# Pygame Series Part 2: Programming Caro Chess Game in Python

Duong Bao Vy
19-12-2021

CS 101 classmates have worked hard to accompany STEAM for Vietnam teachers to practice many projects to conquer their dream of becoming programmers in Silicon Valley. No dream is too big if we really try. Coming to part 2 of Pygame Series, students should join STEAM for Vietnam to discover how to implement the project "Checker Game" of Phan Gia Huy (account is GoodStudent), student of CS 101 Summer 2021 course!

You can review part 1 here

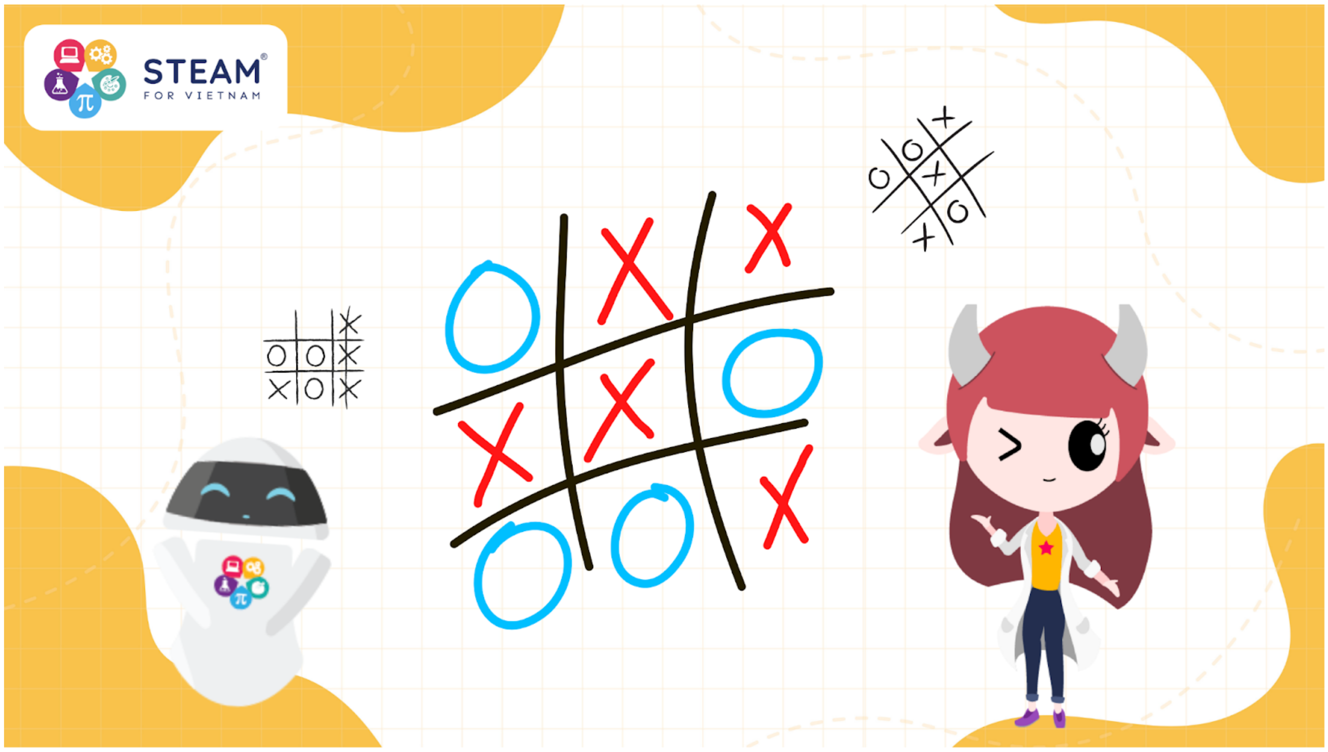
## 1. **Programming knowledge**

- Game programming with pygame library
- Conditional sentence
- 2-dimensional array in python
- Loop in python
- Functions in python
- Some RGB codes of primary colors

## 2. **Introduce game**

The students must be very familiar with the traditional caro game. The way to play this game is very simple, 2 players represent 2 pieces X and O. Each turn, players take turns hitting their pieces on the board. The game only determines the winner or loser when there are 5 pieces of their pieces lying in a row in a row or a vertical row or a diagonal.

