Last Updated : 05 Dec, 2022

This article will guide you and give you a basic idea of designing a game Tic Tac Toe using *pygame* library of Python. Pygame is a cross-platform set of Python modules designed for writing video games. It includes computer graphics and sound libraries designed to be used with the Python programming language. Let's break the task in five parts:

1. Importing the required libraries and setting up the required global variables.
2. Designing the game display function, that will set a platform for other components to be displayed on the screen.
3. Main algorithm of win and draw
4. Getting the user input and displaying the "X" or "O" at the proper position where the user has clicked his mouse.
5. Running an infinite loop, and including the defined methods in it.

Note: The required PNG files can be downloaded below as follows:

modified_cover.png

X_modified.png

o_modified.png

Importing the required libraries and setting up the required global variables

We are going to use the pygame, time, and the sys library of Python. **time** library is used to keep track of time and sleep() method that we are going to use inside our code. Have a look at the code below.

Python3

```
# importing the required libraries
import pygame as pg
import sys
import time
from pygame.locals import *

# declaring the global variables

# for storing the 'x' or 'o'
# value as character
XO = 'x'

# storing the winner's value at
# any instant of code
winner = None

# to check if the game is a draw
draw = None

# to set width of the game window
width = 400

# to set height of the game window
height = 400

# to set background color of the
# game window
white = (255, 255, 255)

# color of the straightlines on that
# white game board, dividing board
# into 9 parts
line_color = (0, 0, 0)
```

```
# setting up a 3 * 3 board in canvas
board = [[None]*3, [None]*3, [None]*3]
```

Designing the game display

This is the trickier part, that makes the utmost importance in game development. We can use the **display.set_mode()** method to set up our display window. This takes three arguments, first one being a tuple having (width, height) of the display that we want it to be, the other two arguments are depth and fps respectively.**display.set_caption()**, sets a caption on the name tag of our display. **pg.image.load()** is an useful method to load the background images to customize the display. This method takes the file name as an argument along with the extension. There is a small problem with image.load(), it loads the image as a Python object in its native size, which may not be optimized along with the display. So we use another method in pygame known as **pg.transform.scale()**. This method takes two arguments, one being the name of the image object and the other is a tuple having (width, height), that we want our image to scale to. Finally we head to the first function, **game_initiating_window()**. On the very first line there is a **screen.blit()** function. The screen is the Python function and blit is the method that enables pygame to display something over another thing. Here out image object has been displayed over the screen, which was set white initially. **pg.display.update()** is another important function in game development. It updates the display of our window when called. Pygame also enables us to draw geometric objects like line, circle, etc. In this project we have used **pg.draw.line()** method that takes five arguments, namely – *(display, line color, starting point, ending point, width)*. This involves a little bit of coordinate geometry to draw the lines properly. This is not sufficient. At each update of the display we need to know the game status, Whether it is win or lose.**draw_status()** helps us in displaying another 100pc window at the bottom of the main window, that updates the status at each click of the user.

Python3

```
# initializing the pygame window
pg.init()

# setting fps manually
fps = 30

# this is used to track time
CLOCK = pg.time.Clock()

# this method is used to build the
# infrastructure of the display
screen = pg.display.set_mode((width, height + 100), 0, 32)

# setting up a nametag for the
# game window
pg.display.set_caption("My Tic Tac Toe")

# loading the images as python object
initiating_window = pg.image.load("modified_cover.png")
x_img = pg.image.load("X_modified.png")
y_img = pg.image.load("o_modified.png")

# resizing images
initiating_window = pg.transform.scale(
    initiating_window, (width, height + 100))
x_img = pg.transform.scale(x_img, (80, 80))
o_img = pg.transform.scale(y_img, (80, 80))


def game_initiating_window():

    # displaying over the screen
    screen.blit(initiating_window, (0, 0))

    # updating the display
    pg.display.update()
    time.sleep(3)
    screen.fill(white)

    # drawing vertical lines
    pg.draw.line(screen, line_color, (width / 3, 0), (width / 3, height), 7)
    pg.draw.line(screen, line_color, (width / 3 * 2, 0),
                 (width / 3 * 2, height), 7)

    # drawing horizontal lines
    pg.draw.line(screen, line_color, (0, height / 3), (width, height / 3), 7)
    pg.draw.line(screen, line_color, (0, height / 3 * 2),
                 (width, height / 3 * 2), 7)
```

```
    draw_status()


def draw_status():

    # getting the global variable draw
    # into action
    global draw

    if winner is None:
        message = XO.upper() + "'s Turn"
    else:
        message = winner.upper() + " won !"
    if draw:
        message = "Game Draw !"

    # setting a font object
    font = pg.font.Font(None, 30)

    # setting the font properties like
    # color and width of the text
    text = font.render(message, 1, (255, 255, 255))

    # copy the rendered message onto the board
    # creating a small block at the bottom of the main display
    screen.fill((0, 0, 0), (0, 400, 500, 100))
    text_rect = text.get_rect(center=(width / 2, 500-50))
    screen.blit(text, text_rect)
    pg.display.update()
```

