**INTRODUCTION TO ICT**

**SEMESTER PROJECT**

TIC TAC TOE GAME IN PYTHON

**TIC TAC TOE GAME**

**ALGORITHM**

* First of all, we create a global variable named as game which is going to control the main loop of our game. We will assign True to this game variable. If we use continue in this loop we can start the game again. To exit from the loop we can use break statement.
* We will take global variables and assign them values.
* The board variable is a 2d array which consists of 3 rows and 3 columns. It is a table of 3 x 3. It is used to store our game data.
* Turn variable is assigned 0 value which will increment with every turn.
* Choice variable is initialized 0. It will store the symbol of the user.
* We print a statement “Welcome to Tic Tac Toe”.
* Now, we call the display\_board function. In display\_board function we have printed a board and its indexes so that user knows about places of the board.
* We take an input from the user to input his name. We store his name is variable user\_name.
* Print “Hi” and user’s name which is stored in user\_name.
* Now, we again make a loop to get symbol of the user and store it in variable user.
* We create an if statement to check if the user input X the computer will have symbol O, else if user input was O then the computer will have symbol O.
* Now, we again make a loop to get heads(press 0) or tails(press 1) and store it in variable user\_choice.
* Toss variable chooses a random number from 0 or 1.
* If the value in toss is same as in user\_choice, then user has won the toss and first turn will be user’s. Else computer has won the toss and will take the first turn.
* We again call the display\_board function to display the board.
* Then we create a loop to break out from when the turn becomes greater than or equal to 9. A game tied message will be displayed.
* If current player is user, it will ask user to enter an index number.
* If current player is computer, the computer will turn first and then turn is incremented by 1 and current player becomes user. In this way we will be taking turns.
* After each turn, a function win\_condition is called to check if user or computer has won the game yet or not.
* When it is computer’s turn it first calls the function can\_cpu\_win. If it returns with true vaue then computer is the winner.
* Else computer checks to block the player move. If user can not block the move of the player, it will prevent it from getting double crossed.
* In this way the computer and user keeps on taking turns unless one of them wins or game is tied.
* The functions used in this python program are as follows:

**display\_board**

It displays the board of the game. It may be called again and again to display the status of the game.

**win\_condition**

It is a function to check if the current player has any win combinations. It checks each row and each column and then both the diagonals to determine if there is any winner and stop the game.

**can\_cpu\_win**

This function determines if there is any single move which computer can play to win the game in one turn. If there is no such move then this function will return false which will take the console to the next function which is brain\_of \_the\_cpu.

**brain\_of \_the\_cpu**

This function calls another function in it which is block\_player\_move. If this functions returns false value then it prevents from getting double crossed.

**block\_player\_move**

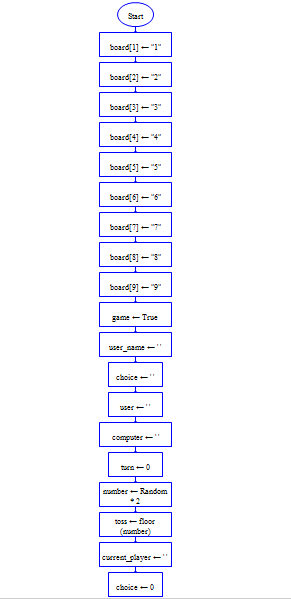
This function blocks the player and does not let it win. It checks each row and column to see if user can win and the block it.

**player\_turn**

This function sees if the input user entered is empty. If it is not then it asks for another input.

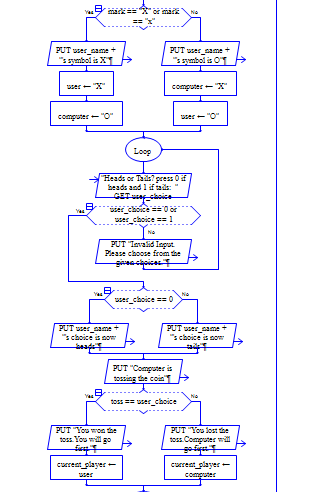
**RAPTOR FLOWCHART**

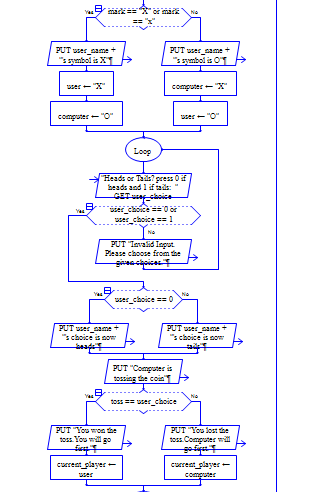
We have used subcharts in our raptor. These are all the subcharts used in our raptor.

