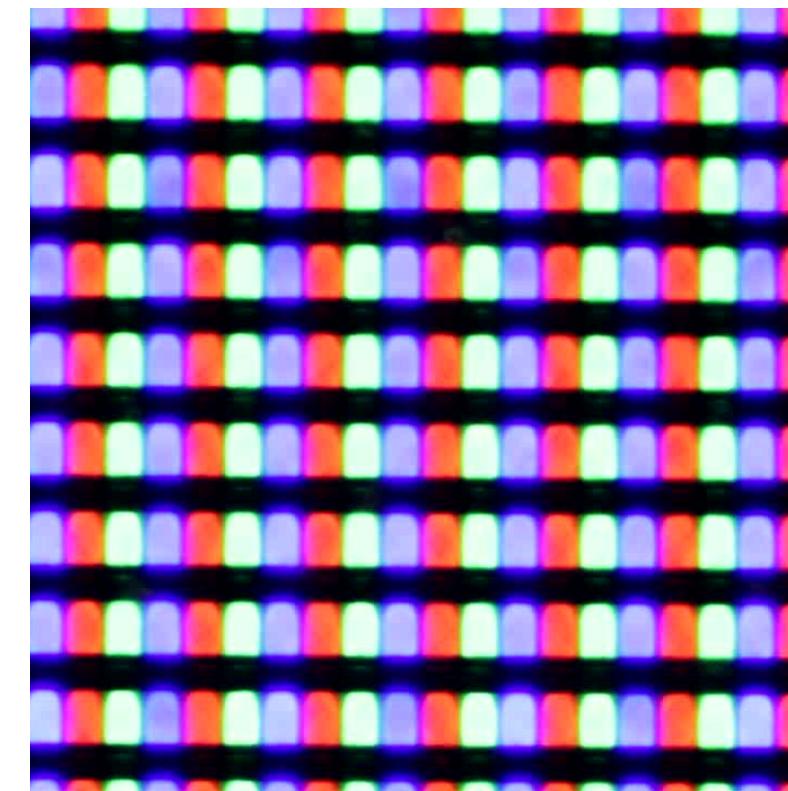
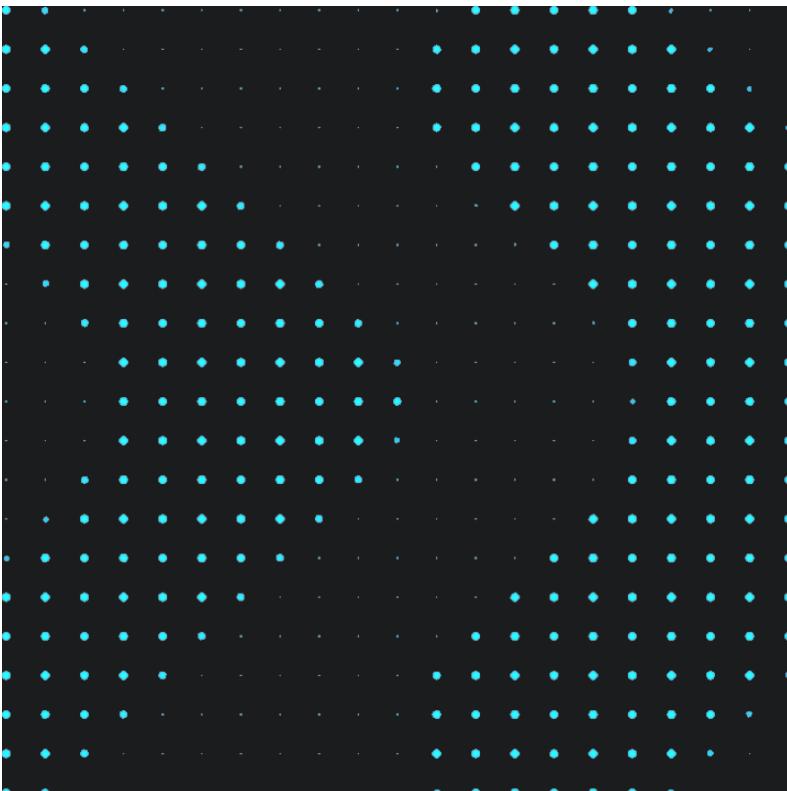
```
import pgzrun  
#Change the lines in yellow  
WIDTH = int(input("Enter the screen width: "))  
HEIGHT = int(input("Enter the screen height: "))
```

```
# You can choose any color you want. We are using slate gray for the game.  
color = "slate gray"
```

```
def draw():  
    screen.clear()  
    screen.fill(color)
```

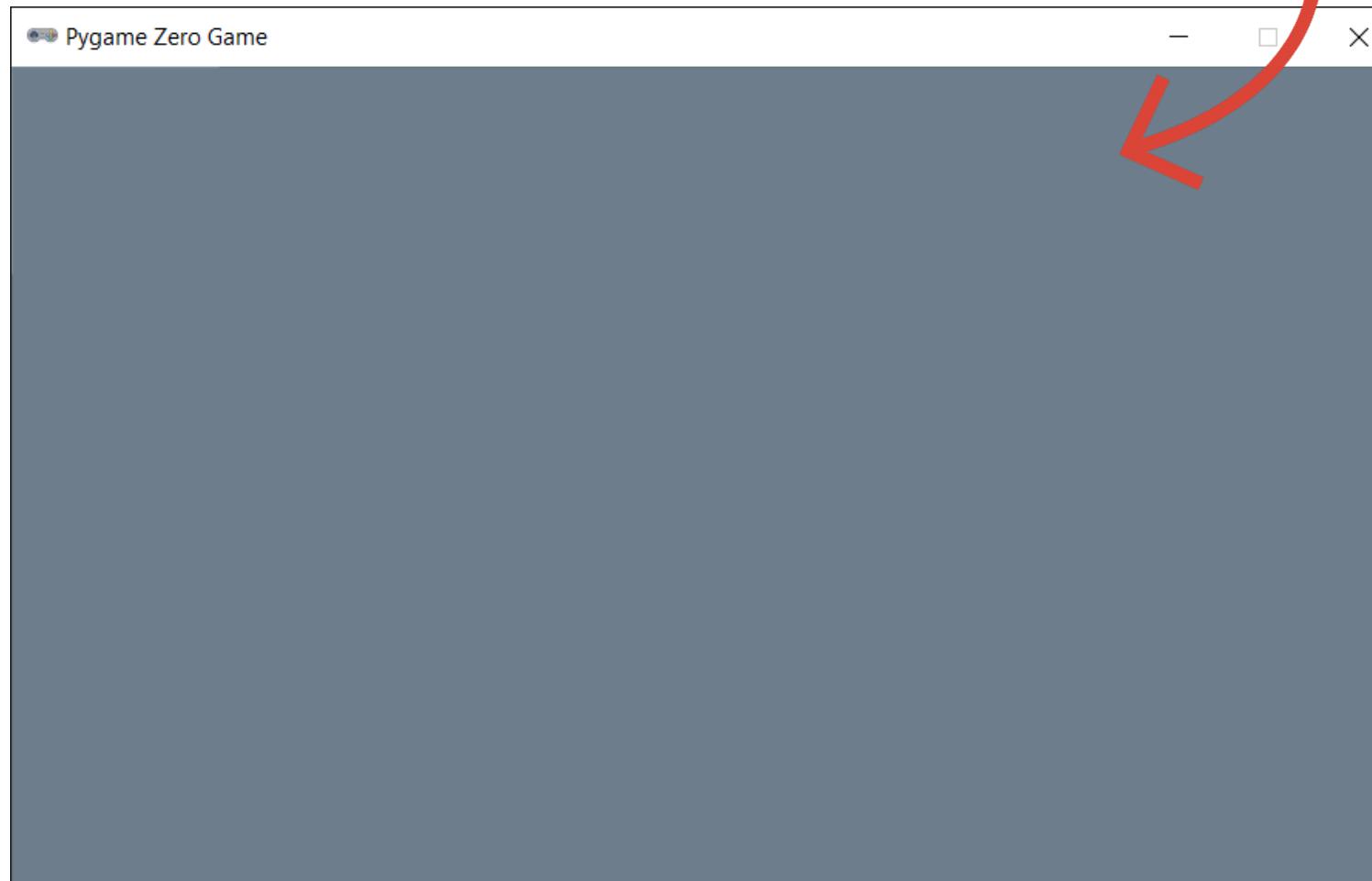
```
def update():  
    pass
```

```
pgzrun.go()
```



ACTIVITY#3

OUTPUT:



Window size should
match the size you
picked

#HINT: try these resolutions to
make the window fill your screen:

WIDTH: **1920** HEIGHT: **1080**

WIDTH: **1366** HEIGHT: **768**

WIDTH: **1280** HEIGHT: **720**

WIDTH: **2560** HEIGHT: **1440**

Note that windows scales apps
according to display settings.