Main algorithm

The main algorithm has a straight forward approach. A user can win row-wise, column-wise, and diagonally. So by using a multidimensional array, we can set up the conditions easily.

Python3

```
def check_win():
    global board, winner, draw

    # checking for winning rows
    for row in range(0, 3):
        if((board[row][0] == board[row][1] == board[row][2]) and (board[row][0] i
```

```
            winner = board[row][0]
            pg.draw.line(screen, (250, 0, 0),
                         (0, (row + 1)*height / 3 - height / 6),
                         (width, (row + 1)*height / 3 - height / 6),
                         4)
            break

    # checking for winning columns
    for col in range(0, 3):
        if((board[0][col] == board[1][col] == board[2][col]) and (board[0][col] i
            winner = board[0][col]
            pg.draw.line(screen, (250, 0, 0), ((col + 1) * width / 3 - width / 6,
                         ((col + 1) * width / 3 - width / 6, height), 4)
            break

    # check for diagonal winners
    if (board[0][0] == board[1][1] == board[2][2]) and (board[0][0] is not None):

        # game won diagonally left to right
        winner = board[0][0]
        pg.draw.line(screen, (250, 70, 70), (50, 50), (350, 350), 4)

    if (board[0][2] == board[1][1] == board[2][0]) and (board[0][2] is not None):

        # game won diagonally right to left
        winner = board[0][2]
        pg.draw.line(screen, (250, 70, 70), (350, 50), (50, 350), 4)

    if(all([all(row) for row in board]) and winner is None):
        draw = True

    draw_status()
```

Getting the user input and displaying the “X” or “O”

This part deals with a visualization of the board and a little bit of coordinate geometry. **drawXO()** takes two arguments row and col. First of all, we have to set up the correct geometrical position to put the image of X and image of O that we have stored as two python objects “x_img” and “y_img” respectively. Have a look at the code for a proper understanding. **user_click()** is a function we have designed to get the input from a user mouse click. Imagine, you have clicked on one of the nine parts (boxes divided by the lines we have drawn

horizontally and vertically), this function will define the coordinate of the position where you have clicked.**pg.mouse.get_pos()** gets the x-coordinate and y-coordinate of the mouse click of the user and return a tuple. Depending upon the (x, y) we can define the exact row and the exact column where the user has clicked. Finally, when we have the row and col, we pass these two as arguments to the function **drawXO(row, col)** to draw the image of 'X' or the image of 'O' at the desired position of the user on the game screen.

Python3

```
def drawXO(row, col):
    global board, XO

    # for the first row, the image
    # should be pasted at a x coordinate
    # of 30 from the left margin
    if row == 1:
        posx = 30

    # for the second row, the image
    # should be pasted at a x coordinate
    # of 30 from the game line
    if row == 2:

        # margin or width / 3 + 30 from
        # the left margin of the window
        posx = width / 3 + 30

    if row == 3:
        posx = width / 3 * 2 + 30

    if col == 1:
        posy = 30

    if col == 2:
        posy = height / 3 + 30

    if col == 3:
        posy = height / 3 * 2 + 30

    # setting up the required board
    # value to display
    board[row-1][col-1] = XO

    if(XO == 'x'):
```

```
            # pasting x_img over the screen
            # at a coordinate position of
            # (pos_y, posx) defined in the
            # above code
            screen.blit(x_img, (posy, posx))
            XO = 'o'

        else:
            screen.blit(o_img, (posy, posx))
            XO = 'x'
        pg.display.update()


def user_click():
    # get coordinates of mouse click
    x, y = pg.mouse.get_pos()

    # get column of mouse click (1-3)
    if(x < width / 3):
        col = 1

    elif (x < width / 3 * 2):
        col = 2

    elif(x < width):
        col = 3

    else:
        col = None

    # get row of mouse click (1-3)
    if(y < height / 3):
        row = 1

    elif (y < height / 3 * 2):
        row = 2

    elif(y < height):
        row = 3

    else:
        row = None

    # after getting the row and col,
    # we need to draw the images at
    # the desired positions
    if(row and col and board[row-1][col-1] is None):
        global XO
```

```
        drawXO(row, col)
        check_win()
```