Main

Diagram

Description automatically generated





Diagram, schematic

Description automatically generated

Diagram

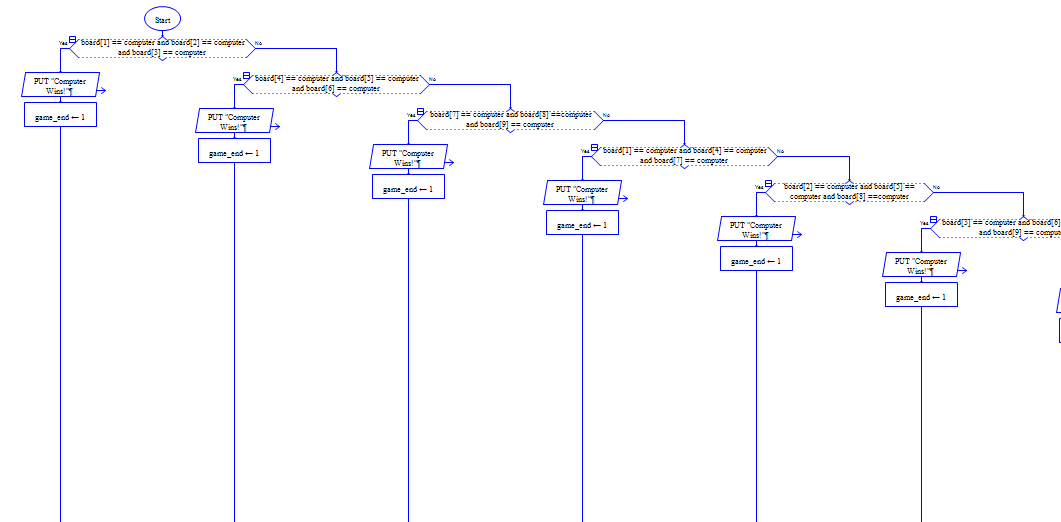
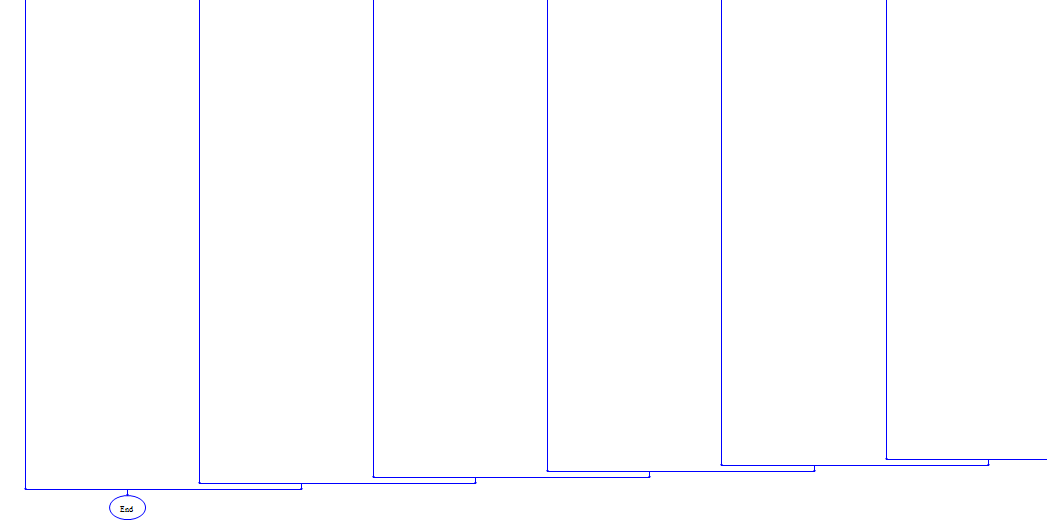
Description automatically generated