ACTIVITY#4

Modify your code and try

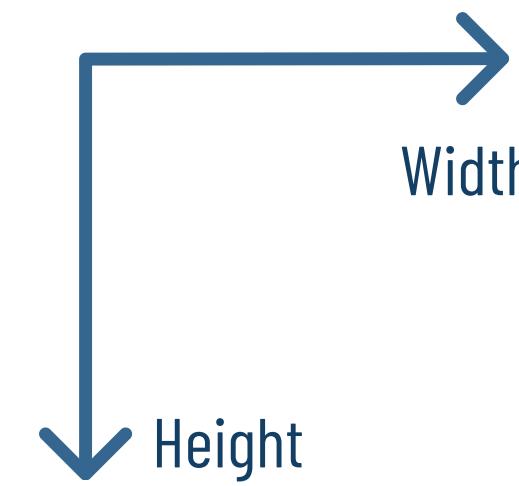
```
#Change the lines in yellow
import pgzrun
# This is the resolution we will use for the game.
WIDTH = 800
HEIGHT = 600

color = "slate gray"

def draw():
    screen.clear()
    screen.fill(color)
    screen.draw.text("Now you are a game developer", (WIDTH/2, HEIGHT/2))

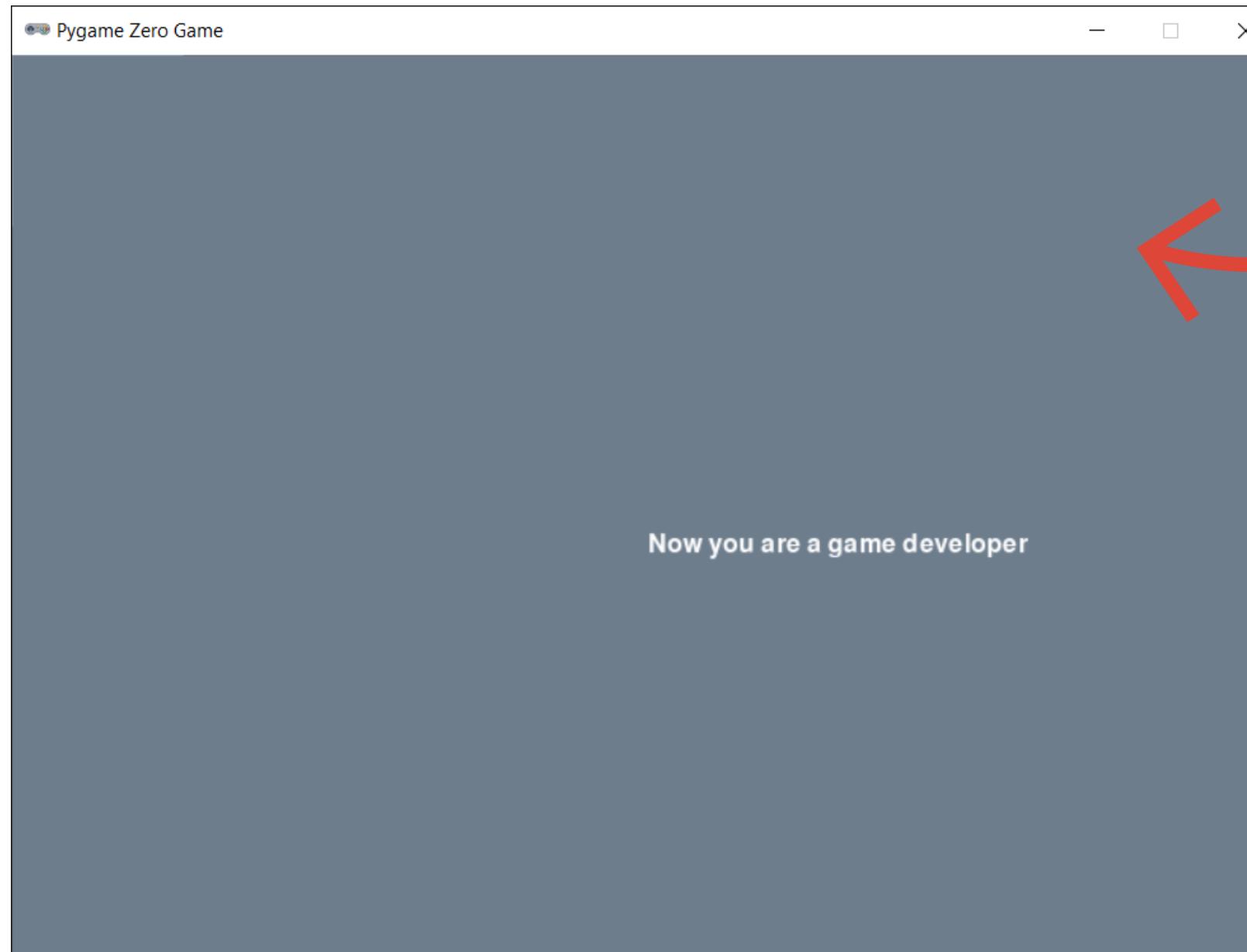
def update():
    pass

pgzrun.go()
```

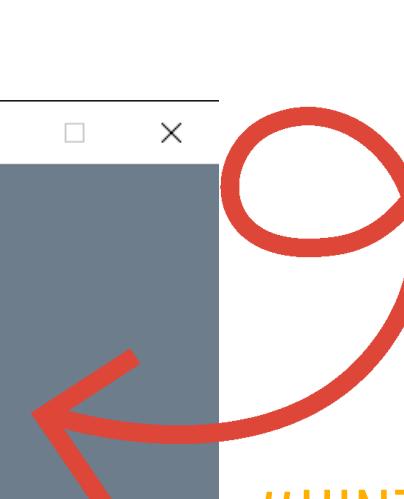


ACTIVITY#4

OUTPUT:



Try to change the
text location



#HINT: in the previous slide the coordinates of the top left corner of the text are determined to be the center of the screen from `(WIDTH/2, HEIGHT/2)`, try passing any other coordinates to the function `draw.text()` as `(100,100)` or `(523,478)` and see what happens.



