**Session 5**

Introduction to PyGame

# Introduction to PyGame

* PyGame is the library used to create GUI effects in games
* It supports the picture, sound, shapes
* PyGame provides a cross-platform with the system's multimedia hardware like keyboard, joystick, sound, etc
* Difference between Turtle library and PyGame - The Turtle package is a tool to help draw Graphical output - On the other hand, PyGame is a game development package that gives you all the tools to create a game with Python. ---



## Installing PyGame

For Windows

pip install pygame

For Mac (Provided you have python3 installed)

pip3 install pygame

***if the commands don't work it's probably because you have not added python/python3 to the environment variable. Please check the steps to resolve this problem attached in the folder***

## Bare minimum code

The below code creates a simple blank pygame window

import pygame  
import time  
  
  
pygame.init()  
  
screen = pygame.display.set\_mode([500, 500]) *# creates a screen with the said size*  
  
run = True  
**while** run:  
 **pass**  
   
pygame.quit()

Command explanation

* The pygame. display.set\_mode([500, 500]) is used to create a screen of a specified size in pygame. Creating a screen by default creates a surface. The surface is the drawable portion of the screen.

#### PROBLEM: The launcher window does not work properly!!!

You can also see the not responding sign on top of the window.

To understand why this is happening we must understand how an app works. Any app can be divided into 2 parts the

1. The main app window
2. The event handler

* The main window is where we get the graphical outputs
* The event handler is a task that is supposed to run continuously in the background checking for all the events that are happening in the main window.

## Events

* Events are any user interaction that can be caught on the screen.
* This can include keyboard key presses, mouse interaction, etc.

**In the above code, we created a main window when we created the display. Calling the init() function also creates an event handler for us which is running in the background.**

Then what is the issue???????

* The event handler that is running in the background is capturing events(user interactions) and storing them in a  
  list(queue).
* We are supposed to go through this list frequently enough and tell the app what has to be done when a particular event has occurred.
* Even if no action has to be performed after a particular event, reading the event list(queue) has to be done.
* In the above, we have a game loop but the game loop is not reading the event list. Which causes the app to not respond. ---

import pygame  
import time  
  
  
pygame.init()  
  
screen = pygame.display.set\_mode([500, 500]) *# creates a screen with the said size*  
  
run = True  
**while** run:  
 **for** event **in** pygame. event.get():  
 **pass**  
   
pygame.quit()

#### PROBLEM: We won't be able to close the above window!!

## To be able to close the window we need to check if the close button was clicked and break the game loop(while loop).

*#TASK: closing output window without error*

import pygame  
import time  
  
pygame.init()  
  
screen = pygame.display.set\_mode([500, 500]) *# creates a screen with the said size*  
  
run = True  
**while** run:  
 **for** event **in** pygame. event.get():  
 **if** event.type == pygame.QUIT:  
 run = False  
   
pygame.quit()

Command explanation

pygame. event.get() returns us a list of events that the event handler has caught. We then need to go through the list and check manually if a particular event has occurred.

### list of all events that can be captured by the event handler.

| Event | Attributes |
| --- | --- |
| QUIT | none |
| ACTIVE EVENT | gain, state |
| KEYDOWN | key, mod, Unicode, scancode |
| KEYUP | key, mod |
| MOUSEMOTION | pos, rel, buttons |
| MOUSEBUTTONUP | pos, button |
| MOUSEBUTTONDOWN | pos, button |
| JOYAXISMOTION | joy (deprecated), instance\_id, axis, value |
| JOYBALLMOTION | joy (deprecated), instance\_id, ball, rel |
| JOYHATMOTION | joy (deprecated), instance\_id, hat, value |
| JOYBUTTONUP | joy (deprecated), instance\_id, button |
| JOYBUTTONDOWN | joy (deprecated), instance\_id, button |
| VIDEORESIZE | size, w, h |
| VIDEOEXPOSE | none |
| USEREVENT | code |

## Screen / Surface

screen = pygame. display.set\_mode([500, 500])

The above line of code creates a screen. The screen that we create is also a surface in pygame that can be used to draw things onto.

Surface objects have their own set of methods.