Board

Diagram

Description automatically generated

Win\_Condition



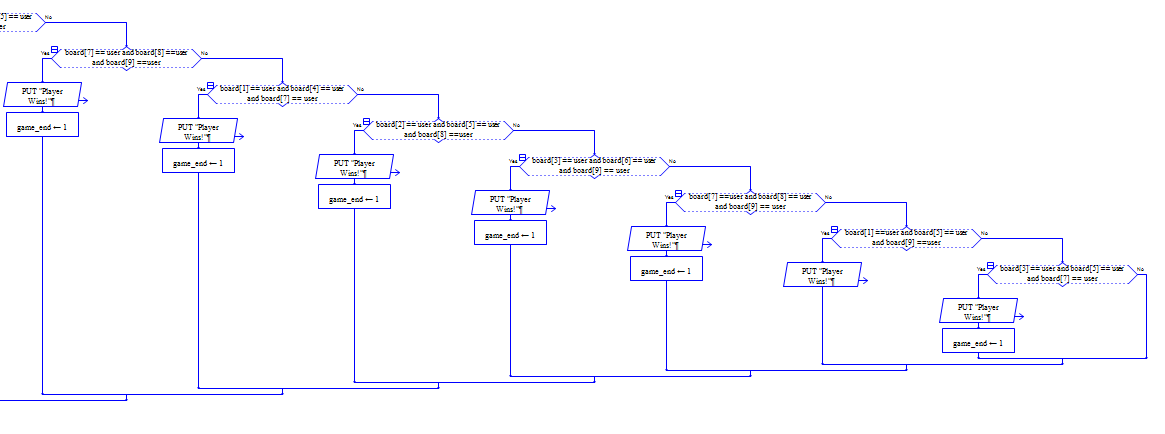
Chart

Description automatically generatedDiagram

Description automatically generated

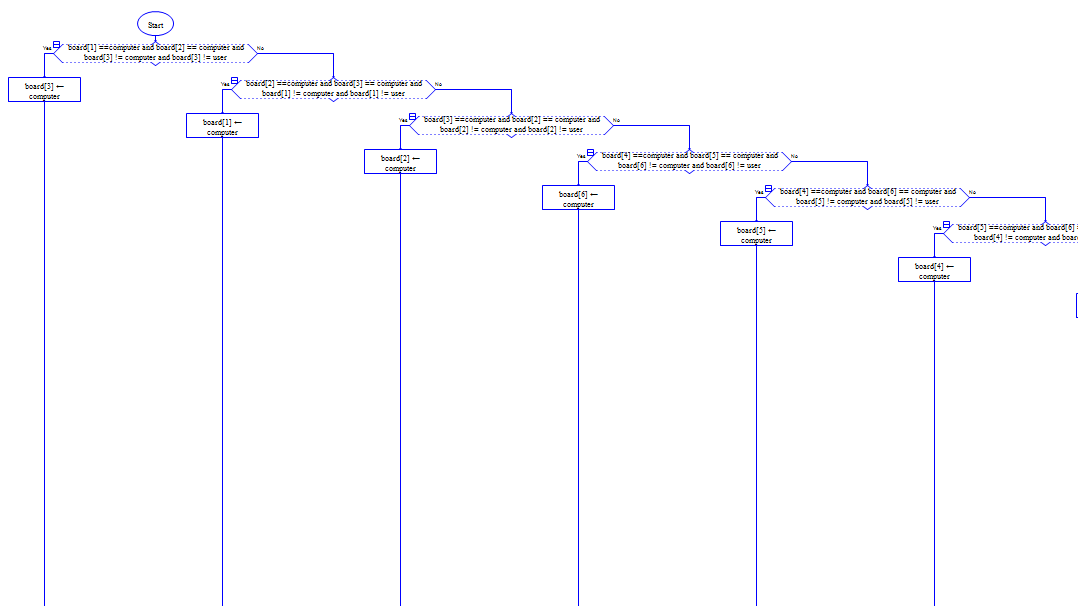
Chart

Description automatically generated



Can\_CPU\_Win

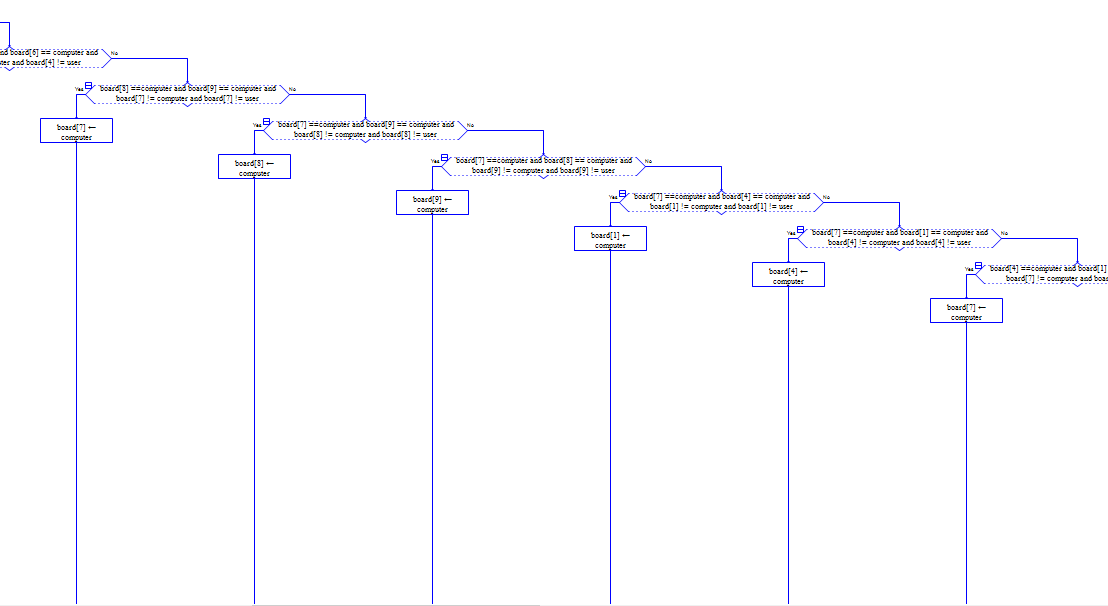
Chart

Description automatically generated

Chart

Description automatically generated

Chart

Description automatically generated

Chart

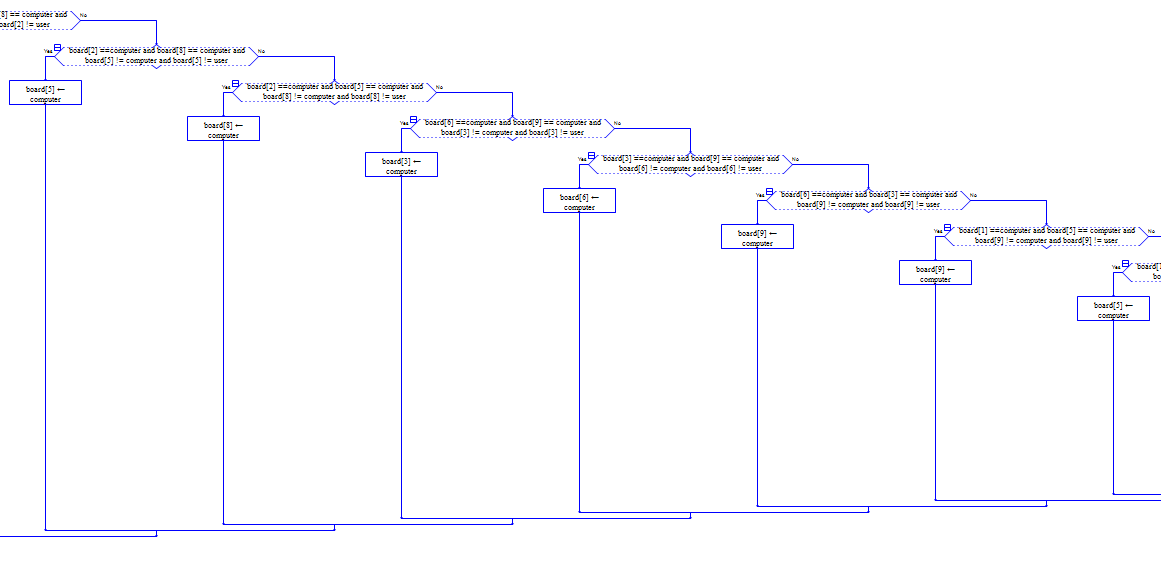
Description automatically generated

Chart

Description automatically generated

Graphical user interface, application

Description automatically generated



Diagram

Description automatically generated

Diagram, schematic

Description automatically generatedBrain\_of\_CPU

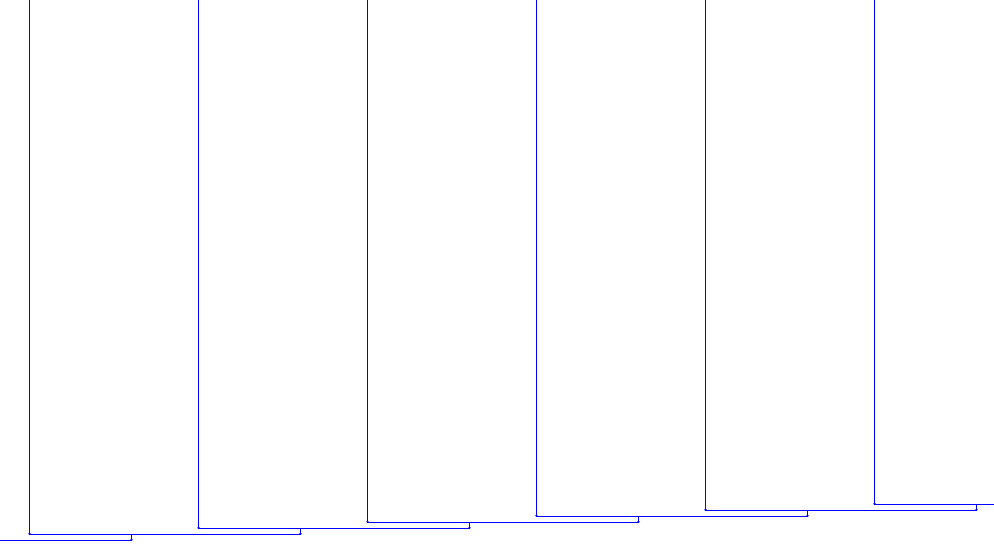
Chart

Description automatically generatedBlock\_Player\_Win

Chart

Description automatically generatedDiagram, schematic

Description automatically generated



Chart

Description automatically generated

Diagram, schematic

Description automatically generated

Chart

Description automatically generatedDiagram, schematic

Description automatically generated

Shape

Description automatically generated with medium confidenceGraphical user interface, diagram, schematic

Description automatically generated

Graphical user interface, diagram

Description automatically generated with medium confidence

**PYTHON CODE**

from random import randint  
  
# ---------global variables-------  
board = [[' ', ' ', ' '], [' ', ' ', ' '], [' ', ' ', ' ']]  
turn = 0  
toss = 0  
choice = 0  
  
  
# show board to the user  
def display\_board():  
 # also printing index beside it for user ease  
 print("-----------------------------------------")  
 print(board[0][0], "|", board[0][1], "|", board[0][2] + " 1 | 2 | 3 ")  
 print("---------- \t\t-------------")  
 print(board[1][0], "|", board[1][1], "|", board[1][2] + " 4 | 5 | 6 ")  
 print("---------- \t\t-------------")  
 print(board[2][0], "|", board[2][1], "|", board[2][2] + " 7 | 8 | 9 ")  
 print("-----------------------------------------")  
  
