1/10/24
TIC-TAC-TOE Game

Code:
```
import random

def check_win(board, r, c):
   if board[r - 1][c - 1] == 'X':
       ch = "O"
   else:
       ch = "X"
   if ch not in board[r - 1] and '-' not in board[r - 1]:
       return True
   elif ch not in (board[0][c - 1], board[1][c - 1], board[2][c - 1]) and
'-' not in (board[0][c - 1], board[1][c - 1], board[2][c - 1]):
       return True
   elif ch not in (board[0][0], board[1][1], board[2][2]) and '-' not in
(board[0][0], board[1][1], board[2][2]):
       return True
   elif ch not in (board[0][2], board[1][1], board[2][0]) and '-' not in
(board[0][2], board[1][1], board[2][0]):
       return True
   return False

def display_board(board):
   for row in board:
       print(row)

def find_block_move(board):
   # Check rows and columns for blocking opportunity
   for i in range(3):
       # Check rows
       if board[i].count('X') == 2 and board[i].count('-') == 1:
           return i, board[i].index('-')
       # Check columns
       col = [board[0][i], board[1][i], board[2][i]]
       if col.count('X') == 2 and col.count('-') == 1:
           return col.index('-'), i

   # Check diagonals for blocking opportunity
   diag1 = [board[0][0], board[1][1], board[2][2]]
   if diag1.count('X') == 2 and diag1.count('-') == 1:
       idx = diag1.index('-')
       return idx, idx

   diag2 = [board[0][2], board[1][1], board[2][0]]
   if diag2.count('X') == 2 and diag2.count('-') == 1:
       idx = diag2.index('-')
       return idx, 2 - idx

   return None  # No blocking move found

def bot_move(board):
   # First, check if there's a move to block the human
   block_move = find_block_move(board)
   if block_move:
       r, c = block_move
       board[r][c] = 'O'
       print(f"Bot blocked X at position: ({r + 1}, {c + 1})")
       display_board(board)
       return r + 1, c + 1

   # Otherwise, make a random move
   available_moves = [(r, c) for r in range(3) for c in range(3) if
board[r][c] == '-']
   if available_moves:
       move = random.choice(available_moves)
       board[move[0]][move[1]] = 'O'
       print(f"Bot placed O at position: ({move[0] + 1}, {move[1] + 1})")
       display_board(board)
       return move[0] + 1, move[1] + 1  # Return the move for win check
   return None, None

# Initial board setup
board = [['-', '-', '-'], ['-', '-', '-'], ['-', '-', '-']]
display_board(board)

xo = 1  # 1 for human, 0 for bot
flag = 0  # Flag to check for win or draw
```

```
while '-' in board[0] or '-' in board[1] or '-' in board[2]:

   if xo == 1:  # Human's turn (X)
       print("Enter position to place X (row and column between 1-3):")
       x = int(input())
       y = int(input())
       if x > 3 or y > 3 or x < 1 or y < 1:
           print("Invalid position")
           continue
       if board[x - 1][y - 1] == '-':
           board[x - 1][y - 1] = 'X'
           xo = 0  # Switch to bot's turn
           display_board(board)
       else:
           print("Invalid position")
           continue

       if check_win(board, x, y):
           print("X wins!")
           flag = 1
           break

   else:  # Bot's turn (O)
       print("Bot's turn:")
       x, y = bot_move(board)
       if x and y:  # If bot made a valid move
           xo = 1  # Switch back to human's turn
           if check_win(board, x, y):
               print("O (Bot) wins!")
               flag = 1
               break

if flag == 0:
   print("Draw")
print("Game Over")
```

01/10/24

LAB-1

TIC - TAC - TOE GAME

ALGORITHM:

```
check_win(board, r, c)
    Step-1: Identify which letter was placed (X or O)
    Step-2: Check the corresponding row, column
            and diagonals for a win
            • If all cells in a row, column or
              diagonal match the player's letter (with no
              empty cells) return true
            • Else return False

display_board(board)
    Step 1: Print all rows of the board

find_block_move(board):
    Step-1: Check each row, col, diagonals for two X and
            one empty spot (-)
            • If found, return the position to block the
              player
            • If no block return none.

bot_move(board)
    Step-1: Call find_block_move(board). If found
            place O there
    Step-2: If no block is needed, choose a
            random available move.

Main algo:
    Step-1: Initialize 3x3 board with '-'
    Step-2: Set X O (1 for player, 0 for bot)
```