ACTIVITY#5

Modify your code and try

```
#Change the lines in yellow  
import pgzrun
```

```
WIDTH = 800
```

```
HEIGHT = 600
```

```
TITLE = input("Enter the title of the game: ")
```

```
color = "slate gray"
```

```
def draw():
```

```
    screen.clear()
```

```
    screen.fill(color)
```

```
    screen.draw.text("SHOOT THE FRUIT", (WIDTH/2-90, 10))
```

```
# We've used (WIDTH/2-90, 10) to position the text at the top of the screen at the center.
```

```
def update():
```

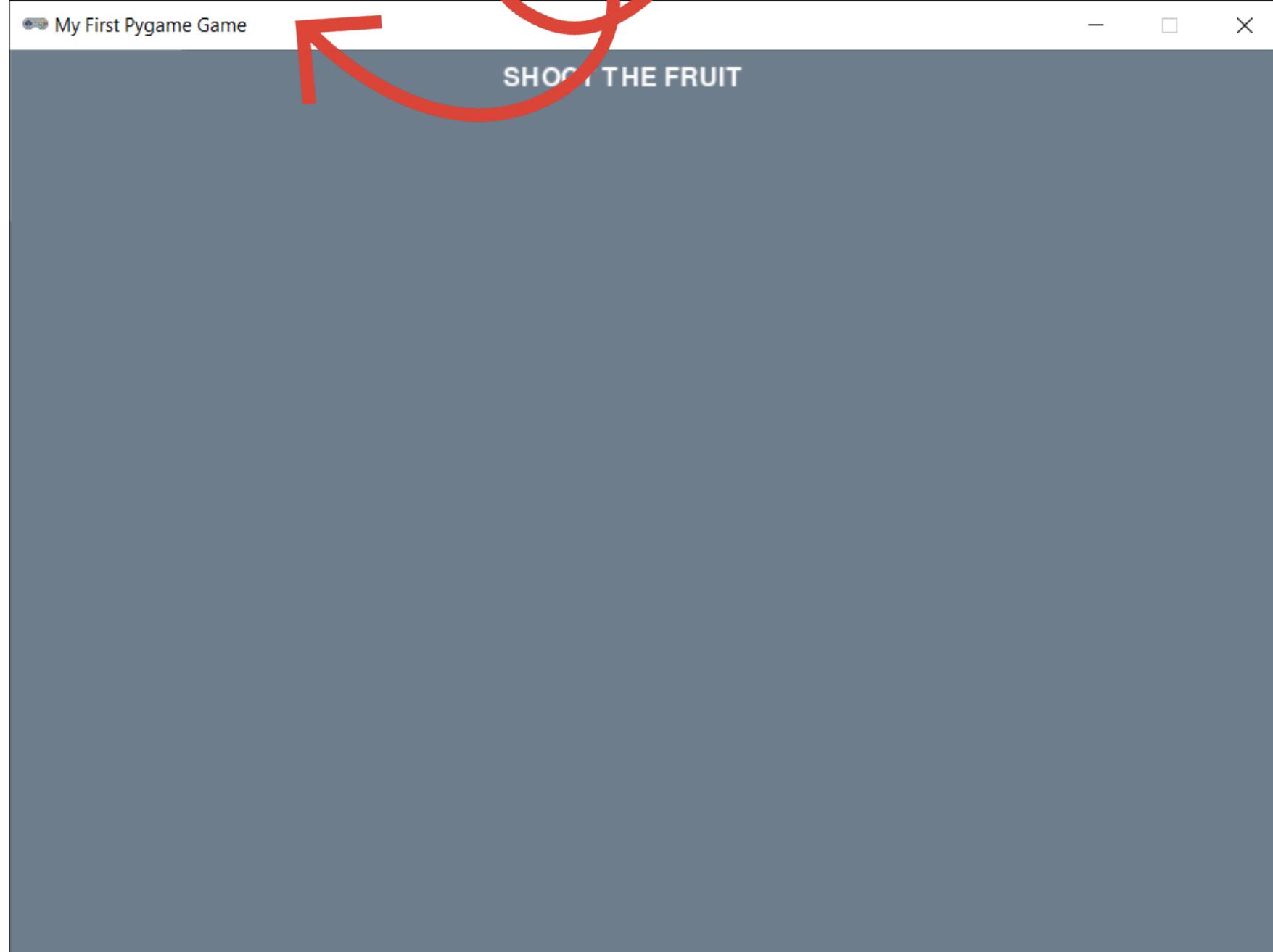
```
    pass
```

```
pgzrun.go()
```

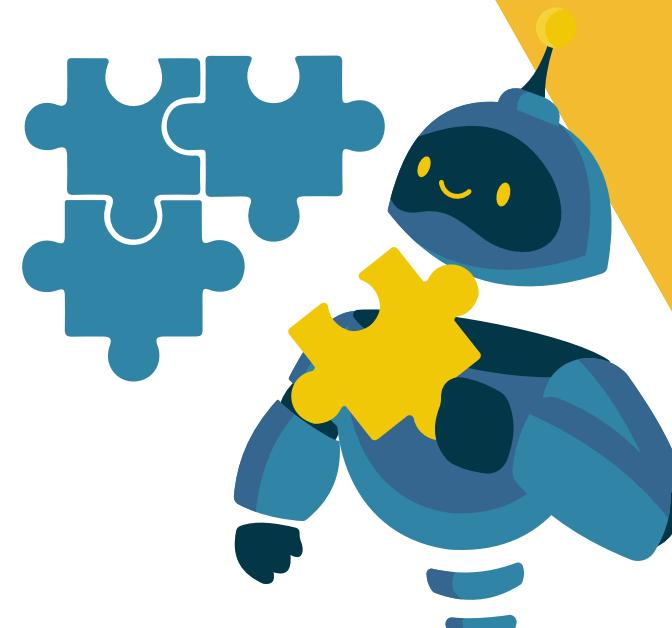


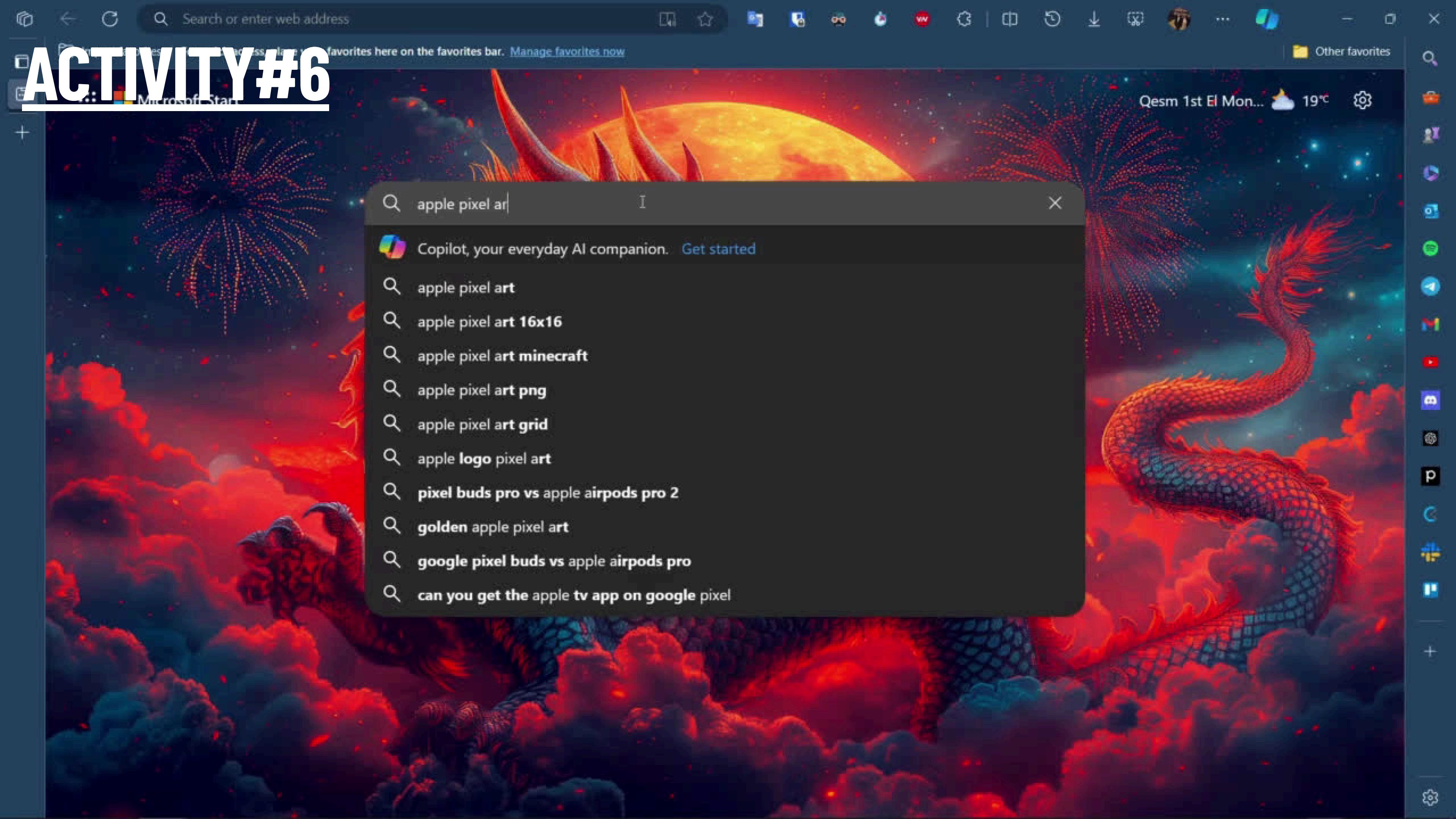
ACTIVITY#5

OUTPUT:



Notice how the
window title changed





ACTIVITY#6

ACTIVITY#6

Modify your code and try

#Change the lines in yellow
import pgzrun

WIDTH = 800