Running an infinite loop

This is the final important step to run our game infinitely until the user clicks **exit**. Before running an infinite loop, we need to set up a function that can reset all the global values and parameters to initial values for a fresh start of the game. **reset_game()** is used for this purpose. It resets the board value to 3 * 3 None value again and initializes global parameters. In the game development, every action by the player is an **event**. Whether he clicks on the window or clicks on the exit/close icon. To get these events as an object, pygame has a built-in method used as **pg.event.get()**. If the event type is “QUIT”, we use the sys library of Python to exit the game. But if the mouse is pressed, the **event.get()** will return “MOUSEBUTTONDOWN” and our call to **user_click()** happens to know the exact coordinate of the board where the user has clicked. In the entire code, we have used the .sleep() method to pause our game for some time and make that user-friendly and smooth.

Python3

```
def reset_game():
    global board, winner, XO, draw
    time.sleep(3)
    XO = 'x'
    draw = False
    game_initiating_window()
    winner = None
    board = [[None]*3, [None]*3, [None]*3]


game_initiating_window()

while(True):
    for event in pg.event.get():

        if event.type == QUIT:
            pg.quit()
            sys.exit()
```

```
        elif event.type is MOUSEBUTTONDOWN:
            user_click()

            if(winner or draw):
                reset_game()

    pg.display.update()
    CLOCK.tick(fps)
```

The complete code:

Python3

```
# importing the required libraries
import pygame as pg
import sys
import time
from pygame.locals import *

# declaring the global variables

# for storing the 'x' or 'o'
# value as character
XO = 'x'

# storing the winner's value at
# any instant of code
winner = None

# to check if the game is a draw
draw = None

# to set width of the game window
width = 400

# to set height of the game window
height = 400

# to set background color of the
# game window
white = (255, 255, 255)

# color of the straightlines on that
# white game board, dividing board
# into 9 parts
```

```
line_color = (0, 0, 0)

# setting up a 3 * 3 board in canvas
board = [[None]*3, [None]*3, [None]*3]


# initializing the pygame window
pg.init()

# setting fps manually
fps = 30

# this is used to track time
CLOCK = pg.time.Clock()

# this method is used to build the
# infrastructure of the display
screen = pg.display.set_mode((width, height + 100), 0, 32)

# setting up a nametag for the
# game window
pg.display.set_caption("My Tic Tac Toe")

# loading the images as python object
initiating_window = pg.image.load("modified_cover.png")
x_img = pg.image.load("X_modified.png")
y_img = pg.image.load("o_modified.png")

# resizing images
initiating_window = pg.transform.scale(
    initiating_window, (width, height + 100))
x_img = pg.transform.scale(x_img, (80, 80))
o_img = pg.transform.scale(y_img, (80, 80))


def game_initiating_window():

    # displaying over the screen
    screen.blit(initiating_window, (0, 0))

    # updating the display
    pg.display.update()
    time.sleep(3)
    screen.fill(white)

    # drawing vertical lines
    pg.draw.line(screen, line_color, (width / 3, 0), (width / 3, height), 7)
    pg.draw.line(screen, line_color, (width / 3 * 2, 0),
                 (width / 3 * 2, height), 7)
```

