KJSCE/IT/SY/SEM IV/HO-IAI/2022-23

# Batch: B2 Roll No.: 16010421073 Experiment No.: 4

## Aim of the Experiment: Implementation of Adversarial algorithm-Min-Max for Tic-Tac-Toe Game

**Program/ Steps:**

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 == 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): 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('\n You: X \n Computer: Y\nBoard positions are like this: ') for i in range(3):

print(

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

" | " + str(i \* 3 + 3) + " |"

)

print('Type position no. for your move') while not board.check\_game\_over():

player = 'X'

player\_move = int(input("Your Move: ")) - 1

if player\_move not in board.available\_moves(): print('Move not available!')

continue

board.make\_move(player\_move, player) board.print\_board()

print()

if board.check\_game\_over(): break

print('Ai 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 ("Congrats you win!") else:

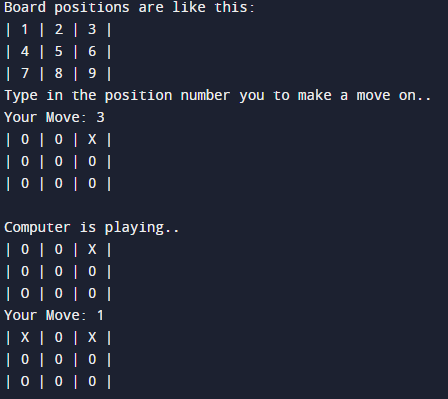
print('Computer Wins!')

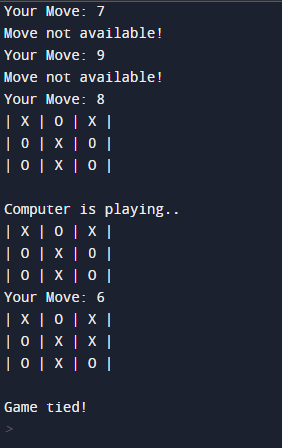
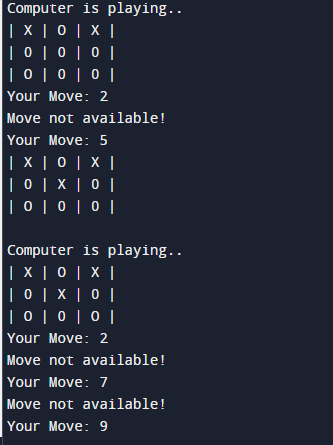
else:

print("Game tied!")

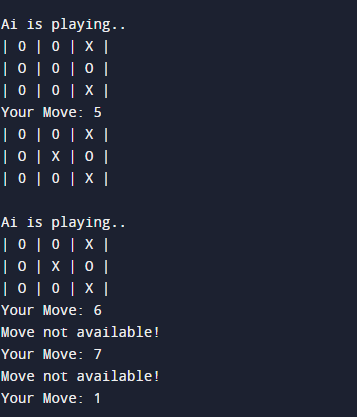
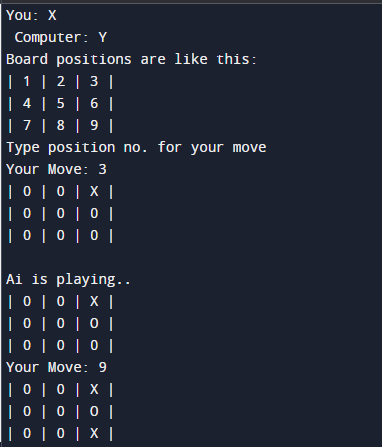
## Output/Result:

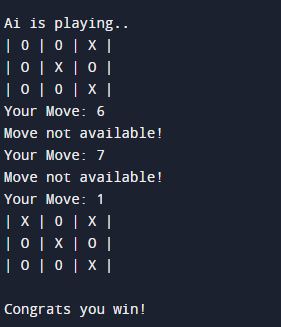
**Case 1: Game tied**



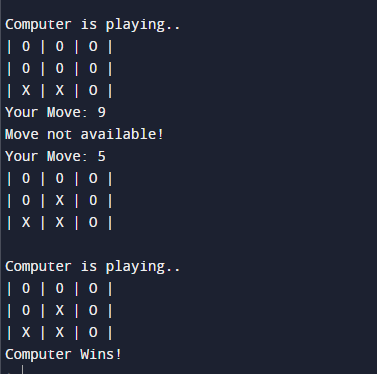


**Case 2: Human wins**





**Case 3: Ai wins**



## Post Lab Question-Answers:

1. **Game playing is often called as an**
2. **Non-adversial search**
3. **Adversial search**
4. **Sequential search**
5. **None of the above**

**Ans: b) Adversial search**

1. **What are the basic requirements or need of AI search methods in game playing?**
2. **Initial State of the game**
3. **Operators defining legal moves**
4. **Successor functions**
5. **Goal test**
6. **Path cost**

**Ans: All of the above**

# Outcomes:

**CO2: Analyze and formalize the problem (as a state space, graph, etc.) and select the appropriate search method and write the algorithm.**

# Conclusion (based on the Results and outcomes achieved):

## We have Learned about adversarial algorithm and successfully created a Tic-Tac-Toe game using Minmax algorithm and has 2 players –i)Human

**ii)Ai**

# References:

How to make your Tic Tac Toe game unbeatable by using the minimax algorithm: [https://www.freecodecamp.org/news/how-to-make-your-tic-tac-toe-game-unbeatable-by-using-the-minimax-](https://www.freecodecamp.org/news/how-to-make-your-tic-tac-toe-game-unbeatable-by-using-the-minimax-algorithm-) [algorithm-](https://www.freecodecamp.org/news/how-to-make-your-tic-tac-toe-game-unbeatable-by-using-the-minimax-algorithm-)

Stuart Russell and Peter Norvig, Artificial Intelligence: A Modern Approach, 2ndEdition, Pearson Publication

Elaine Rich, Kevin Knight, Artificial Intelligence, Tata McGraw Hill, 1999.

(A Constituent College of Somaiya Vidyavihar University)