EXPERIMENT 05

Name: Rohit Vilas Bhadane	Date:
Roll No: 04	Grade:
Class: TE Comps	Sign:

Aim: To study adversarial search (MIN MAX Algorithm).

Theory:

Adversarial Search:

Adversarial search is a search, where we examine the problem which arises when we try to plan ahead of the world and other agents are planning against us.There might be some situations where more than one agent is searching for the solution in the same search space, and this situation usually occurs in game playing.The environment with more than one agent is termed as multi-agent environment, in which each agent is an opponent of other agent and playing against each other. Each agent needs to consider the action of other agent and effect of that action on their performance. So, Searches in which two or more players with conflicting goals are trying to explore the same search space for the solution, are called adversarial searches, often known as Games. Games are modeled as a Search problem and heuristic evaluation function, and these are the two main factors which help to model and solve games in AI.

● **Perfect information:** A game with the perfect information is that in which agents can look into the complete board. Agents have all the information about the game, and they can see each other moves also. Examples are Chess, Checkers, Go, etc.

● **Imperfect information:** If in a game agents do not have all information about the game and not aware with what's going on, such type of games are called the game with imperfect information, such as tic-tac-toe, Battleship, blind, Bridge, etc.

● **Deterministic games:** Deterministic games are those games which follow a strict pattern and set of rules for the games, and there is no randomness associated with them. Examplesare chess, Checkers, Go, tic-tac-toe, etc.

● **Non-deterministic games:** Non-deterministic are those games which have various unpredictable events and has a factor of chance or luck. This factor of chance or luck is introduced by either dice or cards. These are random, and each action response is not fixed. Such games are also called as stochastic games. Example: Backgammon,
Monopoly, Poker, etc.

Formalization of the Problem:

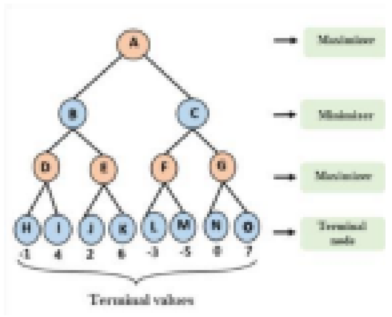
A game can be defined as a type of search in AI which can be formalized of the following elements:

- **Initial state:** It specifies how the game is set up at the start.
- **Player(s):** It specifies which player has moved in the state space.
- **Action(s):** It returns the set of legal moves in state space.
- **Result(s, a):** It is the transition model, which specifies the result of moves in the state space.
- **Terminal-Test(s):** Terminal test is true if the game is over, else it is false at any case. The state where the game ends is called terminal states.
- **Utility(s, p):** A utility function gives the final numeric value for a game that ends in terminal states s for player p. It is also called payoff function. For Chess, the outcomes are a win, loss, or draw and its payoff values are +1, 0, ½. And for tic-tac-toe, utility values are +1, -1, and 0.

MIN MAX 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. Mini-Max algorithm uses recursion to search through the game-tree. Min-Max algorithm is mostly used for game playing in AI. Such as Chess, Checkers, tic-tac-toe, go, and various towplayers game. This Algorithm computes the minimax decision for the current state. In this algorithm two players play the game, one is called MAX and other is called MIN. Both the players fight it as the opponent player gets the minimum benefit while they get the maximum benefit. Both Players of the game are opponent of each other, where MAX will select the maximized value and MIN will select the minimized value. The minimax algorithm performs a depth-first search algorithm for the exploration of the complete game tree. The minimax algorithm proceeds all the way down to the terminal node of the tree, then backtrack the tree as the recursion. The time complexity of Min-Max algorithm is O(b^m), where b is branching factor of the game-tree, and m is the maximum depth of the tree.Space complexity of Mini-max algorithm is also similar to DFS which is O(bm).

Advantages:



- A thorough assessment of the search space is performed.
- Decision making in AI is easily possible.
- New and smart machines are developed with this algorithm.

Disadvantages:

- Because of the huge branching factor, the process of reaching the goal is slower.
- Evaluation and search of all possible nodes and branches degrades the performance and efficiency of the engine.

● Both the players have too many choices to decide from.

● If there is a restriction of time and space, it is not possible to explore the entire tree.

Applications: ●
Decisionmaki
ng ● Game
Theory
● Two-player turn-based game

Algorithm: function minimax(node, depth,

```
maximizingPlayer)
is if depth ==0 or node is a terminal node then return
static evaluation of node
if MaximizingPlayer then // for Maximizer
Player maxEva= -infinity for each child of node do
eva= minimax(child, depth-1, false)
maxEva= max(maxEva,eva) //gives Maximum of the values
return maxEva
else // for Minimizer player
minEva= +infinity for each child of
node do eva= minimax(child,
depth-1, true)
minEva= min(minEva, eva) //gives minimum of the values
return minEva
```