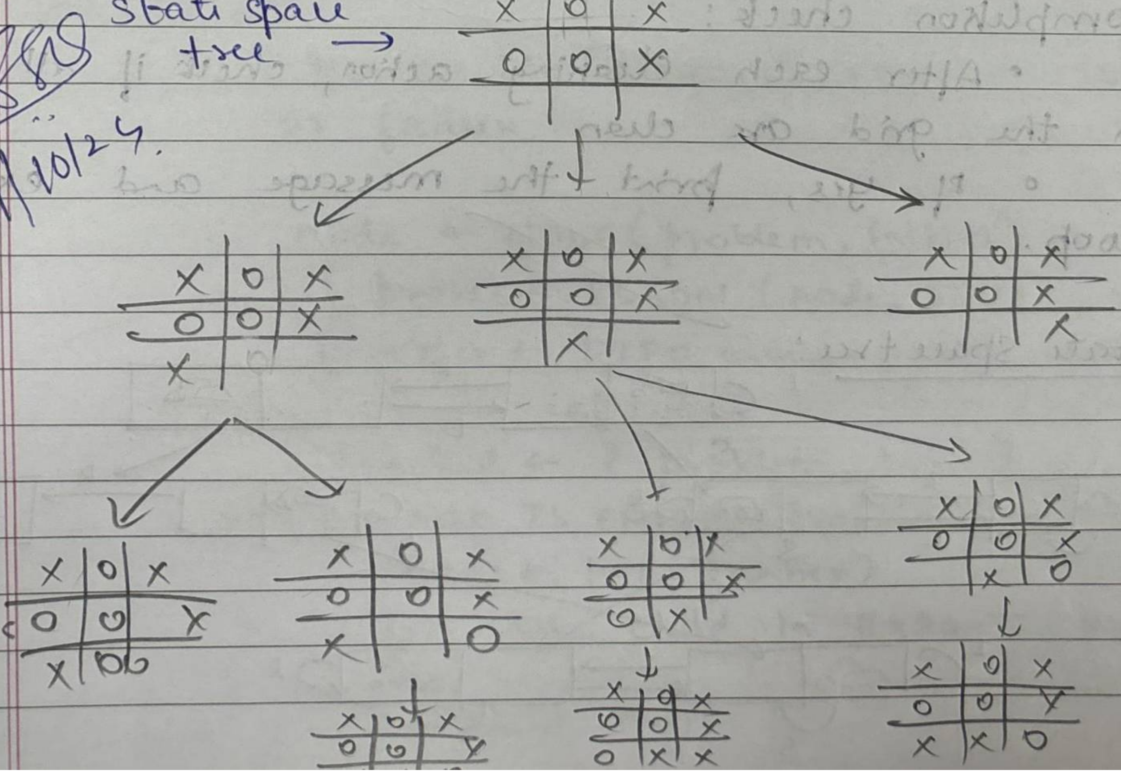
```
and flag to check the game status.
        Step-3: While there empty spots
            • If player's turn (x)
                → prompt for row and column
                → validate the ~~plate~~ position and
              place x.
                → check for win
          Bot's Turn(o)
          ~~step 4:~~
            → call bot_move(board)
              check for win
Step-4: If no winner, print 'Draw')
Step-5: Print game over.
```

Output:

```
[ -   -   - ]
[ -   -   - ]
[ -   -   - ]
Enter position to place x
1
1
[ x , - , - ]
[ - , - , - ]
[ - , - , - ]
Bot's turn
Bot placed at position (3,3)
[ x , - , - ]
[ - , - , - ]
[ - , - , o ]
Enter position to place x
1
2
```

```
[ x , x , - ]
[ - , - , - ]
[ - , - , 0 ]
Bot's turn:
Bot blocked x at position(1,3)
[ x , x , 0 ]
[ - , - , - ]
[ - , - , 0 ]
Enter position of x
2
2
[ x , x , 0 ]
[ - , x , - ]
[ - , 0 , 0 ]
Bot's turn:
Bot blocked x at position (3,1)
[ x , x , 0 ]
[ - , x , - ]
[ 0 , 0 , 0 ]
0 (Bot) won!
Game over.
```

State space tree →



[illegible] 11/10/24.