HEIGHT = 600

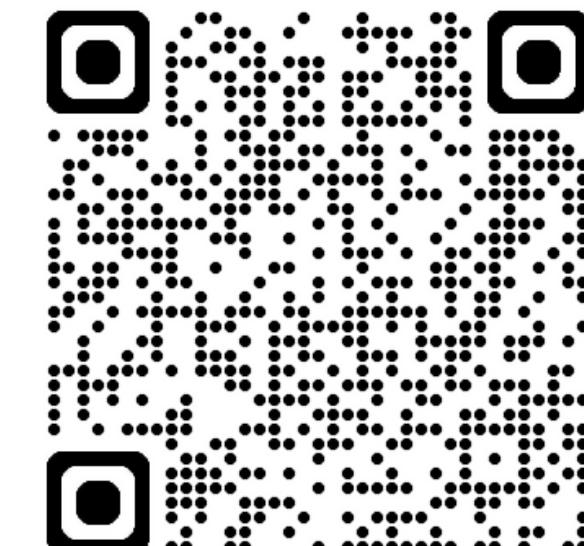
TITLE = "MY FIRST GAME" # We will use this title for the game.
color = "slate gray"

actor = Actor('apple')# Replace 'apple' with the name of your png.
actor.pos = WIDTH/2, HEIGHT/2

def draw():
 screen.clear()
 screen.fill(color)
 screen.draw.text("SHOOT THE FRUIT", (WIDTH/2-90, 10))
 actor.draw()

def update():
 pass

pgzrun.go()

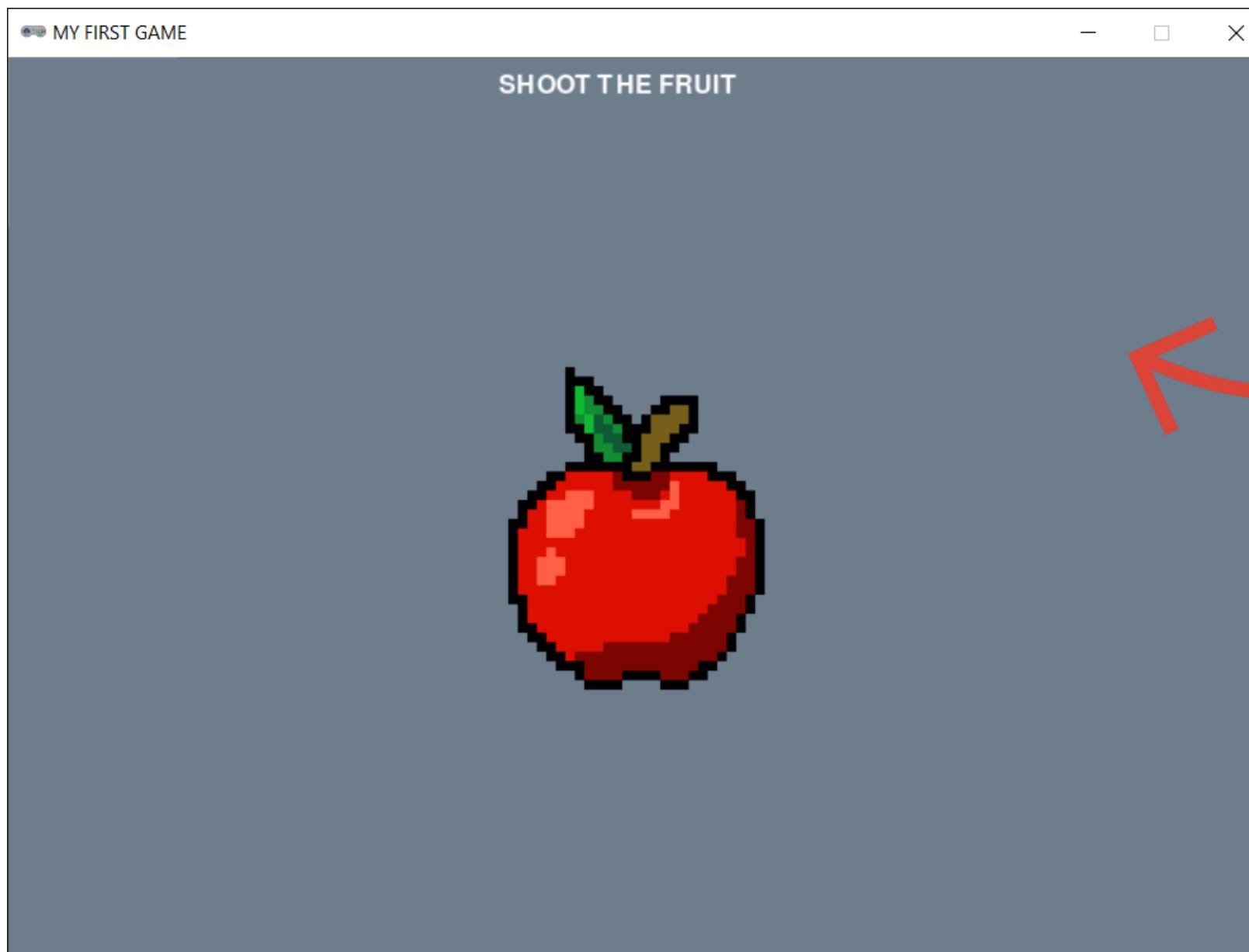


You Can Use This Apple

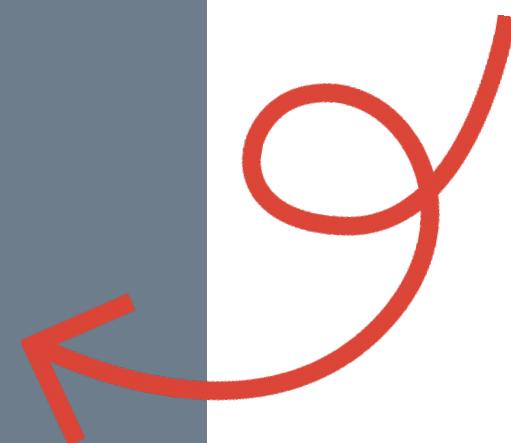


ACTIVITY#6

OUTPUT:



Our apple is now in the middle of the screen



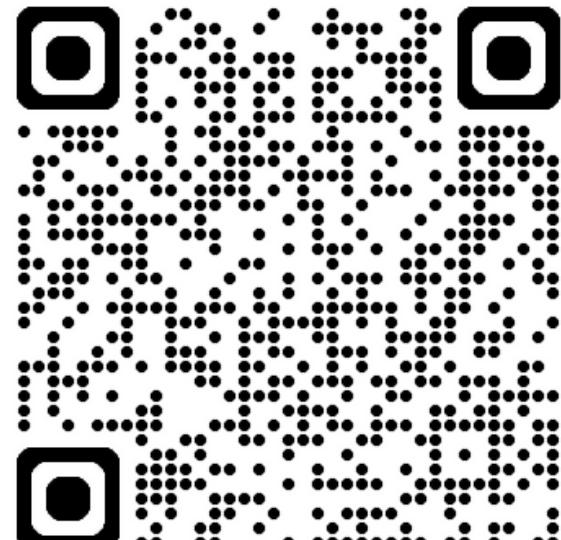
#HINT: in the previous slide the coordinates of the apple are determined to be the center of the screen from `(WIDTH/2, HEIGHT/2)`, try giving it any other coordinates as `(100,100)` or `(523,478)` and see what happens.



