TicTacTo Game

The right is my code and the left is the instructor code

1. Move class:

Text

Description automatically generated

Everything is pretty the same except for line 11 in my code, which is not a big deal since I wanted to check whether the move object is an integer or no.

2.Player class

Text

Description automatically generated

Here I have an additional attribute for the player, which I will use further.

Text

Description automatically generated

Here there are some differences. First of all, on line 29 of my code, I have written:

computer\_choice = int(random.choice(range(1, 10)))

putting int behind random function is not required.

Also, I have the move object that is the attribute of the player class, while the professor, didn’t put it as an attribute but directly defined a new instance of the move class.

then, in part 2, there are a bunch of differences. As you can see the instructor code is much more cleaner than mine, cause she has used the is\_valid() method of the move object, while I didn’t use it despite that I have already defined it. The reason is that she first define the move instance and then it checks whether it is valid or not, while, I first get the input from the user and checks the input itself (instead of checking the object).

3.Board Class

A screenshot of a computer

Description automatically generated with medium confidence

Here I have player and computer as attributes while the instructor, defines them again by new separate objects.

Text

Description automatically generated

Here the part 1 are the same with the difference that the instructor has defined the position method again inside the print board method.

Text

Description automatically generated

the other huge difference is in the way we have define the body of the print\_board method. There are some parts that I could improve.

print("|", end="")

I could have put this after the first for loop.

Also, since col is already defined when calling the second for loop, it was not necessary to edit the matrix, instead I could wrote only the column.

Also there is another bad mistake which I could put the print(“ |\n”) after the end of the inward for loop so to go to the next line after the end of the columns.

Couple of notes:

end=""

this means to print the new commands from the for loop in the continue of the previous line. So continue to print the statements horizontally.

print(" |\n")

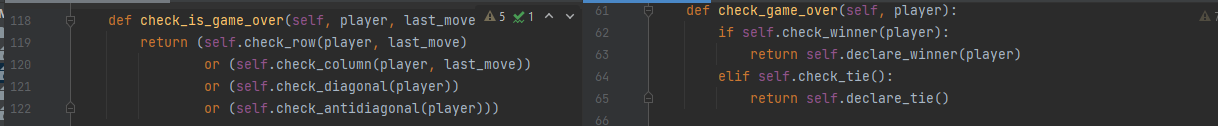
this I will induce two enters instead of one. Instead I could have used this one:

print()

A screenshot of a computer

Description automatically generated with medium confidence

For this method we have basically the same way, except that I have introduced two new print lines.



For this method we are almost different from each other.

Text

Description automatically generated

The instructor method for implementing the algorithm of the game is very interesting. For example in check\_row method she gives as an input two parameters namely player and the last\_move. By the last\_move we find the row of the last move and by that row we extract the whole row in the board which is a list. Then using count method of the list, we count the number of the player markers and if it were equal to 3 we return true.  
the other methods are quite similar.

A screenshot of a computer

Description automatically generated with medium confidence

for checking if we have a tie also, since we have defined the self.game\_board as a list, which itself has three other lists inside, we can say that for each of these three lists, count the number of the empty cells and then if it were zero, return 1.

4. TicTacToeGame Class: The following is the whole main code of the instructor:

class TicTacToeGame:  
  
 def start(self):  
 print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  
 print(" Welcome to Tic-Tac-Toe ")  
 print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  
  
 board = Board()  
 player = Player()  
 computer = Player(False)  
  
 board.print\_board()  
  
 while True:  
  
 while True:  
  
 player\_move = player.get\_move()  
 board.submit\_move(player, player\_move)  
 board.print\_board()  
  
 if board.check\_is\_tie():  
 print("It's a tie! 👍 Try again.")  
 break  
 elif board.check\_is\_game\_over(player, player\_move):  
 print("Awesome. You won the game! 🎉")  
 break  
 else:  
 computer\_move = computer.get\_move()  
 board.submit\_move(computer, computer\_move)  
 board.print\_board()  
  
 if board.check\_is\_game\_over(computer, computer\_move):  
 print("Oops... 😱 The Computer Won. Try again.")  
 break  
  
 play\_again = input("Would you like to play again? Enter X for YES or O for NO: ").upper()  
  
 if play\_again == "O":  
 print("Bye! Come back soon 👋")  
 break  
 elif play\_again == "X":  
 self.start\_new\_round(board)  
 else:  
 print("Your input was not valid but I will assume that you want to play again. 💡")  
 self.start\_new\_round(board)  
  
 def start\_new\_round(self, board):  
 print("\*\*\*\*\*\*\*\*\*\*\*\*\*")  
 print(" New Round ")  
 print("\*\*\*\*\*\*\*\*\*\*\*\*\*")  
 board.reset\_board()  
 board.print\_board()

I didn’t defined a new class but I defined a new method for the board class.

She starts by defining a board instance and two player instances.

it starts with a while loop so that the player can play multiple times the game if he wants. And then, another loop inside it for each round.

Then inside the inner while loop, so for each round she defines a move object for the player and then she prints the board. Then she checks the tie or the game over for the player. If these weren’t the case, now she defines the computers move and check if the computer wins. Then she asks if the user wants to play again or not.

So the take home message is that I am still not sure where and when to use each object. It is very important to use each object where in reality it is needed. For example we have to have a move object whenever we have a new round.

Also as you can see the computer move is defined after checking the winning conditions of the player, cause if the player wins there is no sense to form the computer move.

Or for starting the game we have to have at first the real board, then we have two players for them, and each player generates the move in each round.

The other thing is that the methods are like functions so, you can give any parameter with any name to them regardless of the attributes of the class itself!