Pranav Nair
2019130042
TE Comps Batch – C
26th October, 2021

Experiment-3

Aim: To implement the game, Tic-Tac-Toe using the A* search strategy

Code:

```
# if current can win
# if opponent can win
# calculate heuristic
# after taking all moves for maximum values calculate if there is any position with win in possible next move but check if there is next move possible
# select the random possibility

from tkinter import *
window=Tk()

window.title('Tic-Tac-Toe')
window.geometry("450x350+900+250")


def on_click(box_number, symbol):
    if box_number == 1:
        btn1['text'] = symbol
        if symbol == 'X':
            btn1['bg'] = 'red'
        else:
            btn1['bg'] = 'blue'

    elif box_number == 2:
        btn2['text'] = symbol
        if symbol == 'X':
            btn2['bg'] = 'red'
        else:
            btn2['bg'] = 'blue'

    elif box_number == 3:
        btn3['text'] = symbol
        if symbol == 'X':
            btn3['bg'] = 'red'
        else:
            btn3['bg'] = 'blue'

    elif box_number == 4:
        btn4['text'] = symbol
```

```
            if symbol == 'X':
                btn4['bg'] = 'red'
            else:
                btn4['bg'] = 'blue'

        elif box_number == 5:
            btn5['text'] = symbol
            if symbol == 'X':
                btn5['bg'] = 'red'
            else:
                btn5['bg'] = 'blue'

        elif box_number == 6:
            btn6['text'] = symbol
            if symbol == 'X':
                btn6['bg'] = 'red'
            else:
                btn6['bg'] = 'blue'

        elif box_number == 7:
            btn7['text'] = symbol
            if symbol == 'X':
                btn7['bg'] = 'red'
            else:
                btn7['bg'] = 'blue'

        elif box_number == 8:
            btn8['text'] = symbol
            if symbol == 'X':
                btn8['bg'] = 'red'
            else:
                btn8['bg'] = 'blue'

        elif box_number == 9:
            btn9['text'] = symbol
            if symbol == 'X':
                btn9['bg'] = 'red'
            else:
                btn9['bg'] = 'blue'

    btn1=Button(window, width=10, height=5 )
    btn1.place(x=100, y=50)
    btn4=Button(window, width=10, height=5 )
    btn4.place(x=100, y=135)
    btn7=Button(window, width=10, height=5 )
    btn7.place(x=100, y=220)
    btn2=Button(window, width=10, height=5 )
    btn2.place(x=180, y=50)
    btn5=Button(window, width=10, height=5 )
    btn5.place(x=180, y=135)
    btn8=Button(window, width=10, height=5 )
    btn8.place(x=180, y=220)
    btn3=Button(window, width=10, height=5)
    btn3.place(x=260, y=50)
```

```
btn6=Button(window, width=10, height=5)
btn6.place(x=260, y=135)
btn9=Button(window, width=10, height=5)
btn9.place(x=260, y=220)

import random
import time

num = [0,1,2,3,4,5,6,7,8]
board = ['0', '0', '0', '0', '0', '0', '0', '0', '0']
board_row = [['0', '0', '0'], ['0', '0', '0'], ['0', '0', '0']]
board_column = [['0', '0', '0'], ['0', '0', '0'], ['0', '0', '0']]
board_diagonal = [['0', '0', '0'], ['0', '0', '0']]

#check for win
def player_win(player):
    over = False
    for i in range(3):
        if board_row[i].count(player) == 3:
            over = True
            break

        if board_column[i].count(player) == 3:
            over = True
            break

        if i< 2:
            if board_diagonal[i].count(player) == 3:
                over = True
                break

    if over :
        return True
    else:
        return False

#check if any player has winning moves
def check_wins():
    win_X = []
    win_O = []
    for i in range(3):
        if board_row[i].count('0') == 1 :
            index_zero = board_row[i].index('0')
            index_zero = 3*i + index_zero
            if board_row[i].count('X') == 2 :
                win_X.append(index_zero)
            elif board_row[i].count('O') == 2 :
                win_O.append(index_zero)

        if board_column[i].count('0') == 1 :
            index_zero = board_column[i].index('0')
            index_zero = i + (index_zero * 3)
            if board_column[i].count('X') == 2 :
                win_X.append(index_zero)
```

```
            elif board_column[i].count('O') == 2 :
                win_O.append(index_zero)
        if i<2 :
            if board_diagonal[i].count('0') == 1 :
                index_zero = board_diagonal[i].index('0')
                if i==0:
                    index_zero = 4 * index_zero
                elif i==1:
                    index_zero = 2 + (index_zero * 2)
                if board_diagonal[i].count('X') == 2 :
                    win_X.append(index_zero)
                elif board_diagonal[i].count('O') == 2 :
                    win_O.append(index_zero)

    return [win_O , win_X]

def heuristic(current_player):
    empty = []
    for i in range(9):
        if(board[i] == '0'):
            empty.append(i)

    heuristic_values = []
    heuristic_values_choices = []

    for choice in empty:
        X_combo = 0
        O_combo = 0

        x = choice//3
        y = choice%3
        board[choice] = current_player
        board_row[x][y] = current_player
        board_column[y][x] = current_player
        if x==y:
            board_diagonal[0][x] = current_player
            if x == 1:
                board_diagonal[1][x] = current_player
        if choice == 2 or choice == 6:
            z = choice//2 - 1
            board_diagonal[1][z] = current_player

        empty_minus_one = len(empty) - 1

        for i in range(3):