ACTIVITY#7

Modify your code and try

```
#Change the lines in yellow  
import pgzrun  
  
WIDTH = 800  
HEIGHT = 600  
TITLE = "MY FIRST GAME"  
color = "slate gray"  
  
actor = Actor('apple')# Replace 'apple' with the name of your png  
actor.pos = WIDTH/2, HEIGHT/2  
  
def draw():  
    screen.clear()  
    actor.draw()  
    screen.fill(color)  
    screen.draw.text("SHOOT THE FRUIT", (WIDTH/2-90, 10))  
  
def update():  
    pass  
  
pgzrun.go()
```



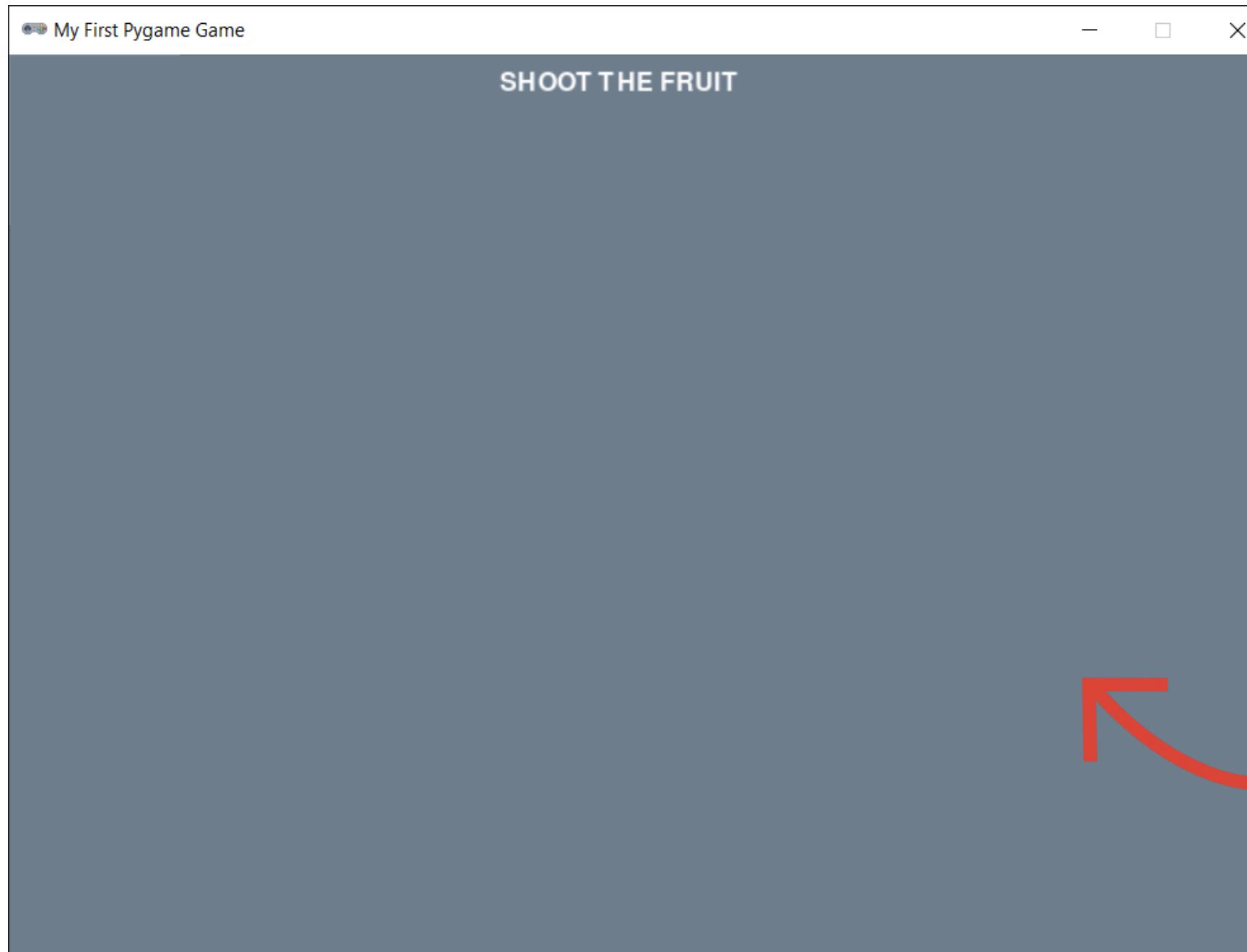
You Can Use This Apple

#NOTE: The actor should be a .png in a folder named images in the same folder as your python script.



ACTIVITY#7

OUTPUT:



**Our apple is hidden
behind the background
fill, Always watch out
for the Drawing order
of your game screen.**



ACTORS

Actors are the building blocks of the game.

They allow us to add characters to the game, move them around and interact with them using the mouse and keyboard to complete the game.

To add an actor to the game, we should :

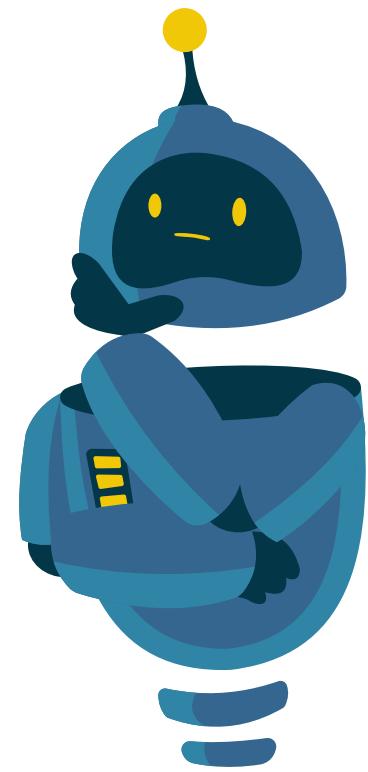
a. Find or create an image to represent the actor.

->We use png files when we want the actor to not have a background

b. Add the image to a folder named images in the same folder as the python script.

c. Add the actor to the script using “`actor_name” = Actor(“actor_file_name_in_images”)` and determining their starting position using “`actor_name”.pos = starting_width, starting_height`

d. Drawing the actor to the screen inside the draw function, while making sure to watch out for drawing order.