  
# for inputing player choice in board  
# arguments choice is user input  
# player and computer  
def player\_turn(choice, player, computer):  
 # if user choose first row index then we will subtract -1 from column index  
 # also we will ensure that space is occupied by user or computer  
 if (choice >= 1 and choice <= 3) and (board[0][choice - 1] != player) and (board[0][choice - 1] != computer):  
 board[0][choice - 1] = player  
 return True  
 # next we have assign every index a board index value to make it simple rather than subtracting  
 elif choice == 4 and (board[1][0] != player) and (board[1][0] != computer):  
 board[1][0] = player  
 return True  
  
 elif choice == 5 and (board[1][1] != player) and (board[1][1] != computer):  
 board[1][1] = player  
 return True  
  
 elif choice == 6 and (board[1][2] != player) and (board[1][2] != computer):  
 board[1][2] = player  
 return True  
  
 elif choice == 7 and (board[2][0] != player) and (board[2][0] != computer):  
 board[2][0] = player  
 return True  
  
 elif choice == 8 and (board[2][1] != player) and (board[2][1] != computer):  
 board[2][1] = player  
 return True  
  
 elif choice == 9 and (board[2][2] != player) and (board[2][2] != computer):  
 board[2][2] = player  
 return True  
 else:  
 return False  
  
  
# function for hecking winning conditions  
# argument current\_player  
  
def win\_condition(current\_player):  
 # for rows  
 # for checking if row 1 is completed with a current\_player  
 if (board[0][0] == current\_player) and (board[0][1] == current\_player) and (board[0][2] == current\_player):  
 return True  
 # for checking if row 2 is completed with a currentplayer  
 elif (board[1][0] == current\_player) and (board[1][1] == current\_player) and (board[1][2] == current\_player):  
 return True  
 # for checking if row 3 is completed with a currentplayer  
  
 elif (board[2][0] == current\_player) and (board[2][1] == current\_player) and (board[2][2] == current\_player):  
 return True  
 # for columns  
  
 # for checking if column 1 is completed with a currentplayer  
 elif (board[0][0] == current\_player) and (board[1][0] == current\_player) and (board[2][0] == current\_player):  
 return True  
 # for checking if column 2 is completed with a currentplayer  
 elif (board[0][2] == current\_player) and (board[1][2] == current\_player) and (board[2][2] == current\_player):  
 return True  
 # for checking if column 3 is completed with a currentplayer  
 elif (board[0][1] == current\_player) and (board[1][1] == current\_player) and (board[2][1] == current\_player):  
 return True  
 # for diagonals  
  
 # for checking if diagonal 1 is completed with a currentplayer  
 elif (board[0][0] == current\_player) and (board[1][1] == current\_player) and (board[2][2] == current\_player):  
 return True  
 # for checking if diagonal 2 is completed with a currentplayer  
 elif (board[0][2] == current\_player) and (board[1][1] == current\_player) and (board[2][0] == current\_player):  
 return True  
 # otherwise return with no player  
 else:  
 return False  
  
  
# function to ensure that computer can win in its turn or not  
# arguments computer, player  
def can\_cpu\_win(computer, player):  
 # checking if cpu can win anyway in this turn. Win it as first priority  
 # we will check if we have cover any two places in any win condition and third one is empty place and win  
 # checking every possibility  
 # for first rows  
 if (board[0][0] == computer) and (board[0][1] == computer) and (board[0][2] != player):  
 board[0][2] = computer  
 return True  
 elif (board[0][1] == computer) and (board[0][2] == computer) and (board[0][0] != player):  
 board[0][0] = computer  
 return True  
 elif (board[0][0] == computer) and (board[0][2] == computer) and (board[0][1] != player):  
 board[0][1] = computer  
 return True  
 # for 2nd row  
  
 elif (board[1][0] == computer) and (board[1][1] == computer) and (board[1][2] != player):  
 board[1][2] = computer  
 return True  
 elif (board[1][1] == computer) and (board[1][2] == computer) and (board[1][0] != player):  
 board[1][0] = computer  
 return True  
 elif (board[1][0] == computer) and (board[1][2] == computer) and (board[1][1] != player):  
 board[1][1] = computer  
 return True  
 # for 3rd row  
  
 elif (board[2][0] == computer) and (board[2][1] == computer) and (board[2][2] != player):  
 board[2][2] = computer  
 return True  
 elif (board[2][1] == computer) and (board[2][2] == computer) and (board[2][0] != player):  
 board[2][0] = computer  
 return True  
 elif (board[2][2] == computer) and (board[2][0] == computer) and (board[2][1] != player):  
 board[2][1] = computer  
 return True  
 # for first column  
  