Command explanation:

* The most frequently used one is the fill(color) method which fills the screen with a specified color.

import pygame  
import time  
  
  
pygame.init()  
  
screen = pygame.display.set\_mode([500, 500]) *# creates a screen with the said size*  
  
run = True  
**while** run:  
 **for** event **in** pygame. event.get():  
 **if** event.type == pygame.QUIT:  
 run = False  
   
 screen.fill((255,255,255))  
   
pygame.quit()

#### PROBLEM: We just filled the screen with white but then why is the screen not white

* The issue is that the screen has been painted white but has not yet been updated.
* To update the screen we need to flip the display. Fliping is the operation by which the screen is updated.

Command explanation:

pygame. display.flip()

import pygame  
import time  
  
  
pygame.init()  
  
screen = pygame.display.set\_mode([500, 500]) *# creates a screen with the said size*  
  
run = True  
**while** run:  
 **for** event **in** pygame. event.get():  
 **if** event.type == pygame.QUIT:  
 run = False  
   
 screen.fill((255,0,0)) *# RGB (0-255)*  
   
 pygame. display.flip()  
   
   
pygame.quit()

## Drawing shapes

The pygame. draw module has a bunch of functions that can used to draw shapes.

**Drawing circles** pygame. draw.circle(Surface, color , center location , radius)

| Arguments |  |
| --- | --- |
| Surface | The screen or any drawable surface |
| color | Tuple representing the RGB values |
| center location | Tuple representing the x and y coordinate of the center point of the rectangle |
| radius | Scalar values representing the radius of the circle |

**Drawing Rectangles** pygame. draw.rect(Surface, color , rect)

| Arguments |  |
| --- | --- |
| Surface | The screen or any drawable surface |
| color | Tuple representing the RGB values |
| center location | Tuple representing the x and y coordinate of the center point of the rectangle |
| rect | Tuple representing the x and y coordinate of the top left-hand corner of the rectangle along with the width and height of the rectangle(left, top, width, height) |

**Drawing Circle**

import pygame  
import time  
  
  
pygame.init()  
  
screen = pygame.display.set\_mode([500, 500]) *# creates a screen with the said size*  
  
run = True  
**while** run:  
 **for** event **in** pygame. event.get():  
 **if** event.type == pygame.QUIT:  
 run = False  
   
 screen.fill((255,255,255))  
   
 *# (screen, color, center, radius)*  
 pygame.draw.circle(screen, (0, 0, 255), (250, 250), 75,width= 1)  
   
 pygame. display.flip()  
   
   
pygame.quit()

**Drawing Rectangle**

import pygame  
import time  
  
pygame.init()  
  
screen = pygame.display.set\_mode([500, 500]) *# creates a screen with the said size*  
  
run = True  
**while** run:  
 **for** event **in** pygame. event.get():  
 **if** event.type == pygame.QUIT:  
 run = False  
   
 screen.fill((255,255,255))  
*# (screen, color, (left, top, width, height))*  
 pygame.draw.rect(screen, (0, 0, 255), (100,100,300,300),1)  
 pygame. display.flip()  
   
   
pygame.quit()

## Setting the FPS

FPS stands for frames per second. This parameter defines how fast the screen is refreshing. As of now, we don't have any control over the speed at which the while loop is running. We can limit the speed by creating a clock and setting the frame rate using the tick method.

We won't see any difference though as we are not animating anything as of yet. But once we start with the animation we will be able to see the difference.

clock = pygame. time.Clock() The above line of code creates a clock object

clock. tick(30) the above line of code sets the frame rate. A frame rate of 30 means the while loop/ game loop will run 30 times every second. The clock. tick(30) has to be inside the game loop for which the frame rate has to be controlled.

import pygame  
import time  
  
  
pygame.init()  
  
screen = pygame.display.set\_mode([500, 500]) *# creates a screen with the said size*  
  
clock = pygame.time.Clock()  
  
run = True  
**while** run:  
 **for** event **in** pygame. event.get():  
 **if** event.type == pygame.QUIT:  
 run = False  
   
 screen.fill((255,255,255))  
   
 pygame.draw.rect(screen, (0, 0, 255), (100,100,300,300))  
   
 pygame. display.flip()  
 print("hello")  
 clock.tick(30)  
   
   
pygame.quit()

# Animation in PyGame:

* Animation is a series of images which is being shown at a particular frame rate.
* **Let's try making the ball move around on the screen.**
* To make the ball move, all we need to do is change the x and y coordinates of the ball and keep on doing this as the while loop is running.