ACTIVITY#8

Modify your code and try

#Change the lines in yellow

```
import pgzrun
```

```
WIDTH = 800
```

```
HEIGHT = 600
```

```
TITLE = "MY FIRST GAME"
```

```
color = "slate gray"
```

```
actor = Actor('apple')# Replace 'apple' with the name of your png
```

```
actor.pos = WIDTH/2, HEIGHT/2
```

```
def draw():
```

```
    screen.clear()
```

```
    screen.fill(color)
```

```
    screen.draw.text("SHOOT THE FRUIT", (WIDTH/2-90, 10))
```

```
    actor.draw()
```

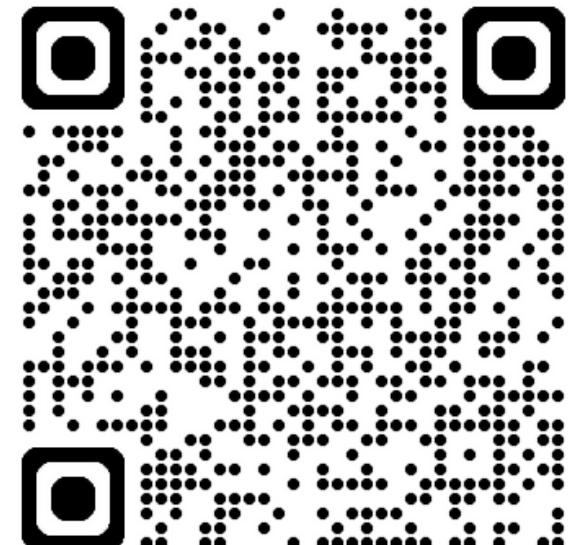
```
def update():
```

```
    actor.x += 2
```

```
    if actor.x > WIDTH:
```

```
        actor.x = 0
```

```
pgzrun.go()
```



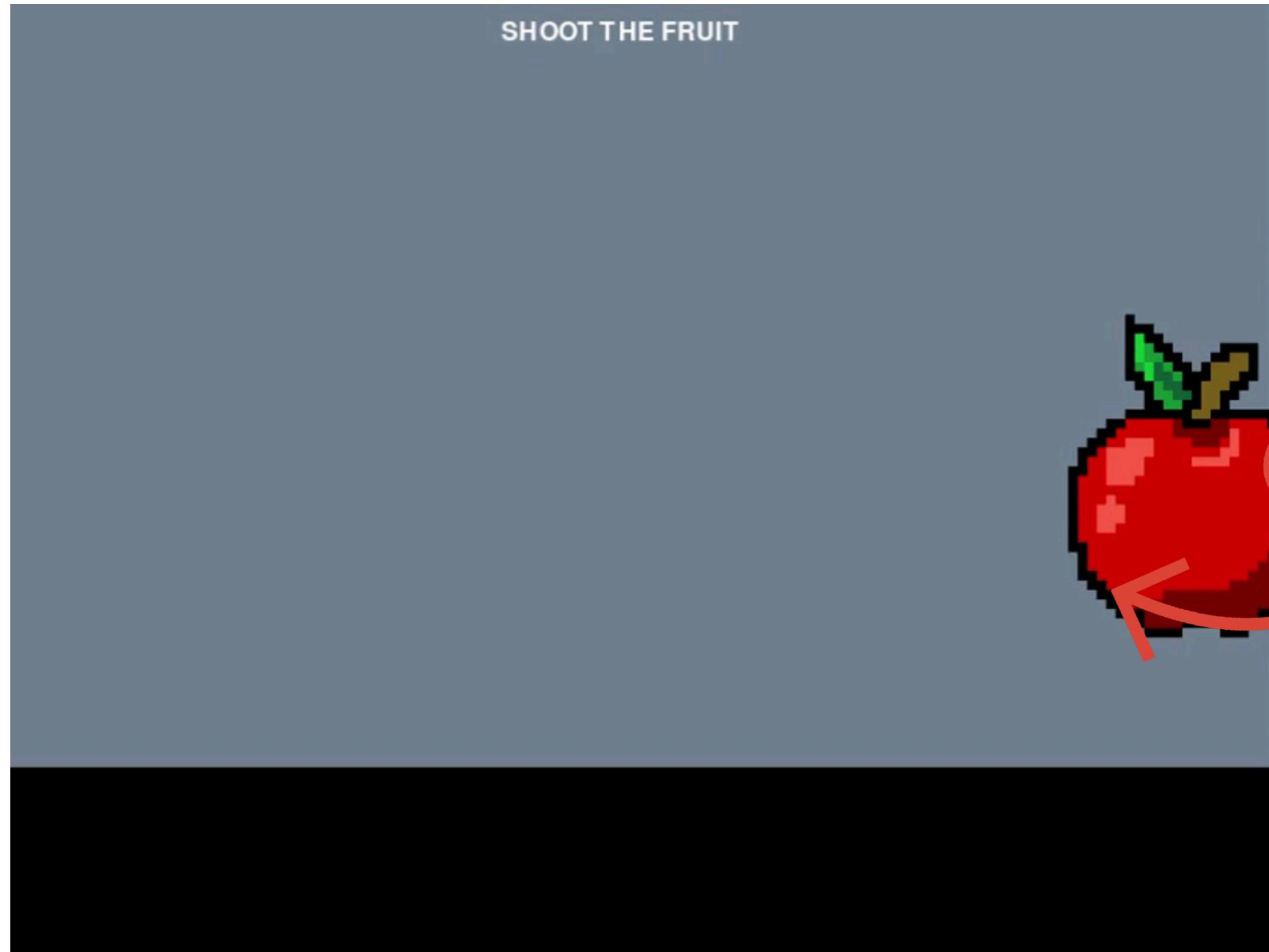
[You Can Use This Apple](#)

#NOTE: The actor should be a .png in a folder named images in the same folder as your python script.



ACTIVITY#8

OUTPUT:



**Our apple is now
moving across the
screen**



UPDATE()

The **update()** function runs after the **draw()** function to update our game.

It's where our game logic resides to do different functions.

In the previous example it moves our actor by 2 pixels before the draw functions draws it again, so that our actor moves across the screen.

Syntax:

```
def update():
    #Our game logic to perform different functions
```

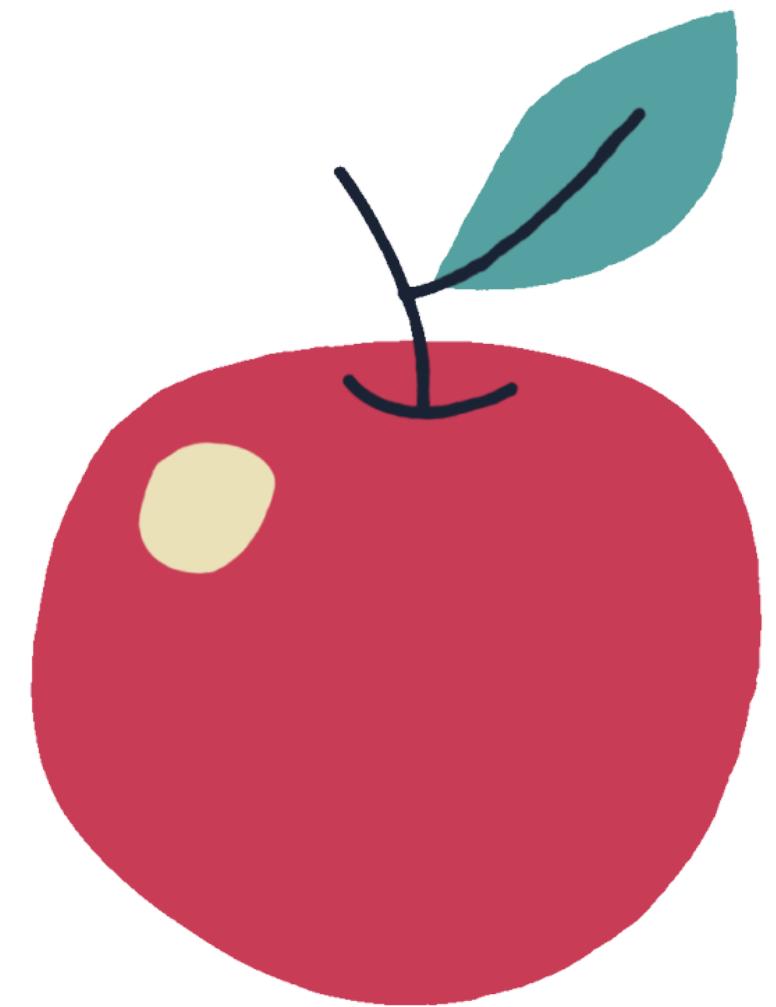


ACTIVITY#9

Add To Your Previous Code & Try It

Add this function after def **draw()**:

```
def on_mouse_down(pos):  
    if actor.collidepoint(pos):  
        actor.pos = WIDTH/2, HEIGHT/2  
        print("Good shot!")  
    else:  
        print("You missed me!")
```