            if board_row[i].count('0') == 3:
                if(empty_minus_one == 5):
                    if current_player == 'O':
                        X_combo = X_combo + 1
                    else:
                        O_combo = O_combo + 1
            elif board_row[i].count('0') == 2:
                if empty_minus_one > 3:
                    if board_row[i].count('X') == 1:
```

```
                X_combo = X_combo + 1
            elif board_row[i].count('O') == 1:
                O_combo = O_combo + 1
        elif empty_minus_one == 3:
            if board_row[i].count('X') == 1:
                if current_player == 'O':
                    X_combo = X_combo + 1
            elif board_row[i].count('O') == 1:
                if current_player == 'X':
                    O_combo = O_combo + 1
    elif board_row[i].count('0') == 1:
        if empty_minus_one > 1:
            if board_row[i].count('X') == 2:
                X_combo = X_combo + 1
            elif board_row[i].count('O') == 2:
                O_combo = O_combo + 1
        elif empty_minus_one == 1:
            if board_row[i].count('X') == 2:
                if current_player == 'O':
                    X_combo = X_combo + 1
            elif board_row[i].count('O') == 2:
                if current_player == 'X':
                    O_combo = O_combo + 1

    if board_column[i].count('0') == 3:
        if(empty_minus_one == 5):
            if current_player == 'O':
                X_combo = X_combo + 1
            else:
                O_combo = O_combo + 1
    elif board_column[i].count('0') == 2:
        if empty_minus_one > 3:
            if board_column[i].count('X') == 1:
                X_combo = X_combo + 1
            else:
                O_combo = O_combo + 1
        elif empty_minus_one == 3:
            if board_column[i].count('X') == 1:
                if current_player == 'O':
                    X_combo = X_combo + 1
            elif board_column[i].count('O') == 1:
                if current_player == 'X':
                    O_combo = O_combo + 1
    elif board_column[i].count('0') == 1:
        if empty_minus_one > 1:
            if board_column[i].count('X') == 2:
                X_combo = X_combo + 1
            elif board_column[i].count('O') == 2:
                O_combo = O_combo + 1
        elif empty_minus_one == 1:
            if board_column[i].count('X') == 2:
                if current_player == 'O':
                    X_combo = X_combo + 1
            elif board_column[i].count('O') == 2:
```

```
                if current_player == 'X':
                    O_combo = O_combo + 1

    if i<2:
        if board_diagonal[i].count('0') == 3:
            if(empty_minus_one == 5):
                if current_player == 'O':
                    X_combo = X_combo + 1
                else:
                    O_combo = O_combo + 1
        elif board_diagonal[i].count('0') == 2:
            if empty_minus_one > 3:
                if board_diagonal[i].count('X') == 1:
                    X_combo = X_combo + 1
                else:
                    O_combo = O_combo + 1
            elif empty_minus_one == 3:
                if board_diagonal[i].count('X') == 1:
                    if current_player == 'O':
                        X_combo = X_combo + 1
                elif board_diagonal[i].count('O') == 1:
                    if current_player == 'X':
                        O_combo = O_combo + 1
        elif board_diagonal[i].count('0') == 1:
            if empty_minus_one > 1:
                if board_diagonal[i].count('X') == 2:
                    X_combo = X_combo + 1
                elif board_column[i].count('O') == 2:
                    O_combo = O_combo + 1
            elif empty_minus_one == 1:
                if board_diagonal[i].count('X') == 2:
                    if current_player == 'O':
                        X_combo = X_combo + 1
                elif board_diagonal[i].count('O') == 2:
                    if current_player == 'X':
                        O_combo = O_combo + 1
board[choice] = '0'
board_row[x][y] = '0'
board_column[y][x] = '0'
if x==y:
    board_diagonal[0][x] = '0'
    if x == 1:
        board_diagonal[1][x] = '0'
if choice == 2 or choice == 6:
    z = choice//2 - 1
    board_diagonal[1][z] = '0'

if current_player == 'X':
    heuristic_values.append(X_combo-O_combo)
else:
    heuristic_values.append(O_combo-X_combo)


heuristic_values_choices.append(choice)
```

```
    final_heuristics = []
    max_value = max(heuristic_values)
    for i in range(len(heuristic_values)):
        if heuristic_values[i] == max_value:
            final_heuristics.append(heuristic_values_choices[i])

    if len(empty) >= 7:
        return random.choice(final_heuristics)

    final_positions=[]
    min_values = []

    for choice in final_heuristics:
        x = choice//3
        y = choice%3
        board[choice] = current_player
        board_row[x][y] = current_player
        board_column[y][x] = current_player
        if x==y:
            board_diagonal[0][x] = current_player
            if x == 1:
                board_diagonal[1][x] = current_player
        if choice == 2 or choice == 6:
            z = choice//2 - 1
            board_diagonal[1][z] = current_player

        checks = check_wins()
        if current_player == 'X':
            min_values.append(len(checks[0]))
        else:
            min_values.append(len(checks[1]))

        board[choice] = '0'
        board_row[x][y] = '0'