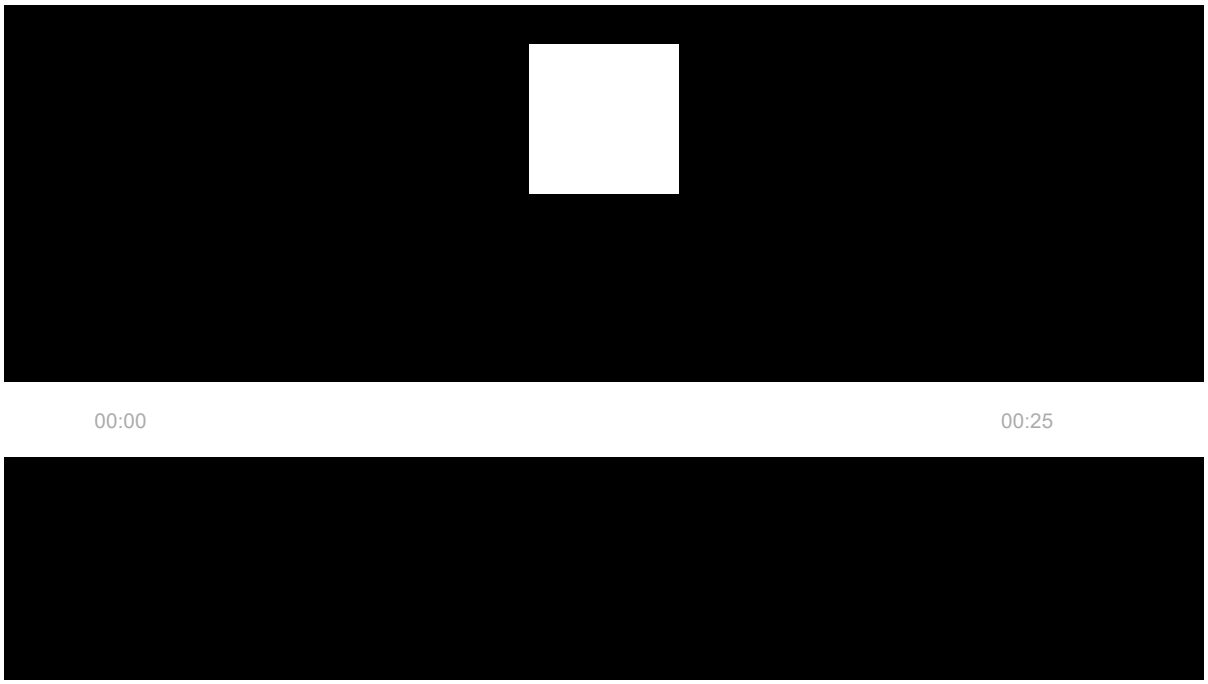
```
    # drawing horizontal lines
    pg.draw.line(screen, line_color, (0, height / 3), (width, height / 3), 7)
    pg.draw.line(screen, line_color, (0, height / 3 * 2),
                 (width, height / 3 * 2), 7)
    draw_status()


def draw_status():

    # getting the global variable draw
    # into action
    global draw

    if winner is None:
        message = XO.upper() + "'s Turn"
    else:
        message = winner.upper() + " won !"
    if draw:
        message = "Game Draw !"

    # setting a font object
    font = pg.font.Font(None, 30)

    # setting the font properties like
    # color and width of the text
    text = font.render(message, 1, (255, 255, 255))

    # copy the rendered message onto the board
    # creating a small block at the bottom of the main display
    screen.fill((0, 0, 0), (0, 400, 500, 100))
    text_rect = text.get_rect(center=(width / 2, 500-50))
    screen.blit(text, text_rect)
    pg.display.update()


def check_win():
    global board, winner, draw

    # checking for winning rows
    for row in range(0, 3):
        if((board[row][0] == board[row][1] == board[row][2]) and (board[row][0] i
            winner = board[row][0]
            pg.draw.line(screen, (250, 0, 0),
                         (0, (row + 1)*height / 3 - height / 6),
                         (width, (row + 1)*height / 3 - height / 6),
                         4)
            break
```

```
    # checking for winning columns
    for col in range(0, 3):
        if((board[0][col] == board[1][col] == board[2][col]) and (board[0][col] i
            winner = board[0][col]
            pg.draw.line(screen, (250, 0, 0), ((col + 1) * width / 3 - width / 6
                         ((col + 1) * width / 3 - width / 6, height), 4)
            break

    # check for diagonal winners
    if (board[0][0] == board[1][1] == board[2][2]) and (board[0][0] is not None)

        # game won diagonally left to right
        winner = board[0][0]
        pg.draw.line(screen, (250, 70, 70), (50, 50), (350, 350), 4)

    if (board[0][2] == board[1][1] == board[2][0]) and (board[0][2] is not None)

        # game won diagonally right to left
        winner = board[0][2]
        pg.draw.line(screen, (250, 70, 70), (350, 50), (50, 350), 4)

    if(all([all(row) for row in board]) and winner is None):
        draw = True
    draw_status()