 elif (board[0][0] == computer) and (board[1][0] == computer) and (board[2][0] != player):  
 board[2][0] = computer  
 return True  
 elif (board[1][0] == computer) and (board[2][0] == computer) and (board[0][0] != player):  
 board[0][0] = computer  
 return True  
 elif (board[0][0] == computer) and (board[2][0] == computer) and (board[1][0] != player):  
 board[1][0] = computer  
 return True  
 # for 3rd column  
 elif (board[0][2] == computer) and (board[1][2] == computer) and (board[2][2] != player):  
 board[2][2] = computer  
 return True  
 elif (board[1][2] == computer) and (board[2][2] == computer) and (board[0][2] != player):  
 board[0][2] = computer  
 return True  
 elif (board[2][2] == computer) and (board[0][2] == computer) and (board[1][2] != player):  
 board[2][2] = computer  
 return True  
 # for 2nd column  
 elif (board[0][1] == computer) and (board[1][1] == computer) and (board[2][1] != player):  
 board[2][1] = computer  
 return True  
 elif (board[1][1] == computer) and (board[2][1] == computer) and (board[0][1] != player):  
 board[0][1] = computer  
 return True  
 elif (board[2][1] == computer) and (board[0][1] == computer) and (board[1][1] != player):  
 board[1][1] = computer  
 return True  
 # for first diagonal  
 elif (board[0][0] == computer) and (board[1][1] == computer) and (board[2][2] != player):  
 board[2][2] = computer  
 return True  
 elif (board[1][1] == computer) and (board[2][2] == computer) and (board[0][0] != player):  
 board[0][0] = computer  
 return True  
 elif (board[0][0] == computer) and (board[2][2] == computer) and (board[1][1] != player):  
 board[1][1] = computer  
 return True  
 # for 2nd diagonal  
 elif (board[0][2] == computer) and (board[1][1] == computer) and (board[2][0] != player):  
 board[2][0] = computer  
 return True  
 elif (board[1][1] == computer) and (board[2][0] == computer) and (board[0][2] != player):  
 board[0][2] = computer  
 return True  
 elif (board[0][2] == computer) and (board[2][0] == computer) and (board[1][1] != player):  
 board[1][1] = computer  
 return True  
  
 else:  
 return False  
  
  
# to check whether comp can defend itself by blocking user win  
# arguments player, computer  
def block\_player\_win(player, computer):  
 # Blocking every possible win in top row  
 # for row 1  
 if (board[0][0] == player) and (board[0][1] == player) and (board[0][2] != player) and (board[0][2] != computer):  
 board[0][2] = computer  
 return True  
 elif (board[0][1] == player) and (board[0][2] == player) and (board[0][0] != player) and (board[0][0] != computer):  
 board[0][0] = computer  
 return True  
 elif (board[0][0] == player) and (board[0][2] == player) and (board[0][1] != player) and (board[0][1] != computer):  
 board[0][1] = computer  
 return True  
 # Blocking every possible win in middle row  
 elif (board[1][0] == player) and (board[1][1] == player) and (board[1][2] != player) and (board[1][2] != computer):  
 board[1][2] = computer  
 return True  
 elif (board[1][1] == player) and (board[1][2] == player) and (board[1][0] != player) and (board[1][0] != computer):  
 board[1][0] = computer  
 return True  
 elif (board[1][0] == player) and (board[1][2] == player) and (board[1][1] != player) and (board[1][1] != computer):  
 board[1][1] = computer  
 return True  
 # Blocking every possible win in bottom row  
 elif (board[2][0] == player) and (board[2][1] == player) and (board[2][2] != player) and (board[2][2] != computer):  
 board[2][2] = computer  
 return True  
 elif (board[2][1] == player) and (board[2][2] == player) and (board[2][0] != player) and (board[2][0] != computer):  
 board[2][0] = computer  
 return True  
 elif (board[2][2] == player) and (board[2][0] == player) and (board[2][1] != player) and (board[2][1] != computer):  
 board[2][1] = computer  
 return True  
 # Blocking every possible win in right column  
 elif (board[0][0] == player) and (board[1][0] == player) and (board[2][0] != player) and (board[2][0] != computer):  
 board[2][0] = computer  
 return True  
 elif (board[1][0] == player) and (board[2][0] == player) and (board[0][0] != player) and (board[0][0] != computer):  
 board[0][0] = computer  
 return True  
 elif (board[0][0] == player) and (board[2][0] == player) and (board[1][0] != player) and (board[1][0] != computer):  
 board[1][0] = computer  
 return True  
 # Blocking every possible win in left column  
 elif (board[0][2] == player) and (board[1][2] == player) and (board[2][2] != player) and (board[2][2] != computer):  
 board[2][2] = computer  
 return True  
 elif (board[1][2] == player) and (board[2][2] == player) and (board[0][2] != player) and (board[0][2] != computer):  
 board[0][2] = computer  
 return True  
 elif (board[2][2] == player) and (board[0][2] == player) and (board[1][2] != player) and (board[1][2] != computer):  
 board[1][2] = computer  
 return True  
 # Blocking every possible win in middle column  
 elif (board[0][1] == player) and (board[1][1] == player) and (board[2][1] != player) and (board[2][1] != computer):  
 board[2][1] = computer  
 return True  
 elif (board[1][1] == player) and (board[2][1] == player) and (board[0][1] != player) and (board[0][1] != computer):  
 board[0][1] = computer  
 return True  
 elif (board[2][1] == player) and (board[0][1] == player) and (board[1][1] != player) and (board[1][1] != computer):  
 board[1][1] = computer  
 return True  
 # Blocking every possible win on diagonal  
 elif (board[0][0] == player) and (board[1][1] == player) and (board[2][2] != player) and (board[2][2] != computer):  
 board[2][2] = computer  
 return True  
 elif (board[1][1] == player) and (board[2][2] == player) and (board[0][0] != player) and (board[0][0] != computer):  
 board[0][0] = computer  
 return True  
 elif (board[0][0] == player) and (board[2][2] == player) and (board[1][1] != player) and (board[1][1] != computer):  
 board[1][1] = computer  
 return True  
 # Blocking every possible win in other diagonal  
  