*#TASK 1: Moving ball on-screen*

import pygame  
import time  
  
pygame.init()  
  
screen = pygame.display.set\_mode([500, 500]) *# creates a screen with the said size*  
  
clock = pygame.time.Clock()  
  
*# variables used to represent the center location of the circle*  
x = 250  
y = 250  
  
run = True  
**while** run:  
 **for** event **in** pygame. event.get():  
 **if** event.type == pygame.QUIT:  
 run = False  
   
 screen.fill((255,255,255))  
   
 pygame.draw.circle(screen, (0, 0, 255), (x, y), 75)  
   
 pygame. display.flip()  
   
 *# changing the x and y coordinate of the circle*  
 x = x+1  
 y = y+1  
   
 clock.tick(30)  
   
   
pygame.quit()

**EXPLANATION OF CODE**

* The x and y coordinate of the circle keeps updating in the while loop itself.
* This causes the circle to be redrawn in the new location every time the loop runs once.
* And since we have set the fps of the game loop by using a clock. tick() the ball will move by 30 times every second.

You can also try running the code without the screen. fill((255,255,255)). This will cause the ball to leave a trail behind. So to avoid the trail behind we fill the screen with the white color again and then redraw the ball. These steps are already being followed in the while loop.

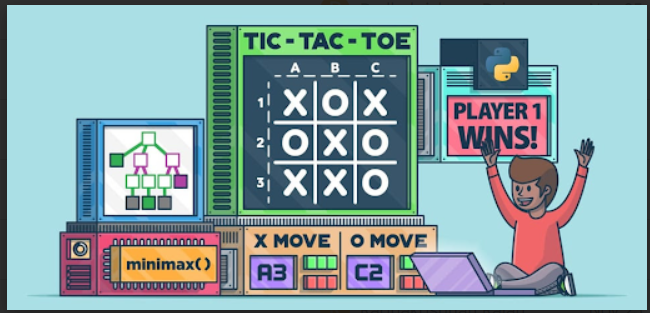
x = x+1  
y = y+1

The +1 controls the speed at which the ball is moving ahead.

The ball's direction is determined by whether the ball's x and y coordinate is increasing or decreasing.

*#task 2: Add 4 more balls each with different, colors, and speeds, and moving in different directions.*   
*# Also make sure the balls have different sizesPractice problem*  
  
import pygame  
import time  
pygame.init()  
screen = pygame.display.set\_mode([500, 500]) *# creates a screen with the said size*  
clock = pygame.time.Clock()  
  
x1 = 250  
y1 = 250  
  
x2 = -100  
y2 = 100  
  
x3 = 50  
y3 = -50  
  
x4 = -150  
y4 = -150  
  
  
run = True  
**while** run:  
 **for** event **in** pygame. event.get():  
 **if** event.type == pygame.QUIT:  
 run = False  
   
 screen.fill((255,255,255))  
   
 pygame.draw.circle(screen, (0, 0, 255), (x1, y1), 75)  
   
 pygame.draw.circle(screen, (0, 255, 255), (x2, y2), 25)  
 pygame.draw.circle(screen, (10, 40, 255), (x3, y3), 60)  
 pygame.draw.circle(screen, (255, 0, 255), (x4, y4), 10)  
   
 pygame. display.flip()  
   
 *# changing the x and y coordinate of the circle*  
 x1 = x1+10  
 y1 = y1+20  
   
 x2 = x2+2  
 y2 = y2-2  
   
 x3 = x3-5  
 y3 = y3+5  
   
 x4 = x4+10  
 y4 = y4+10  
   
 clock.tick(30)  
   
   
pygame.quit()

## TIC TAC TOE GAME Using PyGame



Pygame can be best experienced when we create 2-dimensional games, one of which is the creation of the Tic-Tac-Toe game.

It involves the following steps:

**Step 1:** Importing required modules and initializing the Pygame

**Step 2:** Defining Constants and Colors

**Step 3:** Setting up the display

**Step 4:** Setting up the game state

**Step 5:** Defining the check\_win() function

**Step 6:** Defining the draw\_lines(), draw\_x(), and draw\_o() functions

**Step 7:** The main game loop

**Step 8:** Exit procedures

## Step 1: Importing required modules and initializing the Pygame

* In this step, we import the Pygame module and the sys module.
* The Pygame module is required to create games using Python, while the sys module is required to quit the game when the user closes the game window.
* We then initialize Pygame using the pygame.init() function.

import pygame  
import sys  
  
*# Initialize Pygame*  
pygame. init()

(5, 0)

## Step 2: Defining Constants and Colors

* Here we define the constants and colors used in the game. WINDOW\_SIZE represents the size of the game window, GRID\_SIZE represents the size of each grid cell in the game, and LINE\_WIDTH represents the width of the lines that are drawn to create the game grid.
* We also define three colors WHITE, BLACK, and LINE\_COLOR used in the game.