def drawXO(row, col):
    global board, XO

    # for the first row, the image
    # should be pasted at a x coordinate
    # of 30 from the left margin
    if row == 1:
        posx = 30

    # for the second row, the image
    # should be pasted at a x coordinate
    # of 30 from the game line
    if row == 2:

        # margin or width / 3 + 30 from
        # the left margin of the window
        posx = width / 3 + 30

    if row == 3:
        posx = width / 3 * 2 + 30

    if col == 1:
        posy = 30
```

```
        if col == 2:
            posy = height / 3 + 30

        if col == 3:
            posy = height / 3 * 2 + 30

        # setting up the required board
        # value to display
        board[row-1][col-1] = XO

        if(XO == 'x'):

            # pasting x_img over the screen
            # at a coordinate position of
            # (pos_y, posx) defined in the
            # above code
            screen.blit(x_img, (posy, posx))
            XO = 'o'

        else:
            screen.blit(o_img, (posy, posx))
            XO = 'x'
        pg.display.update()


    def user_click():
        # get coordinates of mouse click
        x, y = pg.mouse.get_pos()

        # get column of mouse click (1-3)
        if(x < width / 3):
            col = 1

        elif (x < width / 3 * 2):
            col = 2

        elif(x < width):
            col = 3

        else:
            col = None

        # get row of mouse click (1-3)
        if(y < height / 3):
            row = 1

        elif (y < height / 3 * 2):
            row = 2
```

```
    elif(y < height):
        row = 3

    else:
        row = None

    # after getting the row and col,
    # we need to draw the images at
    # the desired positions
    if(row and col and board[row-1][col-1] is None):
        global XO
        drawXO(row, col)
        check_win()


def reset_game():
    global board, winner, XO, draw
    time.sleep(3)
    XO = 'x'
    draw = False
    game_initiating_window()
    winner = None
    board = [[None]*3, [None]*3, [None]*3]


game_initiating_window()

while(True):
    for event in pg.event.get():
        if event.type == QUIT:
            pg.quit()
            sys.exit()
        elif event.type is MOUSEBUTTONDOWN:
            user_click()
            if(winner or draw):
                reset_game()
    pg.display.update()
    CLOCK.tick(fps)
```

