AI LAB EXP-6

Implementation of minimax algorithm for an application

Name:Rahul Goel

Reg No: RA1911030010094

Aim: To implement mini-max algorithm as a tic tac toe game using python **Algorithm:**

Mini-max algorithm is a recursive or backtracking algorithm which is used in decision-making and game theory. It provides an optimal move for the player assuming that opponent is also playing optimally. Min-Max algorithm is mostly used for game playing in AI. Such as Chess, Checkers, tic-tac-toe, go, and various tow-players game. This Algorithm computes the minimax decision for the current state.

Code:

```
import random
class TicTacToe(object):

winning_combos = (
[0, 1, 2], [3, 4, 5], [6, 7, 8], [0, 3, 6], [1, 4, 7], [2, 5, 8], [0, 4, 8],
[2, 4, 6]

)

winners = ('X-win', 'Draw', 'O-win')

def __init__(self, board=[]): if len(board) == 0:

self.board = [0 for i in range(9)]
```

```
else:
self.board = board

def print_board(self): for i in range(3):

print(
"| " + str(self.board[i * 3]) +
" | " + str(self.board[i * 3 + 1]) +
" | " + str(self.board[i * 3 + 2]) + " |"

)

def check_game_over(self):
if 0 not in [element for element in self.board]:

return True
if self.winner() != 0:

return True return False

def available_moves(self):
return [index for index, element in enumerate(self.board) if
element is 0]

def available_combos(self, player):
return self.available_moves() + self.get_acquired_places(player)

def X_won(self):
return self.winner() == 'X'

def O_won(self):
return self.winner() == 'O'

def is_tie(self):
return self.winner() == 0 and self.check_game_over()
```

```
def winner(self):
for player in ('X', 'O'):

positions = self.get_acquired_places(player) for combo in self.winning_combos:

win = True
for pos in combo:

if pos not in positions: win = False

if win:
return player

return 0

def get_acquired_places(self, player):

return [index for index, element in enumerate(self.board) if element == player]

def make_move(self, position, player): self.board[position] = player

def minimax(self, node, player): if node.check_game_over():

if node.X_won(): return -1

elif node.is_tie(): return 0

elif node.O_won(): return 1

best = 0
for move in node.available_moves():

node.make_move(move, player)
val = self.minimax(node, get_enemy(player))
```

```
node.make_move(move, 0)
if player == 'O':

if val > best: best = val

else:
if val < best:

best = val return best

def determine(board, player): '''

Driver function to apply minimax algorithm '''

a=0
choices = []
if len(board.available_moves()) == 9:

return 4
for move in board.available_moves():

board.make_move(move, player)
val = board.minimax(board, get_enemy(player))
board.make_move(move, 0)
if val > a:

a = val

choices = [move] elif val == a:

choices.append(move) try:

return random.choice(choices) except IndexError:

return random.choice(board.available_moves())

def get_enemy(player): if player == 'X':
```

