Experiment-3

Yash Patel
2019130047
Batch C
TE Comps

Aim:
Implement TicTacToe using a* Algorithm

Code:

```
board = {1: " ", 2: " ", 3: " ",
        4: " ", 5: " ", 6: " ",
        7: " ", 8: " ", 9: " "}

player = "O"
bot = "X"



def print_board(board):
   print(board[1] + "|" + board[2] + "|" + board[3])
   print("-+-+-")
   print(board[4] + "|" + board[5] + "|" + board[6])
   print("-+-+-")
   print(board[7] + "|" + board[8] + "|" + board[9])
   print("\n")




def is_empty_cell(position):
   if board[position] == " ":
       return True
   return False


def check_draw():
   for key in board:
       if board[key] == " ":
```

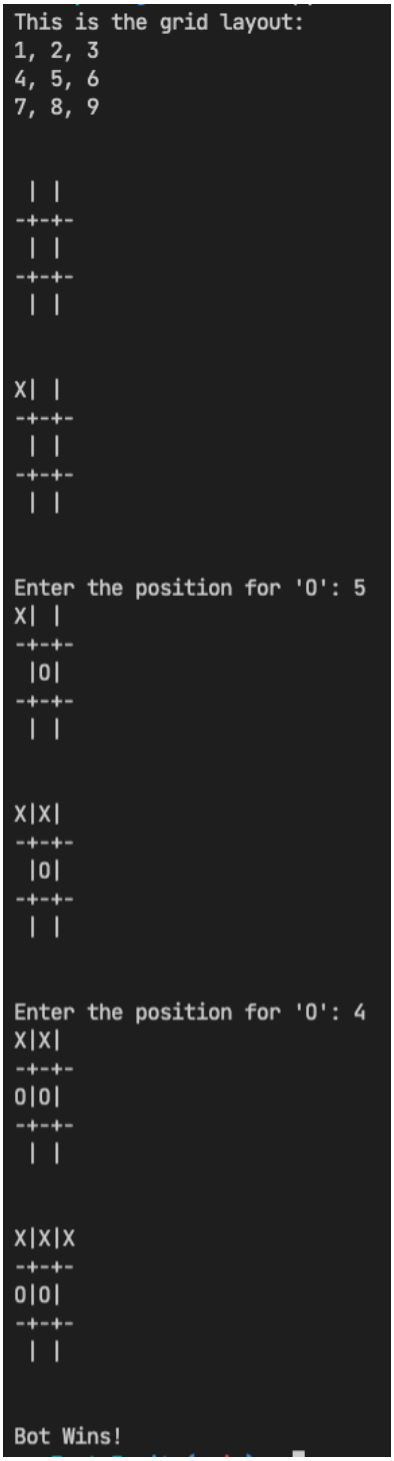
```
            return False
    return True


def check_win():
    if board[1] == board[2] and board[1] == board[3] and board[1] != ' ':
        return True
    elif board[4] == board[5] and board[4] == board[6] and board[4] != ' ':
        return True
    elif board[7] == board[8] and board[7] == board[9] and board[7] != ' ':
        return True
    elif board[1] == board[4] and board[1] == board[7] and board[1] != ' ':
        return True
    elif board[2] == board[5] and board[2] == board[8] and board[2] != ' ':
        return True
    elif board[3] == board[6] and board[3] == board[9] and board[3] != ' ':
        return True
    elif board[1] == board[5] and board[1] == board[9] and board[1] != ' ':
        return True
    elif board[7] == board[5] and board[7] == board[3] and board[7] != ' ':
        return True
    else:
        return False


def insert_letter(letter, position):
    if is_empty_cell(position):
        board[position] = letter
        print_board(board)
        if check_draw():
            print("Draw!")
            exit()

        if check_win():
            if letter == "X":
                print("Bot Wins!")
```

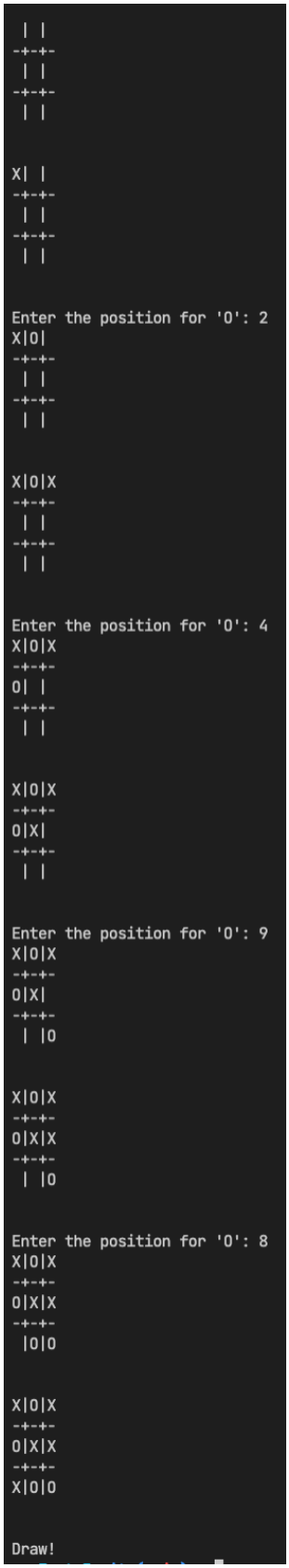
```
            else:
                print("You win!")
            exit()

    else:
        print("Can't insert there!")
        position = int(input("Enter new position: "))
        insert_letter(letter, position)
        return


def check_which_mark_won(mark):
    if board[1] == board[2] and board[1] == board[3] and board[1] != mark:
        return True
    elif board[4] == board[5] and board[4] == board[6] and board[4] !=
mark:
        return True
    elif board[7] == board[8] and board[7] == board[9] and board[7] !=
mark:
        return True
    elif board[1] == board[4] and board[1] == board[7] and board[1] !=
mark:
        return True
    elif board[2] == board[5] and board[2] == board[8] and board[2] !=
mark:
        return True
    elif board[3] == board[6] and board[3] == board[9] and board[3] !=
mark:
        return True
    elif board[1] == board[5] and board[1] == board[9] and board[1] !=
mark:
        return True
    elif board[7] == board[5] and board[7] == board[3] and board[7] !=
mark:
        return True
    else:
```