Output:



3. Next, it stores the winner's value at any instant of code.
4. To check if the game is a draw, there is a variable called draw that will be set to None when we are done with this function and then used in other functions later on in this program.
5. The next step is to set up some variables for width of the game window and height of the game window which will be 400 pixels wide and 400 pixels tall respectively.
6. It also sets background color of the game window to white (255, 255, 255).
7. The color of straightlines on that white board dividing into 9 parts is line_color = (0, 0, 0) while setting up a 3 * 3 board in canvas creates an empty array where each element has three spaces followed by two more elements which create nine squares on our tic tac toe board.
8. Lastly it initializes pygame window with pg.init() method before loading images as python object using pg image library methods such as load("modified_cover.png") , x_img = pg .image .load("X_modified."), y_img = pg
9. The code is a snippet of code that displays the game board in canvas.
10. The code declares global variables such as winner, draw, XO and line_color.
11. The code sets up the game window with width and height values.
12. The background color is white which is set to be 255, 255, 255.
13. The code starts by checking for the winner of the game.

14. If there is no winner, then it prints “XO’s Turn” and sets a font object to print text on screen.
15. The code checks for winning rows and columns.
16. It also checks if there are diagonal winners in order to determine who won diagonally left or right.
17. The next part of the code starts with a function called check_win().
18. This function loops through all four possible combinations of winning rows and columns, which will be used later when determining who won diagonally left or right.
19. After this loop finishes, it determines whether XO has won by checking if board[0][0] == board[1][1] == board[2][2].
20. If so, then XO wins because they have three in a row horizontally (left-to-right) and two vertically (top-to-bottom).
21. The code is a basic game of tic-tac-toe.
22. The player who gets three in a row first wins the game.
23. The code begins by declaring variables for the board, winner, and draw.
24. The variable board will be used to store information about the current state of the game such as which rows and columns are won or lost.
25. The variable winner will store information about who has won the game so far.
26. The variable draw will hold whether or not there is a winner yet.
27. After declaring these variables, two functions are created: check_win() and draw_status().
28. These functions are called when it’s time to check if someone has won or when it’s time to update what is displayed on screen during gameplay
29. The code starts by defining a function called drawXO.
30. This function takes in two parameters: the row and the col of where to place an image on the screen.
31. The first parameter is defined as being 30 from the left margin, so for each row, we need to define what position it should be pasted at.
32. For example, if you wanted to paste an image over column 1, then you would set posx = width / 3 + 30.
33. The next line defines that XO will always be ‘o’ when it’s not drawing anything else and ‘x’ when it is drawing something else.

34. Then comes a function called user_click which gets coordinates of mouse clicks and draws images accordingly based on those coordinates.
35. It also checks whether there is a winner or not before doing anything else with them (i.e., checking who won).
36. Lastly, reset_game resets everything back to how they were originally before starting up again with game_initiating_window().
37. The code is a program that plays Tic-Tac-Toe.
38. The code starts by defining the board, which is an array of three rows and three columns with the value None in each cell.
39. The game_initiating_window() function is called to start the game.
40. If you want to play again, you can call reset_game().
41. The while loop will continue until a user clicks on “Quit”.

A Abhiji...

Previous Article

Voice Assistant for Movies using Python

Next Article

8-bit game using pygame

Similar Reads

Tic Tac Toe game with GUI using tkinter in Python

Tic-tac-toe (American English), noughts and crosses (British English), or Xs and Os is a paper-and-pencil game for two players, X and O, who take turns markin...

15+ min read

Python implementation of automatic Tic Tac Toe game using random...

Tic-tac-toe is a very popular game, so let's implement an automatic Tic-tac-toe game using Python. The game is automatically played by the program and henc...

5 min read

Draw a Tic Tac Toe Board using Python-Turtle

The Task Here is to Make a Tic Tac Toe board layout using Turtle Graphics in Python. For that lets first know what is Turtle Graphics. Turtle graphics In...

2 min read

Tic Tac Toe Game using PyQt5 in Python

In this article , we will see how we can create a Tic Tac Toe game using PyQt5. Tic-tac-toe, noughts, and crosses, or Xs and Os is a paper-and-pencil game for...

5 min read

Multiplayer Tic Tac Toe Game using React & Node

This project is a Multiplayer Online Tic Tac Toe game developed using React & NodeJS . It allows multiple users to play the classic Tic Tac Toe game in real tim...

6 min read

How to Build a Tic Tac Toe Game with Both Offline and Online Mode in...

In this article, we are going to make a tic-tac-toe game that has both online and offline modes. So for this project, we are going to use Kotlin and XML. Tic-Tac-T...

15+ min read

Implementation of Tic-Tac-Toe for 2 person game (User vs. User)

For 1-Person game (User vs. CPU), please refer Implementation of Tic-Tac-Toe game Rules of the Game The game is to be played between two people (in this...

9 min read

Adding Collisions Using pygame.Rect.colliderect in Pygame

Prerequisite: Drawing shapes in Pygame, Introduction to pygame In this article, we are going to use pygame.Rect.colliderect for adding collision in a shape usin...

3 min read

How to create walking character using multiple images from sprite sheet...

In this article, we will cover how to create a walking character using multiple images from a sprite sheet using Pygame. Any video or films that are made are...

5 min read

Slide Puzzle using PyGame - Python

Slide Puzzle game is a 2-dimensional game i.e the pieces can only be removed inside the grid and reconfigured by sliding them into an empty spot. The slide...

15 min read

Article Tags : Project Python Python Programs Python-gui +1 More

Practice Tags : python