*# Constants*  
WINDOW\_SIZE = 300  
GRID\_SIZE = 100  
LINE\_WIDTH = 5  
  
*# Colors*  
WHITE = (255, 255, 255)  
BLACK = (0, 0, 0)  
LINE\_COLOR = (0, 0, 0)

## Step 3: Setting up the display

* In this step, we set up the Pygame display. We create a Pygame window using the pygame. display.set\_mode() function and passing the WINDOW\_SIZE tuple (300, 300) as its argument.
* We then set the caption of the window using the **pygame. display.set\_caption()** function.

*# Set up the display*  
screen = pygame.display.set\_mode((WINDOW\_SIZE, WINDOW\_SIZE))  
pygame. display.set\_caption('Tic Tac Toe')

## Step 4: Setting up the game state

* Here we set up the game state. We create a game\_board list of lists to store the game state. Each element of the game\_board represents a grid cell in the game. Initially, all cells are empty and represented by a space character.
* We also create a current\_player variable to keep track of the player whose turn it is. The player is initialized as 'X'.

*# Set up the game state*  
game\_board = [[' ' **for** \_ **in** range(3)] **for** \_ **in** range(3)]  
current\_player = 'X'

## Step 5: Defining the check\_win() function

* This function is used to check if the current player has won the game. It takes two arguments, board and player. the board is the current game board state, and a player is the current player ('X' or 'O').
* The function checks all the rows and columns of the board and both diagonals for a win by the player. If the player has won, the function returns True, otherwise, it returns False.

**def** check\_win(board, player):  
 **for** row **in** board:  
 **if** all(cell == player **for** cell **in** row):  
 **return** True  
 **for** col **in** range(3):  
 **if** all(board[row][col] == player **for** row **in** range(3)):  
 **return** True  
 **if** all(board[i][i] == player **for** i **in** range(3)) **or** all(board[i][2 - i] == player **for** i **in** range(3)):  
 **return** True  
 **return** False

## Step 6: Defining the draw\_lines(), draw\_x(), and draw\_o() functions

* **The draw\_lines()** function draws the grid lines for the game. It does this by iterating over the range from 1 to 3, and for each iteration, it draws a vertical line from the top to the bottom of the screen at the x-coordinate i \* GRID\_SIZE, and a horizontal line from the left to the right of the screen at the y-coordinate i \* GRID\_SIZE. The LINE\_COLOR and LINE\_WIDTH variables are used to set the color and thickness of the lines.
* **The draw\_x(row, col)** function draws an "X" symbol at a specific row and column on the game board. It first calculates an offset variable, which is half the size of a grid square, rounded down to the nearest integer. It then draws two diagonal lines to form the "X" shape, using the coordinates of the top-left and bottom-right corners of the square.
* **The draw\_o(row, col)** function draws an "O" symbol at a specific row and column on the game board. It first calculates the same offset variable as in the draw\_x() function. It then draws a circle centered at the midpoint of the square, using a radius equal to half the size of the square minus the offset value. The BLACK variable is used to set the color of the circle.
* These functions can be used together to draw a complete tic-tac-toe game board on the screen, with X's and O's being drawn in specific locations as the game progresses.

**def** draw\_lines():  
 **for** i **in** range(1, 3):  
 pygame.draw.line(screen, LINE\_COLOR, (i \* GRID\_SIZE, 0), (i \* GRID\_SIZE, WINDOW\_SIZE), LINE\_WIDTH)  
 pygame.draw.line(screen, LINE\_COLOR, (0, i \* GRID\_SIZE), (WINDOW\_SIZE, i \* GRID\_SIZE), LINE\_WIDTH)  
  