Let's try the game first! https://replit.com/@STEAM4VNOfficial/Caro



3. **Let's start programming**!

a. **Algorithm**

We'll pick a fairly large size table to start with. Here, students should choose the table 33×64 which is the easiest to see and convenient to manipulate.

The table here will be represented as a 2-dimensional array. Every time we click on a cell to mark, we will review the whole table to check the conditions of vertical, horizontal, diagonal lines. Then you will know if you will win. Not only that, you will use the loop to count the cells in the 2-dimensional array, only when 5 cells of the same color are filled in a row in a vertical or horizontal or diagonal row, we will print the winning player result.

If there is a final winner, the game prints out the winner result and ends the game, otherwise, the game continues. If in case 2 players have filled the whole board and still do not identify the game, the machine will announce the result of the draw and the end of the game.

b. **Project implementation steps**

Step 1: Declare the library used to code the game.

Here, in addition to the available standard functions of the python programming language, we will use 2 more libraries: pygame and sys. Pygame library is a python library that makes it easier to code games. The sys library is concerned with controlling programs on a computer. To declare the program, we use the following 2 commands.

```
import pygame
import sys
```

Step 2: Declare some basic elements in the game

First, we'll define some of the colors displayed in the game according to the RGB encoding scheme.

```
BLACK = (0, 0, 0)
WHITE = (255, 255, 255)
GREEN = (0, 255, 0)
RED = (255, 0, 0)
BLUE = (0,0,255)
```

In RGB encoding, each color will be represented by 3 elements corresponding to the intensity of 3 primary colors: red (Red), green (Green), blue (Blue). Combine these 3 colors, we will get a new color.

We will assume that x is the first and initialize the FPS – Frames Per Second (smoothness of the game) is 120

```
XO = 'x'
FPS = 120
```

Next, we'll initialize the table sizes and checkered cells. The **WIDTH** and **HEIGHT** variables indicate the length and width of each cell as 28×28. The **MARGIN** variable indicates that the edge thickness of each cell is 2. **The rownum** and **colnum** variables are the row numbers and column numbers of the caro table, respectively. We have 33 rows and 64 columns.

```
# This sets the WIDTH and HEIGHT of each board location

WIDTH = 28
HEIGHT = 28
```

```
# This sets the distance between each cell
MARGIN = 2
rownum = 33
colnum = 64
```

Students should chart the table as a 2-dimensional array with rownum rows and **colnum** columns. The first grid initialization is a single array. Then, we use a loop with the number of iterations corresponding to the number of rows (**rownum**) to initialize each element of the grid as an array, corresponding to a row in the table. We continue to use the second loop with the number of iterations corresponding to the number of columns (**colnum**). We use 2 loops to create a two-dimensional array, like the caro board or maze in lesson 6 of CS 101. The value in each position in the table is 0. So the students have created a caro board with rownum * colnum size!

```
# Create a 2 dimensional array. A two dimensional
# array is simply a list of lists.
grid = []
for row in range(rownum):
    # Add an empty array that will hold each cell
    # in this row
    grid.append([])
    for column in range(colnum):
        grid[row].append(0)  # Append a cell
```

Now you have to start the game. If you want to run a game, first use **the pygame.init() function.** In addition, for the game to display we must initialize the game window: The **WINDOW_SIZE** contains 2 sizes of the length and width of the game screen here will be **1920** * **990**. At that time, the **screen** variable is the game screen in python. We will initialize it by declaring **screen = pygame.display.set_mode(WINDOW_SIZE).** So we've created the game window with the size we require.

```
# Set the HEIGHT and WIDTH of the screen
WINDOW_SIZE = [1920,990]
screen = pygame.display.set_mode(WINDOW_SIZE)
```

In terms of the visual element, we have the image of X and O on the chessboard. Here, we prepare the image of Army X as X_modified-100×100.png and Army O as o_modified-100×100.png. (Students can get a picture at the link). Then, we will bring those two images into the game with the following commands:

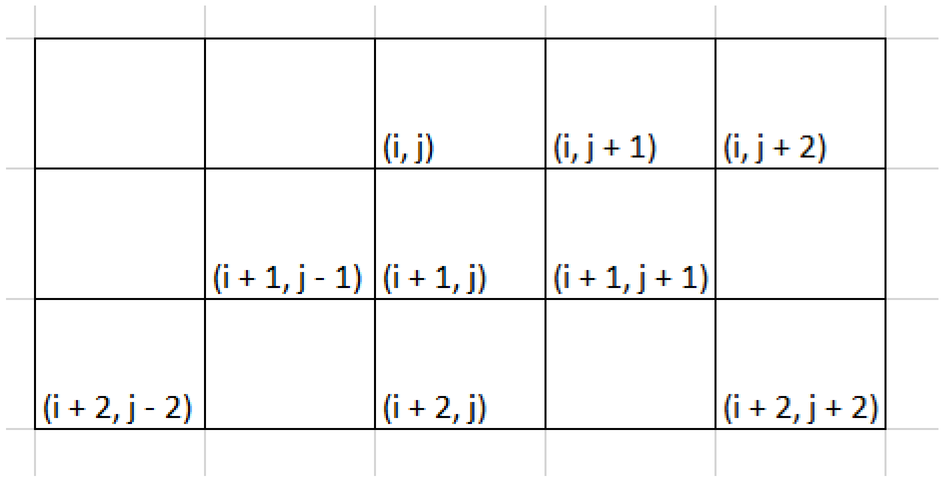
```
x_img = pygame.transform.smoothscale(pygame.image.load("X_modified-100x100.png").convert(),(28,2
o_img = pygame.transform.smoothscale(pygame.image.load("o_modified-100x100.png").convert(),(28,2
```

Here, we can see that **pygame.transform.smoothscale** is the function that helps us convert images into the elements we want in the game. The **pygame.image.load()** function will include the image. The convert**()** function will help us convert images into pixel cells. And finally **(28.28)** is the size of a square in the table I mentioned above.

Step 3: Write the function to check the winning conditions of the game

How do you know the end of the game and the winner?

As you know, the game only has a winner when there exist 5 pieces of the same type located on a horizontal row or a vertical row or a diagonal. So on a board, how would that condition be represented?



| | | | | |
|---|---|---|---|---|
| | | (i, j) | (i, j + 1) | (i, j + 2) |
| | (i + 1, j - 1) | (i + 1, j) | (i + 1, j + 1) | |
| (i + 2, j - 2) | | (i + 2, j) | | (i + 2, j + 2) |

We call cell (i, j) the cell with the address in the ith row and the jth column. Then, assuming cell (i, j) is the cell that starts the horizontal row, the next cells of that horizontal row will be (i, j + 1), (i, j + 2), (i, j + 3), (i, j + 4). If the cell is the one that starts a vertical row, the next cells will be (i + 1,j), (i + 2, j), (i + 3, j), (i + 4, j). We have 2 types of diagonals. For the first type of diagonal, the next cells will be (i + 1, j – 1), (i + 2, j – 2), (i + 3, j – 3), (i + 4, j – 4). Left with the 2nd diagonal type dill is (i + 1, j + 1), (i + 2, j + 2), (i + 3, j + 3), (i + 4, j + 4). We learned how to get an address for a cell in a two-dimensional array in lesson 6 of CS 101.

So to check the winning conditions, we can check the cells in the same row, the same column, the same diagonal more easily. We have the following test function:

```
def checkwin(board):
    indices = [i for i,x in enumerate(board) if 'x' in x]
    for index in indices:
        xrowindices = [i for i, x in enumerate(board[index]) if x == "x"]
        for xs in xrowindices:
            if xs<=len(board[0])-5:
                if board[index][xs] == board[index][xs+1] == board[index][xs+2] == board[index][
                    return 1
            if index<=len(board)-5:
                if board[index][xs] == board[index+1][xs] == board[index+2][xs] == board[index+3
                    return 1
                if xs<=len(board[0])-5:
                    if board[index][xs] == board[index+1][xs+1] == board[index+2][xs+2] == board
                        return 1
                    if board[index][xs] == board[index+1][xs-1] == board[index+2][xs-2] == board
                        return 1
    indices1 = [i for i,x in enumerate(board) if 'o' in x]
    for index1 in indices1:
        orowindices = [i for i, x in enumerate(board[index1]) if x == "o"]
        for os in orowindices:
            if os<=len(board[0])-5:
                if board[index1][os] == board[index1][os+1] == board[index1][os+2] == board[inde
                    return 2
            if index1<=len(board)-5:
                if board[index1][os] == board[index1+1][os] == board[index1+2][os] == board[inde
                    return 2
                if os<=len(board[0])-5:
                    if board[index1][os] == board[index1+1][os+1] == board[index1+2][os+2] == bo
                        return 2
                    if board[index1][os] == board[index1+1][os-1] == board[index1+2][os-2] == bo
                        return 2
    count = 0
    for rows in board:
        for cells in rows:
            if cells == 'x' or cells == 'o':
                count+=1
            if count == rownum*colnum:
```

```
            return 3
    return 0
```

Regarding how the code works, we will separate into 3 test conditions. The first part is to check the winning condition of the X, the 2nd part checks the winning condition of the O and the rest will help us check whether the two players draw or not. We will set if player X wins then return the program result to 1. If player O wins, the result will be returned to 2. If the tournament is drawn, it is worth 3. If the game continues we return a value of 0.