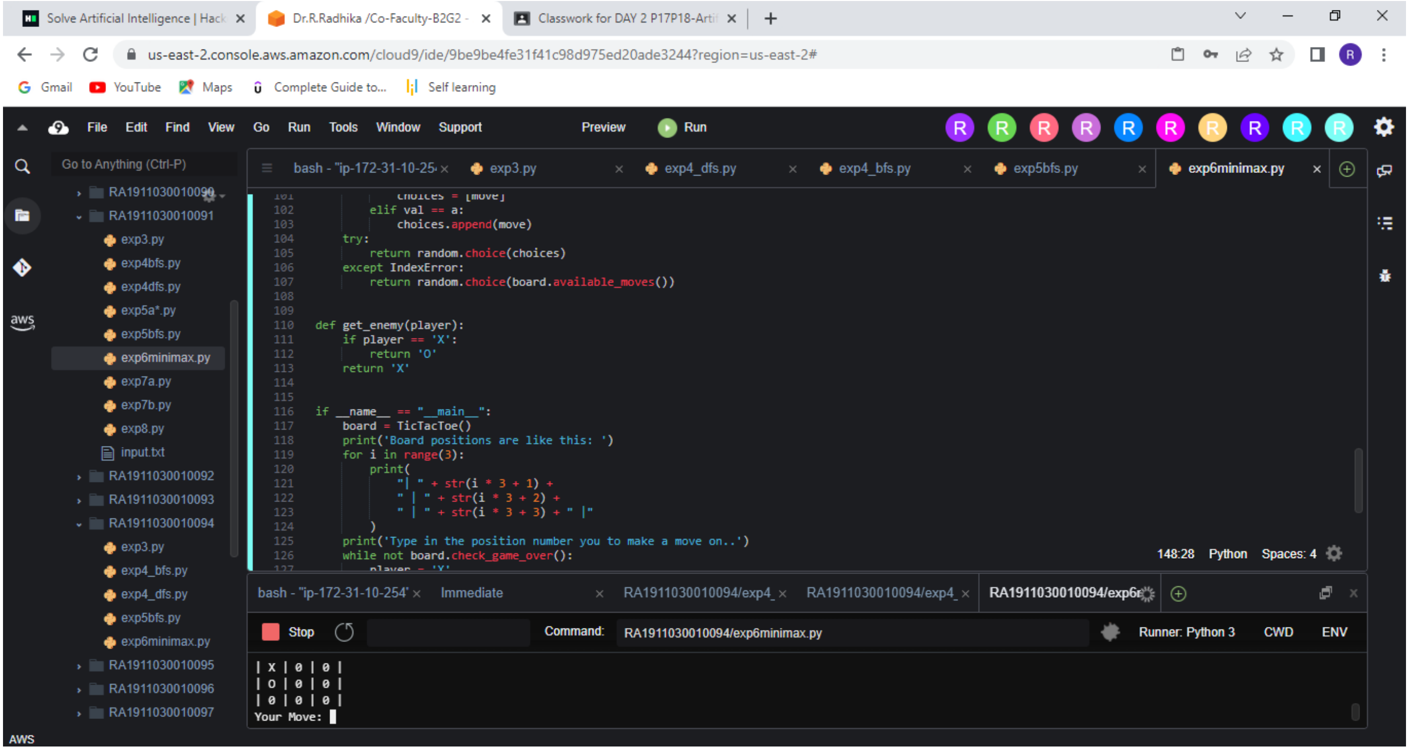
```
return 'O' return 'X'

if __name__ == "__main__":

board = TicTacToe()
print('Board positions are like this: ') for i in range(3):

print(
"| " + str(i * 3 + 1) +
" | " + str(i * 3 + 2) +
" | " + str(i * 3 + 3) + " |"

)
print('Type in the position number you to make a move on..')
while not board.check_game_over():

player = 'X'
player_move = int(input("Your Move: ")) - 1
if player_move not in board.available_moves():

print('Please check the input!')

continue board.make_move(player_move, player)
board.print_board()
print()
if board.check_game_over():

break
print('Computer is playing.. ')
player = get_enemy(player) computer_move = determine(board,
player) board.make_move(computer_move, player)
board.print_board()

if board.winner() != 0:
```

if board.winner() == 'X':
print ("Congratulations you win!")

else:
print('Computer Wins!')

else:
print("Game tied!")

Output:

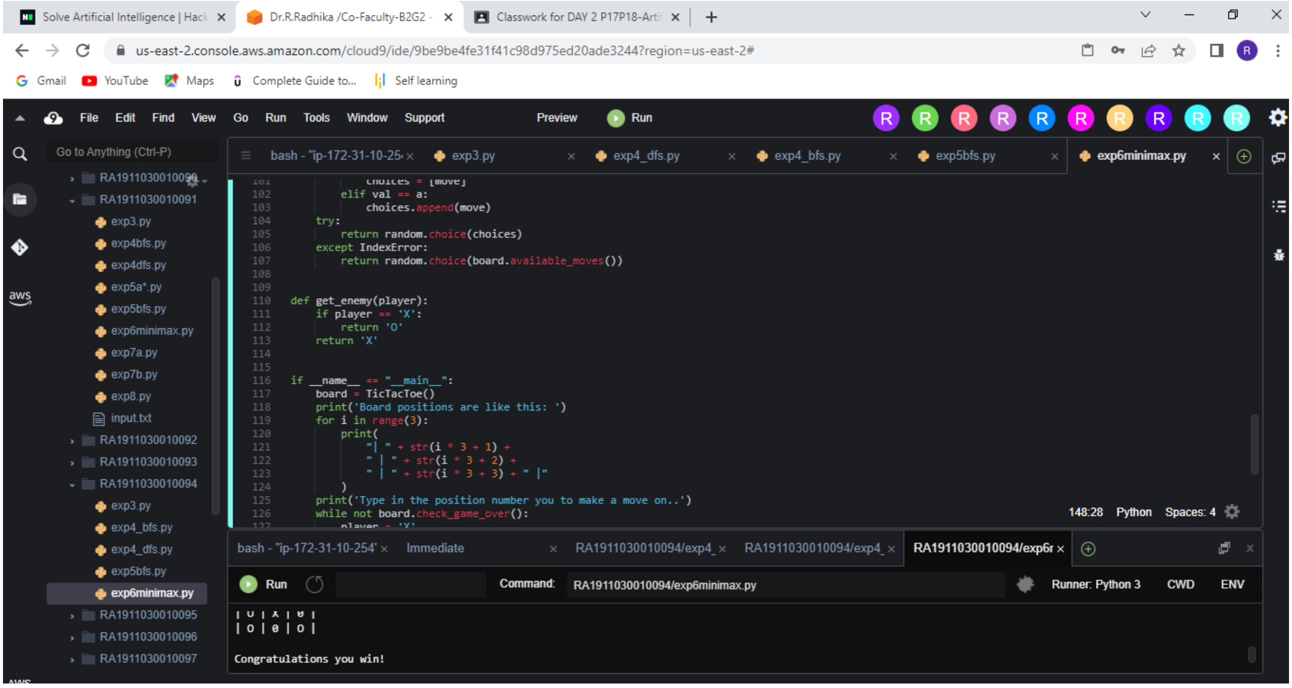


```
            choices = [move]
        elif val == a:
            choices.append(move)
    try:
        return random.choice(choices)
    except IndexError:
        return random.choice(board.available_moves())


def get_enemy(player):
    if player == 'X':
        return 'O'
    return 'X'


if __name__ == "__main__":
    board = TicTacToe()
    print('Board positions are like this: ')
    for i in range(3):
        print(
            "| " + str(i * 3 + 1) +
            " | " + str(i * 3 + 2) +
            " | " + str(i * 3 + 3) + " |"
        )
    print('Type in the position number you to make a move on..')
    while not board.check_game_over():
        player = 'X'
```

Command: RA1911030010094/exp6minimax.py

Runner: Python 3

```
| X | 0 | 0 |
| O | 0 | 0 |
| 0 | 0 | 0 |
Your Move: 
```



```
            choices = [move]
        elif val == a:
            choices.append(move)
    try:
        return random.choice(choices)
    except IndexError:
        return random.choice(board.available_moves())


def get_enemy(player):
    if player == 'X':
        return 'O'
    return 'X'


if __name__ == "__main__":
    board = TicTacToe()
    print('Board positions are like this: ')
    for i in range(3):
        print(
            "| " + str(i * 3 + 1) +
            " | " + str(i * 3 + 2) +
            " | " + str(i * 3 + 3) + " |"
        )
    print('Type in the position number you to make a move on..')
    while not board.check_game_over():
        player = 'X'
```

```
| O | X | 0 |
| O | 0 | O |

Congratulations you win!
```

Result:

Mini-Max algorithm has been successfully implemented using python.