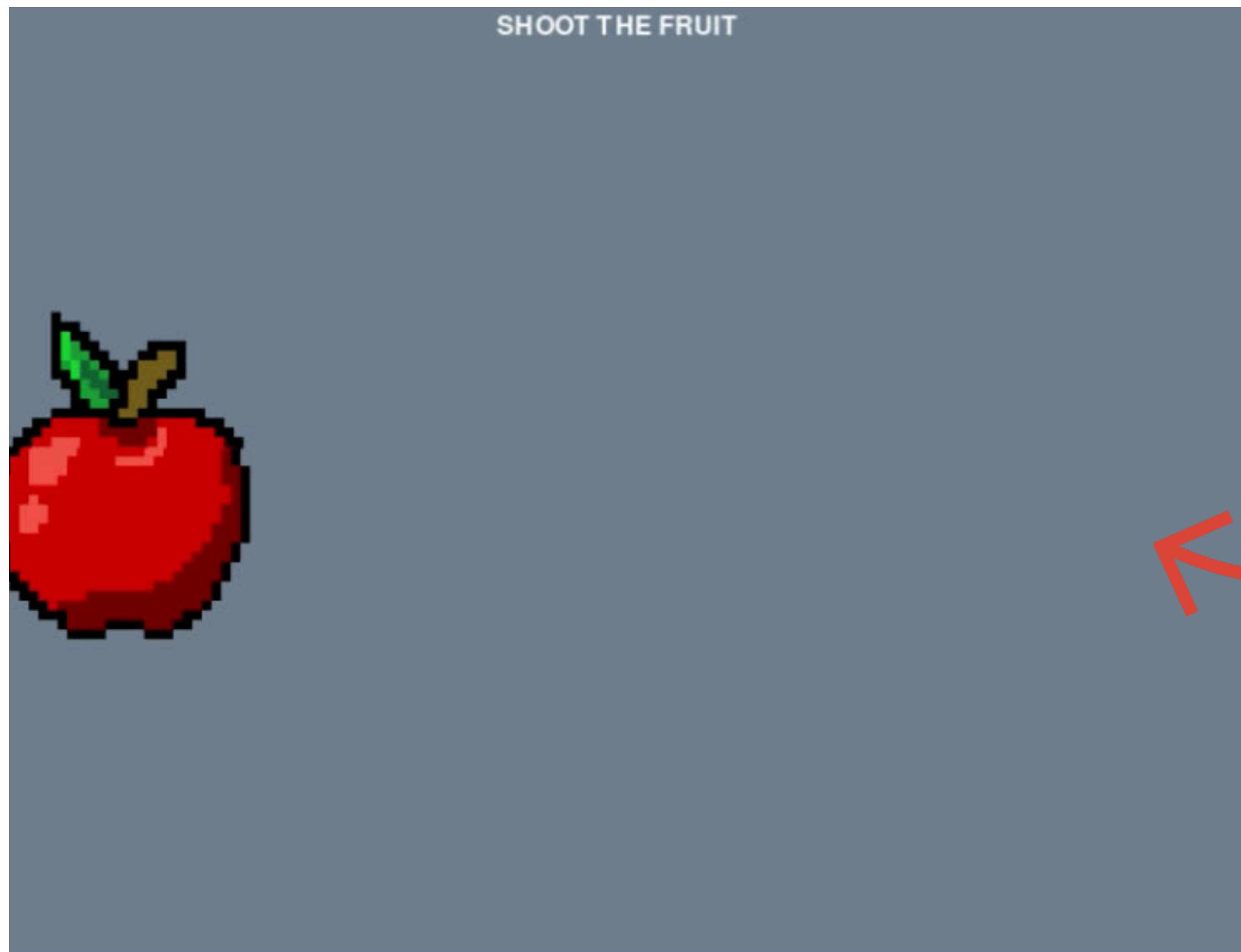


And before def **update()**:

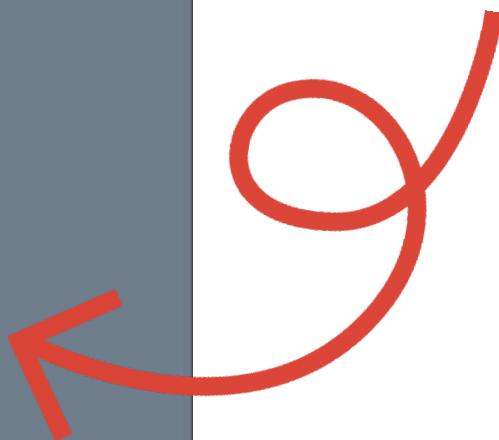


ACTIVITY#9

OUTPUT:



**Now when we click on
the apple it returns to
it's start position**



MOUSE CLICKS

One of the ways we can interact with our game is mouse clicks.

We use the function **on_mouse_click(pixel_coordinates)**: to tell pygame what should happen when we click anywhere on our screen.

It receives the pixel coordinates at which we clicked so that we can determine what to do with the click.

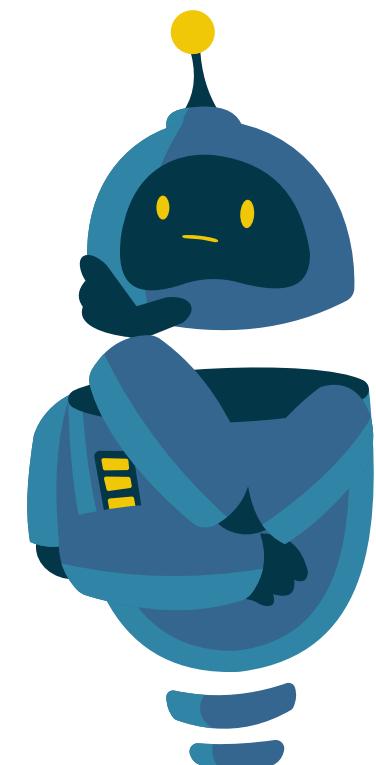
In the previous example it received **pos** which is where we clicked and used the function **actor.collidepoint(pos)** to determine if we clicked on our apple or not.

Syntax:

```
def on_mouse_down(pixel_coordinates):  
    # The code to be executed after mouse click.
```

The function “actor_name”.**collidepoint(pixel_coordinates)** can be used to detect if the pixel coordinates of our click are on our actor “actor_name”

pixel_coordinates: are the pixel coordinates of where we clicked.