You can see in the above code that there is a very good way to create arrays according to the following conditions:

```
 indices = [i for i,x in enumerate(board) if 'x' in x]
```

The way to create the above array is called **list comprehension** and its structure is as follows:

**newlist = [expression for value in array if true]**

This way of creating an array will help us create a new array from the old array by relying on the old array values according to certain conditions. You may notice that the **enumerate()** function will help us create a list of pairs of indicators – elements in the array such as **type 0 – board [0]** very conveniently. In this code, we will see that the indices array contains the row indices of the board array containing the X children, similar to the indices1 array that will contain the row indices of the board array that exist the O children.

Students, notice two pieces of code that check the winning conditions of X and O. In the second loop, we come across the same array creation. The second array will help us get the column index of the cells in the index row.

In addition to checking rows, columns, diagonals, we first need to make sure the cells we check are not beyond the 2-dimensional array. Otherwise, the program will fail. As when checking a row, we must correctly check **xs <= wool(board[0]) – 5** to see if the cells we check then spill out of the board. Remember to pay attention when checking a satisfactory diagonal, you must see the starting row, column and cell index to be able to check all 5 cells.

As for checking the draw player, let's see if the X or O cars are over. If so, the game is tied. Otherwise the game continues

Step 4: finalize the rest for the game to run

First, we will initialize two variables, **done** and **status**, to show that the game is "over" and "there is a winner". Initially, we left these two variables **False** and **None** because the game has not finished and the outcome of the game has not been determined.

```
# Loop until the user clicks the close button.
done = False
status = None
```

After that, you will use a **while** loop to run the game. If **that's** not true, it means that the game continues and the loop still runs. In pygame, the player's activities will be **events**. We're going to use a loop to get the events that we impact on the game.

In particular, if we end the game, the type of event we perform will be **pygame. QUIT** then pygame will end and we will let **the change** be **True** to end the loop. If we click on a box on the screen, this operation is **pygame. MOUSEBUTTONDOWN** means that the player chooses to mark a piece on the board and we will use the **pygame.mouse.get_pos()** function to get the cell position we click on. If the cell is already marked as X or O, then we ignore it. Otherwise, we will mark it as the xo variable and transform the xo reverse so that it is the second player's turn. Then, we will attach the **status** variable to the value of **the checkwin(grid)** function.

```
while not done:
    for event in pygame.event.get():  # User did something
        if event.type == pygame.QUIT:  # If user clicked close
            done = True  # Flag that we are done so we exit this loop
            # Set the screen background
        if event.type == pygame.MOUSEBUTTONDOWN:
            pos = pygame.mouse.get_pos()
            col = pos[0] // (WIDTH + MARGIN)
            row=  pos[1] // (HEIGHT + MARGIN)
            if grid[row][col] == 0:
                if XO == 'x':
```

```
                grid[row][col] = XO
                XO = 'o'
            else:
                grid[row][col] = XO
                XO = 'x'
        status = checkwin(grid)
```

In the above code, you can see that the row and column index that we took has been reversed. This is because the **python.mouse.get_pos()** function returns the column before the following row.

In the loop above, after ticking the box and checking the winning conditions, we will now draw the caroboard. First, we'll leave the whole palette black. Then, draw the checkered cells as white squares of size **WIDTH*HEIGHT** with a distance of **MARGIN.** So, we're done with the caro square. Now let's use the loop to see if the cell fills X or O, let's insert the X or O image into the cell! We have the following code:

```
for row in range(rownum):
        for column in range(colnum):
            color = WHITE
            pygame.draw.rect(screen,
                             color,
                              [(MARGIN + WIDTH) * column + MARGIN,
                              (MARGIN + HEIGHT) * row + MARGIN,
                              WIDTH,
                              HEIGHT])
            if grid[row][column] == 'x':
                screen.blit(x_img,((WIDTH + MARGIN)*column+2,(HEIGHT + MARGIN)*row+2))
            if grid[row][column] == 'o':
                screen.blit(o_img,((WIDTH + MARGIN)*column+2,(HEIGHT + MARGIN)*row+2))
```