        board_column[y][x] = '0'
        if x==y:
            board_diagonal[0][x] = '0'
            if x == 1:
                board_diagonal[1][x] = '0'
        if choice == 2 or choice == 6:
            z = choice//2 - 1
            board_diagonal[1][z] = '0'

    min_val = min(min_values)
    for i in range(len(min_values)):
        if min_values[i] == min_val:
            final_positions.append(final_heuristics[i])

    return random.choice(final_positions)



def main():
```

```
game_won = False
player = ['X' , 'O']
start_Player = random.choice(player)
total_moves = 0

print("User is " + start_Player + "\nComp is ",end=' ')
if(start_Player=='X'):
    print("O")
else:
    print('X\n\n')

while(total_moves < 9):
    for i in range(3):
        for j in range(3):
            if board_row[i][j] == '0':
                print('-',end=' ')
            else:
                print(board_row[i][j], end=" ")
        print()
    print()

    #user vs comp
    if total_moves%2 == 0:
        choice = int(input("Enter your choice from 0 - 8: "))
        while choice not in num:
            choice = int(input("Enter your choice from 0 - 8: "))

    else:
        if total_moves < 2:
            choice = random.choice(num)
        else:
            my_win = check_wins()
            choice = -1
            if start_Player == 'O':
                if len(my_win[0]) > 0:
                    choice = my_win[0][0]
            elif start_Player == 'X':
                if len(my_win[1]) > 0:
                    choice = my_win[1][0]
            if choice == -1:
                if start_Player == 'O':
                    if len(my_win[1]) > 0:
                        choice = my_win[1][0]
                elif start_Player == 'X':
                    if len(my_win[0]) > 0:
                        choice = my_win[0][0]

            if choice == -1:
                choice = heuristic(start_Player)

    # comp vs comp
    # if total_moves < 2:
    #     choice = random.choice(num)
    # else:
```

```
        #     my_win = check_wins()
        #     choice = -1
        #     if start_Player == 'O':
        #         if len(my_win[0]) > 0:
        #             choice = my_win[0][0]
        #     elif start_Player == 'X':
        #         if len(my_win[1]) > 0:
        #             choice = my_win[1][0]
        #     if choice == -1:
        #         if start_Player == 'O':
        #             if len(my_win[1]) > 0:
        #                 choice = my_win[1][0]
        #         elif start_Player == 'X':
        #             if len(my_win[0]) > 0:
        #                 choice = my_win[0][0]

        #     if choice == -1:
        #         choice = heuristic(start_Player)
        # time.sleep(1)
        num.remove(choice)
        print(str(choice) + " " + start_Player)
        on_click(choice+1 , start_Player)
        x = choice//3
        y = choice%3
        board[choice] = start_Player
        board_row[x][y] = start_Player
        board_column[y][x] = start_Player
        if x==y:
            board_diagonal[0][x] = start_Player
            if x == 1:
                board_diagonal[1][x] = start_Player
        if choice == 2 or choice == 6:
            z = choice//2 - 1
            board_diagonal[1][z] = start_Player

        total_moves = total_moves + 1

        if player_win(start_Player):
            print("\n" + start_Player + " WON!!\n")
            game_won = True
            break

        if start_Player == 'X':
            start_Player = 'O'
        else:
            start_Player = 'X'

        time.sleep(0.5)


    for i in range(3):
        for j in range(3):
            if board_row[i][j] == '0':
                    print('-',end=' ')
```

```
            else:
                print(board_row[i][j], end=" ")
        print()
    print()

    if not game_won:
        print("\nTIE!!")


if __name__ == "__main__":
    main()

window.mainloop()
```

Conclusion:

In this experiment, I have implemented the Tic-Tac-Toe game in python using A* search strategy. Here to place the ‘X’ or ‘O’, first I calculated the difference between the current player’s winning combinations possible and opponent’s winning combinations possible for each move possible. Then taking all the moves with the maximum value, I then calculated that for each of these moves if there is move which can make the number of moves to goal state lesser for the current user and more for the opponent. Then after filtering the possible moves using this value along with the first heuristic value, the move is executed. If there are multiple moves after filtering, then a random move is executed. In particular cases, if there is a single move left for the opponent’s victory and multiple empty positions available, the heuristic value for this one position can be less than some other empty slot which won’t lead to the current player’s victory, so to avoid this scenario, there is a function which check if there is a single move victory available for either of the players and the places the ‘X’ or ‘O’ accordingly. The code can demonstrate a user vs computer mode and also computer vs computer mode. Since A* is used for the computer’s move, it is always the optimal move. In user vs comp mode, due to the user’s intelligence the number of cases where ‘TIE’ occurs is less compared to comp vs comp where the number of ‘TIE’ is more than ‘WIN’ as each move in this case is an optimal move played by the computer.

GitHub: https://github.com/pranav567/AI-ML-Lab/tree/master/experiment-3