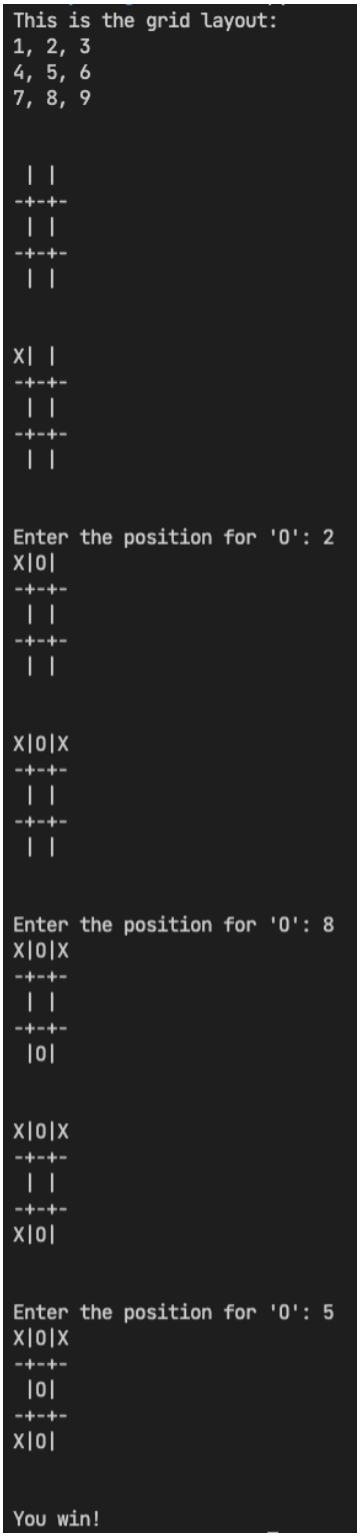
```
        return False


def find_score(board, is_maximum):
    if check_which_mark_won(bot):
        return 1
    elif check_which_mark_won(player):
        return -1
    elif check_draw():
        return 0

    if is_maximum:
        best_score = -800
        for key in board.keys():
            if board[key] == " ":
                board[key] = bot
                score = find_score(board, False)
                board[key] = " "
                if score > best_score:
                    best_score = score
        return best_score
    else:
        best_score = 800
        for key in board.keys():
            if board[key] == " ":
                board[key] = player
                score = find_score(board,True)
                board[key] = " "
                if score < best_score:
                    best_score = score
        return best_score


def player_move():
    position = int(input("Enter the position for 'O': "))
    insert_letter(player, position)
```

```
    return


def bot_move():
    best_score = -800
    best_move = 0

    for key in board.keys():
        if board[key] == " ":
            board[key] = bot
            score = find_score(board, False)
            board[key] = " "
            if score > best_score:
                best_score = score
                best_move = key
    insert_letter(bot, best_move)
    return


def show_layout():
  print("This is the grid layout:")
  print("1, 2, 3 ")
  print("4, 5, 6 ")
  print("7, 8, 9 ")
  print("\n")

show_layout()
print_board(board)


while not check_win():
    bot_move()
    player_move()
```

Output:


This is the grid layout:
1, 2, 3
4, 5, 6
7, 8, 9
Enter the position for 'O': 5
Enter the position for 'O': 4
Bot Wins!


Enter the position for 'O': 2
Enter the position for 'O': 4
Enter the position for 'O': 9
Enter the position for 'O': 8
Draw!


This is the grid layout:
1, 2, 3
4, 5, 6
7, 8, 9
Enter the position for 'O': 2
Enter the position for 'O': 8
Enter the position for 'O': 5
You win!

Conclusion:

The purpose of this experiment was to use the informed search approach to implement the tic-tac-toe game. To find the best place to put the 'X' or 'O,' I first calculated the difference between the winning combinations of bot(X) and user(O) for each choice in that round, then for the maximum values obtained, I calculated which choice would not lead to computer victory and which would lead to user victory by checking the number of moves required for victory for each choice and selecting the one with the fewest moves. If there are multiple movements that are comparable, the piece is placed at random. Download the tic-tac-toe.py file and run the python code to view the demonstration.