**def** draw\_x(row, col):  
 offset = GRID\_SIZE // 4  
 pygame.draw.line(screen, LINE\_COLOR, (col \* GRID\_SIZE + offset, row \* GRID\_SIZE + offset),  
 ((col + 1) \* GRID\_SIZE - offset, (row + 1) \* GRID\_SIZE - offset), LINE\_WIDTH)  
 pygame.draw.line(screen, LINE\_COLOR, ((col + 1) \* GRID\_SIZE - offset, row \* GRID\_SIZE + offset),  
 (col \* GRID\_SIZE + offset, (row + 1) \* GRID\_SIZE - offset), LINE\_WIDTH)  
  
**def** draw\_o(row, col):  
 offset = GRID\_SIZE // 4  
 pygame.draw.circle(screen, BLACK, (col \* GRID\_SIZE + GRID\_SIZE // 2, row \* GRID\_SIZE + GRID\_SIZE // 2),  
 GRID\_SIZE // 2 - offset, LINE\_WIDTH)

## Step 7: The main game loop

* The while loop runs as long as the variable running is True. In each iteration of the loop, the screen is filled with the WHITE color, and the lines for the tic-tac-toe grid are drawn using the **draw\_lines()** function.
* Then, the loop iterates over the events that have occurred since the last time the loop ran using the **pygame. event.get()** method. If the user clicks the "close" button in the window, running is set to False and the loop will exit.
* Otherwise, if the game is not over and the user clicks the mouse button, the row and column of the clicked cell are calculated based on the mouse position. If the clicked cell is empty, the current player's symbol ('X' or 'O') is placed in the cell using the game\_board list.
* The **check\_win()** function is called to check if the current player has won the game. If the player has won, a message is printed to the console, and game\_over is set to True. Otherwise, it is the other player's turn.
* After all the events are processed, **the draw\_x() and draw\_o()** functions are called to draw the symbols of the players on the screen based on the current state of the game\_board list.
* Finally, the **pygame. display.flip()** method updates the display and shows the new state of the screen to the user.

*# Main game loop*  
running = True  
game\_over = False  
  
**while** running:  
 screen.fill(WHITE)  
 draw\_lines()  
  
 **for** events **in** pygame. event.get():  
 **if** event.type == pygame.QUIT:  
 running = False  
  
 **if** **not** game\_over **and** event.type == pygame.MOUSEBUTTONDOWN:  
 x, y = event.pos  
 row, col = y // GRID\_SIZE, x // GRID\_SIZE  
  
 **if** game\_board[row][col] == ' ':  
 game\_board[row][col] = current\_player  
  
 **if** check\_win(game\_board, current\_player):  
 print(f"Player {current\_player} wins!")  
 game\_over = True  
 **else**:  
 current\_player = 'O' **if** current\_player == 'X' **else** 'X'  
 *# Draw symbols*  
 **for** row **in** range(3):  
 **for** col **in** range(3):  
 **if** game\_board[row][col] == 'X':  
 draw\_x(row, col)  
 **elif** game\_board[row][col] == 'O':  
 draw\_o(row, col)  
  
 pygame. display.flip()

## Step 8: Exit procedures

* The **pygame. quit()** function is used to uninitialize all Pygame modules previously initialized with 'pygame. init()'. This function should be called once all Pygame functionality is no longer needed in the program.
* The **sys. exit()** function is used to exit the program and is part of the Python standard library. It causes the Python interpreter to exit with a status code of zero, indicating that the program has been completed.
* Together, these two lines of code are often used at the end of a Pygame program to properly shut down all Pygame modules and exit the program.

pygame.quit()  
sys.exit()

## Final Complete Program

*# Tic-Tac-Toe Game using Python*  
import pygame  
import sys  
  
*# Initialize Pygame*  
pygame.init()  
  
*# Constants*  
WINDOW\_SIZE = 300  
GRID\_SIZE = 100  
LINE\_WIDTH = 5  
  
*# Colors*  
WHITE = (255, 255, 255)  
BLACK = (0, 0, 0)  
LINE\_COLOR = (0, 0, 0)  
  
*# Set up the display*  
screen = pygame.display.set\_mode((WINDOW\_SIZE, WINDOW\_SIZE))  
pygame.display.set\_caption('Tic Tac Toe')  
  
*# Set up the game state*  
game\_board = [[' ' **for** \_ **in** range(3)] **for** \_ **in** range(3)]  
current\_player = 'X'  
  