 elif (board[0][2] == player) and (board[1][1] == player) and (board[2][0] != player) and (board[2][0] != computer):  
 board[2][0] = computer  
 return True  
 elif (board[1][1] == player) and (board[2][0] == player) and (board[0][2] != player) and (board[0][2] != computer):  
 board[0][2] = computer  
 return True  
 elif (board[0][2] == player) and (board[2][0] == player) and (board[1][1] != player) and (board[1][1] != computer):  
 board[1][1] = computer  
 return True  
 # Blocking double cross  
 # corner double cross  
 # [1,9 then put in 6]  
 elif (board[0][0] == player) and (board[2][2] == player) and (board[1][2] != player) and (board[1][2] != computer):  
 board[1][2] = computer  
 return True  
 # [3,7 then put in 4]  
 elif (board[2][0] == player) and (board[0][2] == player) and (board[1][0] != player) and (board[1][0] != computer):  
 board[1][0] = computer  
 return True  
  
 # other double cross  
 # [9,2 then 3]  
 elif (board[2][2] == player) and (board[0][1] == player) and (board[0][2] != player) and (board[0][2] != computer):  
 board[0][2] = computer  
 return True  
 # [7,2 then 3]  
 elif (board[2][0] == player) and (board[0][1] == player) and (board[0][2] != player) and (board[0][2] != computer):  
 board[0][2] = computer  
 return True  
 # [1,8 then 9]  
 elif (board[0][0] == player) and (board[2][1] == player) and (board[2][2] != player) and (board[2][2] != computer):  
 board[2][2] = computer  
 return True  
 # [3,8 then 9]  
 elif (board[0][2] == player) and (board[2][1] == player) and (board[2][2] != player) and (board[2][2] != computer):  
 board[2][2] = computer  
 return True  
 # [7,6 then 9]  
 elif (board[2][0] == player) and (board[1][2] == player) and (board[2][2] != player) and (board[2][2] != computer):  
 board[2][2] = computer  
 return True  
 # [1,6 then 9]  
 elif (board[0][0] == player) and (board[1][2] == player) and (board[2][2] != player) and (board[2][2] != computer):  
 board[2][2] = computer  
 return True  
 # [9,4 then 1]  
 elif (board[2][2] == player) and (board[1][0] == player) and (board[0][0] != player) and (board[0][0] != computer):  
 board[0][0] = computer  
 return True  
 # [3,4 then 1]  
 elif (board[0][2] == player) and (board[1][0] == player) and (board[0][0] != player) and (board[0][0] != computer):  
 board[0][0] = computer  
 return True  
  
 else:  
 return False  
  
  
  
# computer strategy to win the game  
def brain\_of\_the\_cpu():  
 # if our previous function condition does not satisfy then it will make changes  
 if block\_player\_win(user, computer):  
 return True  
  
 # If middle square is open, always mark it as computer.  
 elif board[1][1] != user and board[1][1] != computer:  
 board[1][1] = computer  
 # marking the middle row if its the last options  
 elif turn > 6:  
  
 if board[1][0] != user and board[1][0] != computer:  
 board[1][0] = computer  
 if board[1][2] != user and board[1][2] != computer:  
 board[1][2] = computer  
  
 # Preventing getting double crossed by not marking randomly in the middle row  
 else:  
 while True:  
 x = randint(0, 2)  
 y = randint(0, 2)  
 if (board[x][y] != user and board[x][y] != computer) and (x != 1 and y != 1):  
 board[x][y] = computer  
 break  
  
  
game = True # global variable to start the game  
# we have to assign a varible true to start the game  
# first loop in processing  
# we use it to again start the game after complete game  
  