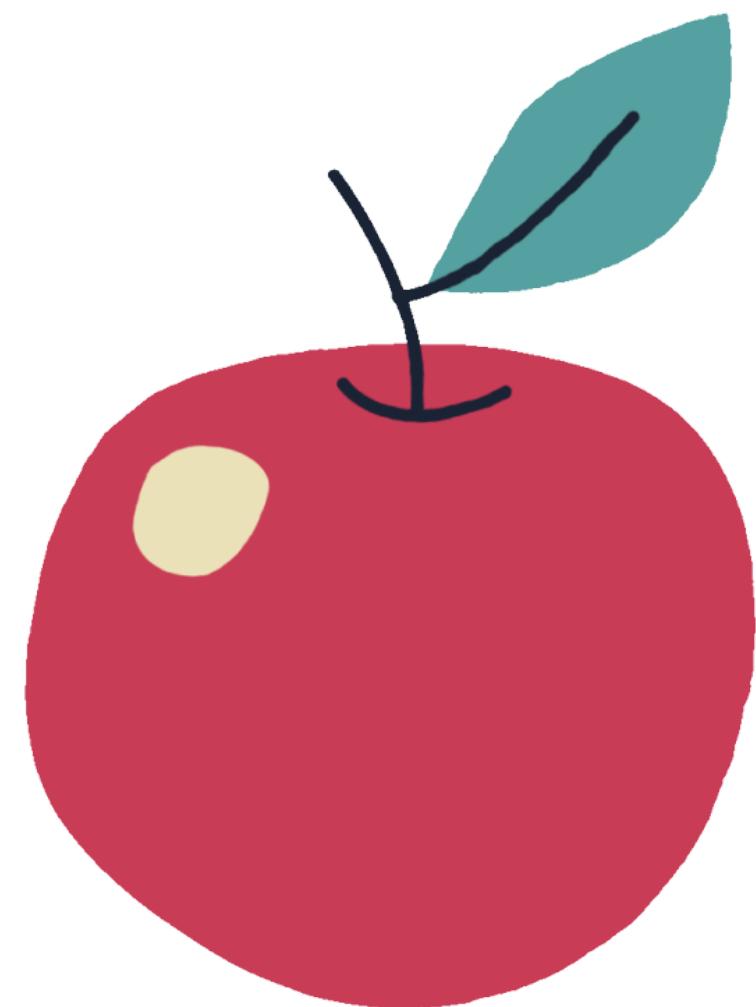
ACTIVITY#10

Add To Your Previous Code & Try It

Add this import
import **random**

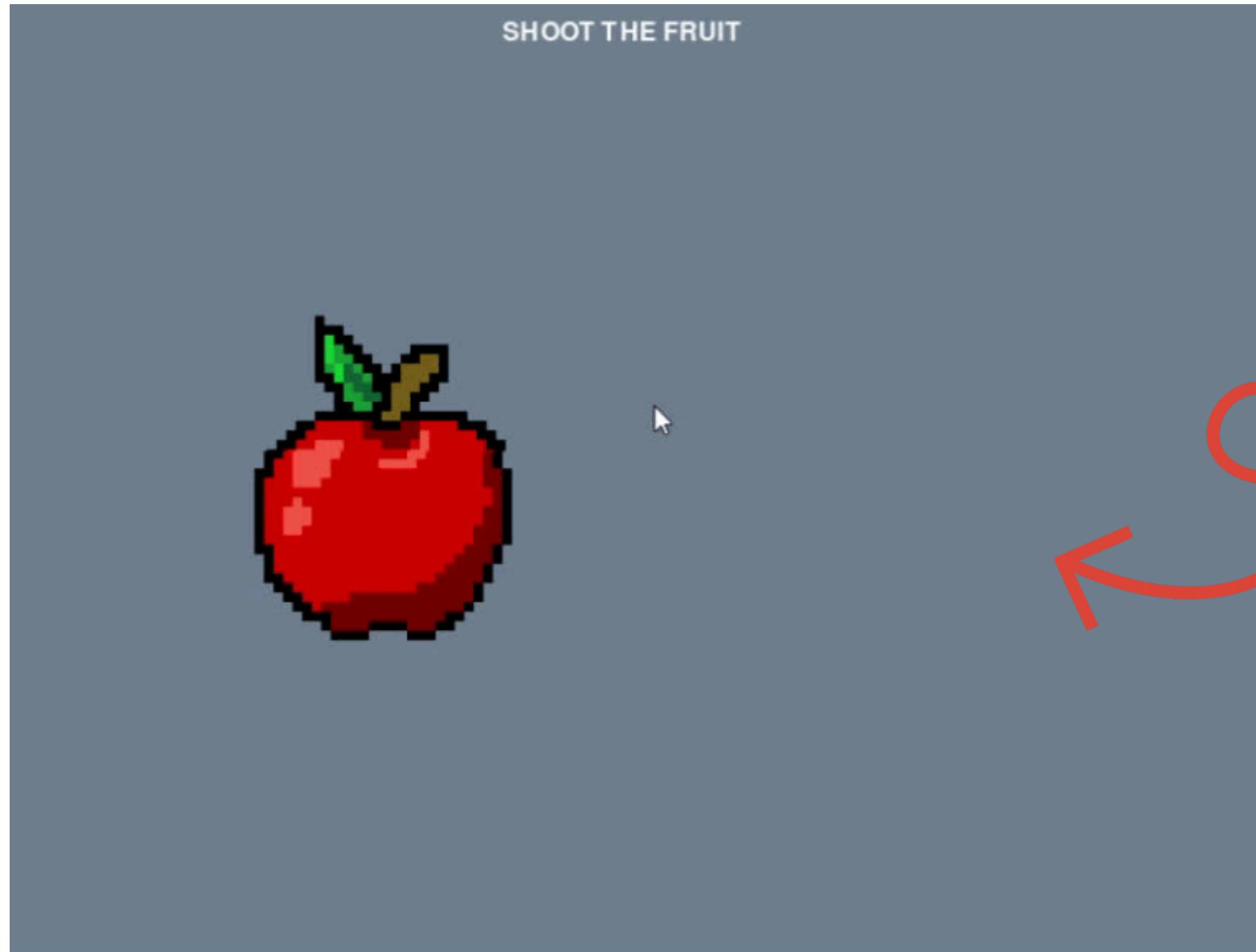
Modify the following function.

```
def on_mouse_down(pos):  
    if actor.collidepoint(pos):  
        actor.pos = random.randint(100, 700), random.randint(100, 500)  
        print("Good shot!")  
    else:  
        print("You missed me!")  
        quit()
```



ACTIVITY#10

OUTPUT:



**Now we have
completed SHOOT THE
FRUIT**



SHOOT THE FRUIT

Now we have finished our first game and you are officially a game developer.

What's next? what additions do you want to add to the game?

How can we increase the game speed?

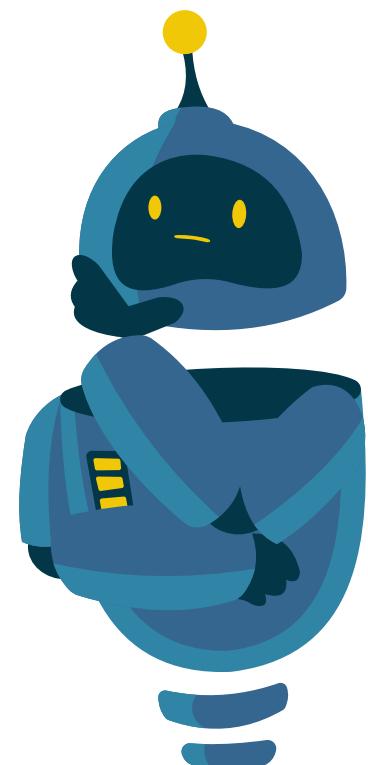
Can we use different Fruits?

HINT: To match the game in the second

slide use `actor.x += 6`

`background color = (40, 43, 59)`

To change fruits you should change the actor
image



ASSIGNMENT

1

NOTES

Write session notes

2

SOLVE ASSIGNMENT

ASSIGNMENT

Continuing on the code that you wrote during the session.

Task1: Modify your code so that it has a game speed of 6 and make your background color (40, 43, 59), with the apple image as your actor.

Task2: Add a score counter to the top left corner of your screen that increases every time you shoot the fruit, also print the final score to the terminal at the end of the game.

Task3: Make the game only decrease your score when you miss instead of quitting.

Task4: Add another Actor to your game that is an animal that moves to the left with a rate of 4 and jumps randomly to another location when it leaves the screen, shooting this animal will quit the game and make you lose.



VORTEX ACADEMY
THE ART OF THINKING

THANK YOU
FOR YOUR ATTENTION