**def** check\_win(board, player):  
 **for** row **in** board:  
 **if** all(cell == player **for** cell **in** row):  
 **return** True  
 **for** col **in** range(3):  
 **if** all(board[row][col] == player **for** row **in** range(3)):  
 **return** True  
 **if** all(board[i][i] == player **for** i **in** range(3)) **or** all(board[i][2 - i] == player **for** i **in** range(3)):  
 **return** True  
 **return** False  
  
**def** draw\_lines():  
 **for** i **in** range(1, 3):  
 pygame.draw.line(screen, LINE\_COLOR, (i \* GRID\_SIZE, 0), (i \* GRID\_SIZE, WINDOW\_SIZE), LINE\_WIDTH)  
 pygame.draw.line(screen, LINE\_COLOR, (0, i \* GRID\_SIZE), (WINDOW\_SIZE, i \* GRID\_SIZE), LINE\_WIDTH)  
  
**def** draw\_x(row, col):  
 offset = GRID\_SIZE // 4  
 pygame. draw.line(screen, LINE\_COLOR, (col \* GRID\_SIZE + offset, row \* GRID\_SIZE + offset),  
 ((col + 1) \* GRID\_SIZE - offset, (row + 1) \* GRID\_SIZE - offset), LINE\_WIDTH)  
 pygame.draw.line(screen, LINE\_COLOR, ((col + 1) \* GRID\_SIZE - offset, row \* GRID\_SIZE + offset),  
 (col \* GRID\_SIZE + offset, (row + 1) \* GRID\_SIZE - offset), LINE\_WIDTH)  
  
**def** draw\_o(row, col):  
 offset = GRID\_SIZE // 4  
 pygame.draw.circle(screen, BLACK, (col \* GRID\_SIZE + GRID\_SIZE // 2, row \* GRID\_SIZE + GRID\_SIZE // 2),  
 GRID\_SIZE // 2 - offset, LINE\_WIDTH)  
  
  
*# Main game loop*  
running = True  
game\_over = False  
  
**while** running:  
 screen.fill(WHITE)  
 draw\_lines()  
  
 **for** event **in** pygame. event.get():  
 **if** event.type == pygame.QUIT:  
 running = False  
  
 **if** **not** game\_over **and** event.type == pygame.MOUSEBUTTONDOWN:  
 x, y = event.pos  
 row, col = y // GRID\_SIZE, x // GRID\_SIZE  
  
 **if** game\_board[row][col] == ' ':  
 game\_board[row][col] = current\_player  
  
 **if** check\_win(game\_board, current\_player):  
 print(f"Player {current\_player} wins!")  
 game\_over = True  
 **else**:  
 current\_player = 'O' **if** current\_player == 'X' **else** 'X'  
  
 *# Draw symbols*  
 **for** row **in** range(3):  
 **for** col **in** range(3):  
 **if** game\_board[row][col] == 'X':  
 draw\_x(row, col)  
 **elif** game\_board[row][col] == 'O':  
 draw\_o(row, col)  
  
 pygame. display.flip()  
  
pygame.quit()  
sys.exit()

**REVISION :**

* Understanding: pygame, the difference between pygame
* Installation of pygame
* Understanding: bare code for the game window
* Event handling: quit the event
* screen color filling
* Updating screen color
* drawing shapes
* The Tic Tac Toe game

**HOMEWORK**

1. Try to create a Chess board using Pygame
2. Draw 2 different shapes together on the screen with different sizes, colors, and positions

Task 1 Solution- To create a chessboard

import pygame

import sys

# Initialize Pygame

pygame. init()

# Constants

WIDTH, HEIGHT = 480, 480

SQUARE\_SIZE = WIDTH // 8

# Colors

WHITE = (255, 255, 255)

BLACK = (0, 0, 0)

# Create the Pygame screen

screen = pygame. display.set\_mode((WIDTH, HEIGHT))

pygame. display.set\_caption("Chessboard")

# Main game loop

while True:

for event in pygame. event.get():

if event.type == pygame.QUIT:

pygame.quit()

sys.exit()

# Draw the chessboard

for row in range(8):

for col in range(8):

color = WHITE if (row + col) % 2 == 0 else BLACK

pygame. draw.rect(screen, color, (col \* SQUARE\_SIZE, row \* SQUARE\_SIZE, SQUARE\_SIZE, SQUARE\_SIZE))

# Update the display

pygame. display.flip()

# Exit the program

pygame.quit()