  
# we will use a while loop to exit anytime from the game or to restart the game  
while game:  
 # board to store our game data  
 board = [[' ', ' ', ' '], [' ', ' ', ' '], [' ', ' ', ' ']]  
 # turn ia a global variable which is initialized to zero and one will increment in it when player have done it's move  
 turn = 0  
 # toss global variable randomly generate one number btw 0 and 1  
 toss = randint(0, 1)  
 # choice global variable initialize to zero, in this variable we will get input from user where to put their symbol  
 choice = 0  
 #the first line that will be displayed  
 print("Welcome to tic tac toe")  
 #to display the board we will call the function  
 display\_board()  
 # asking username  
 user\_name = input("Enter your name: ")  
 user\_name = user\_name.capitalize()  
 print("-----------------------------------------")  
 print("Hi ", user\_name)  
 print("-----------------------------------------")  
 # 2nd loop to get user correct input of symbol  
 # we assign it true to get infinite loop  
 while True:  
 # user choice of symbol  
 user = input("Choose your symbol. X or O: ")  
 print("-----------------------------------------")  
 # upper function to capitalize the character in case user enter lowercase letter  
 user = user.upper()  
 # first condition to check input authentication  
 if user == 'X' or user == 'O':  
 # if input is valid loop will break  
 break  
 else:  
 print("Invalid Input. Please choose from the given choices.")  
 continue  
 # 2nd if condition  
 # assigning appropriate variables  
 if user.upper() == 'X':  
 computer = 'O'  
 else:  
 computer = 'X'  
 # 3rd loop to get user correct input of heads or tails  
 # we assign it true to get infinite loop  
 while True:  
 # user choice of symbol  
 user\_choice = int(input("Heads or Tails? press 0 if heads and 1 if tails: "))  
 print("-----------------------------------------")  
 # upper function to capitalize the character in case user enter lowercase letter  
 # forth if condition to check what user has chosen  
 if user\_choice == 0:  
 print(user\_name, "'s choice is now heads")  
 break  
 elif user\_choice == 1:  
 print(user\_name, "'s choice is now tails")  
 break  
 else:  
 print("Incorrect Option. Please choose from the given choices.")  
 continue  
  
 # Coin Toss  
 print("Computer is tossing the coin")  
 toss = randint(0, 1)  
 # fifth if condition to check what we got in random and check with our assigning  
 if toss == user\_choice:  
 print("You won the toss.You will go first.")  
 current\_player = user  
 else:  
 print("You lost the toss.Computer will go first.")  
 current\_player = computer  
  
 # invoking our first fuction to diplay our board  
 display\_board()  
 # Actual play of game  
 # 4th loop  
 # this loop is to handle turns and valid inputs from user and also computer move  
 # this is an infinte loop  
 while True:  
 # sixth condition  
 # first we will check our game is tied or not because this loop will execute till end  
  
 if turn >= 9:  
 print("Game tied")  
 # if our condition is true loop will break  
 break  
 # seventh condition  
 # to check if user has first turn then our current player is assign to user  
 if current\_player == user:  
 # take index from user  
 choice = int(input("Enter an index number of the board to put your selected symbol on the desired square(1-9): "))  
 # fifth infinite loop  
 # to check user input is valid or not to take input again  
 while True:  
 # eight condition  
 # also invoking our 2nd function through which we will get input on board  
 # if our function does not return with a true value then we will again take input  
 # otherwise loop will break  
 if not player\_turn(choice, current\_player, computer):  
 print("Invalid Input")  
 choice = int(input("Enter an index number of the board to put your selected symbol on the desired square(1-9): "))  
 else:  
 break  
 # invoking our function out of loop if user input is valid  
 player\_turn(choice, current\_player, computer)  
 # assigning the value on the board  
 display\_board()  
 print("Player turn ended")  
 # ninth condition  
 # to check have someone won or not  
 if win\_condition(current\_player):  
 # invoking our 3rd function  
 # it will bring winner as current player if win condition meet  
 win\_condition(current\_player)  
 print("Player Wins!")  
 break  
 # if there is no winner till then we will shift our current player to computer  
 # by using our variable turn  
 # shifting player  
 turn += 1  
 current\_player = computer  
 # to go outside of our nested loop  
 continue  
 # tenth if conditon to check if there is computer turn  
 # DESIGN OF THE COMPUTER  
 if current\_player == computer:  
 # First priority of computer is to win  
 # eleventh condition to check can computer win in ths turn  
 # by invoking our 4th function  
 # if our function return with true value than winner is computer  
 if can\_cpu\_win(computer, user):  
 display\_board()  
 win\_condition(computer)  
 print("Computer WINS!")  
 break  
 # if it cant win, Block enemy player win if possible  
 else:  
 # invoking our fifth function  
 # to attack on our opponent  
 brain\_of\_the\_cpu()  
 display\_board()  
 print("Computer has turned")  
 # twelveth condition  
 # again checking winning condition to return with current player as winner  
 if win\_condition(current\_player):  
 win\_condition(current\_player)  
 print("COMPUTER WINS")  
 break  
 # shifting player  
 turn += 1  
 current\_player = user  
 # continue to our outermost loop  
 continue  
 # asking user to play again or not  
  
 play\_again = input("Would u like to play this game again? Press [Y] for Yes or [N] for No.")  
 play\_again = play\_again.upper()  
 # thirteen condition  
 # to check user input and taking acton according to it  
 if play\_again == "Y":  
 continue  
 else:  
 break