Here, we see the coordinates of a white square corresponding to the row cell, **the** column will be (MARGIN + WIDTH) * column + MARGIN**, (MARGIN + HEIGHT)** * **row**. Please draw a square size **WIDTH** * **HEIGHT.** Pay close attention, you will see that we draw in column order before the back row because pygame will draw a rectangle in size that is horizontal before vertical after. The **screen.blit** function allows us to insert the image we want into the square with the coordinates as above.

Now we have finished drawing the board and if a player wins or draws, we have to print out the result and finish the game.

```
if status == 3:
      font = pygame.font.Font('freesansbold.ttf', 100)
      text = font.render('Draw', True, GREEN, BLUE)
      textRect = text.get_rect()
      textRect.center = (WINDOW_SIZE[0]/2,WINDOW_SIZE[1]/2)
      screen.blit(text,textRect)
      done = True
if status == 1:
      font = pygame.font.Font('freesansbold.ttf', 100)
      text = font.render('X wins', True, GREEN, BLUE)
      textRect = text.get_rect()
      textRect.center = (WINDOW_SIZE[0]/2,WINDOW_SIZE[1]/2)
      screen.blit(text,textRect)
      done = True
if status == 2:
      font = pygame.font.Font('freesansbold.ttf', 100)
      text = font.render('O wins', True, GREEN, BLUE)
      textRect = text.get_rect()
      textRect.center = (WINDOW_SIZE[0]/2,WINDOW_SIZE[1]/2)
      screen.blit(text,textRect)
      done = True
```

Here, you will see the **pygame.font.Font** function that can select the font and font size displayed in turn. Here we will select the freesansbold font and the file of that font is **freesansbold**.ttf, size **100**. If the status is 3 then we have a draw. If the status is 2 we let player O win. As for 1, player X wins.

The **font.render** function will help us denote the text we need. The **True** variable in **the font.render** function helps text appear sharper on the computer. **GREEN** is the color of the letter we choose. **BLUE** is the background color behind it. We've created these colors using the RGB value at the beginning of the program. **The get_rect** function will help get the rectangle containing the words you want. Then, remember to adjust the coordinates of the rectangle in the center to **(**WINDOW_SIZE[**0**]/**2,WINDOW_SIZE[1]/2)**.

In addition, **the screen.blit** function will help the game display the letter we need in that rectangular position. And if the game ends we have to **set the variable** to **True**.

After that, you will make the game update and show up with the following command:

```
clock.tick(FPS)

# Go ahead and update the screen with what we've drawn.
pygame.display.update()
```

**The clock.tick(FPS**) function will make our game smoother and **the python.display.update()** function will help us show what we just did. FPS is the number of frames in a second we have created at the beginning of the program.

At the end of the game, we will stop the game for 10 seconds so that it keeps the player winning and exiting the game.

```
pygame.time.delay(10000)
quit()
pygame.quit()
sys.exit()
```

**The pygame.time.delay(10000**) function will hold the game for about 10 seconds so that we can see the game result and then exit the game with **the pygame.quit() function.**

## 4. **Tadaa !**

After this interesting game, students are updated with interesting knowledge about python programming such as math, functions, conditional sentences, loops …

This is a program that needs knowledge that is quite difficult and long, but equally interesting. Try to create and improve the code by replacing the images of X and

O with whatever you like. Or we can create a bigger board to play.

After completing your personal project, do not forget to share your program on STEAMese Profile for teachers and friends to experience!

— — —

STEAM for Vietnam Foundation is a 501(c)(3) non-profit organization established in the United States with the mission of promoting STEAM (Science, Technology, Engineering, Arts, Mathematics) education in Vietnam. STEAM for Vietnam was founded and operated by a team of volunteers who are international students and Vietnamese experts around the world.

— — —

📧Email: hello@steamforvietnam.org

🌐Website: www.steamforvietnam.org

🌐Fanpage: STEAM for Vietnam

📺YouTube:  http://bit.ly/S4V_YT

🌐Zalo: Zalo Official

Keywords:

READ MORE ARTICLES FROM