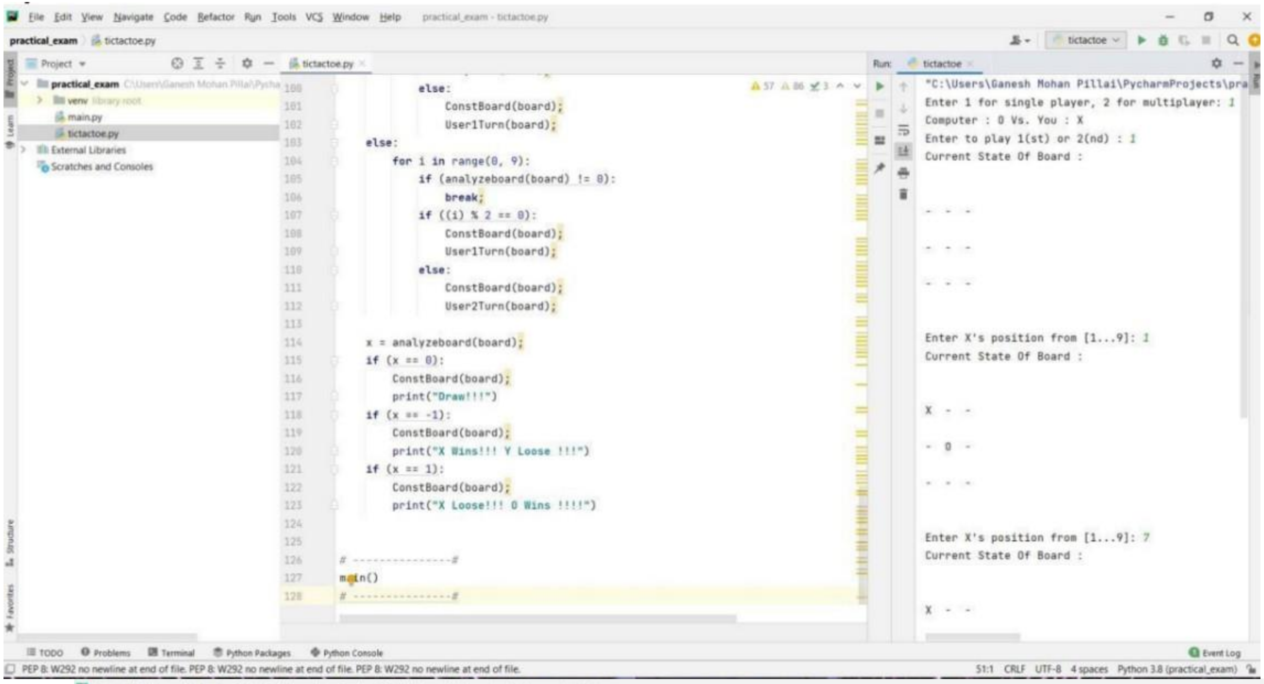
Code:

```
# This function is used to draw the board's current state every time the user turn
arrives. def ConstBoard(board): print("Current State Of Board : \n\n"); for i in
range(0, 9): if ((i > 0) and (i % 3) == 0):
print("\n");
if (board[i] == 0): print("-
", end=" ");
if (board[i] == 1): print("O
", end=" ");
if (board[i] == -1):
print("X ", end=" ");
print("\n\n");

# This function takes the user move as input and make the required changes on the
board.  def User1Turn(board):
pos = input("Enter X's position from [1...9]: ");
pos = int(pos); if (board[pos - 1] != 0):
print("Wrong Move!!!"); exit(0);
board[pos - 1] = -1;

def User2Turn(board):
```

```
pos = input("Enter O's position from [1...9]: ");
pos = int(pos); if (board[pos - 1] != 0): print("Wrong
Move!!!");
exit(0);
board[pos - 1] = 1; #
MinMax function.
def minimax(board,
player): x =
analyzeboard(boar
d); if (x != 0):
return (x * player);
pos = -1; value = 2;
for i in range(0, 9):
if (board[i] == 0): board[i] = player; score
= -minimax(board, (player * -1)); if
(score > value): value = score; pos =
i;
board[i] = 0;

if (pos == -1):
return 0;
return value;

# This function makes the computer's move using minmax
algorithm.  def CompTurn(board): pos = -1; value = 2; for i in
range(0, 9): if (board[i] == 0): board[i] = 1;
score = -minimax(board, -
1); board[i] = 0;
if (score >
value): value =
score; pos = i;
board[pos] = 1;

# This function is used to analyze a game.
def analyzeboard(board):
cb = [[0, 1, 2], [3, 4, 5], [6, 7, 8], [0, 3, 6], [1, 4, 7], [2, 5, 8], [0, 4, 8], [2, 4, 6]];

for i in range(0, 8): if (board[cb[i][0]] != 0 and
board[cb[i][0]] == board[cb[i][1]] and
board[cb[i][0]] == board[cb[i][2]]): return
board[cb[i][2]];
return 0;

# Main Function. def main(): choice = input("Enter 1 for
single  player, 2 for multiplayer: "); choice = int(choice);
```

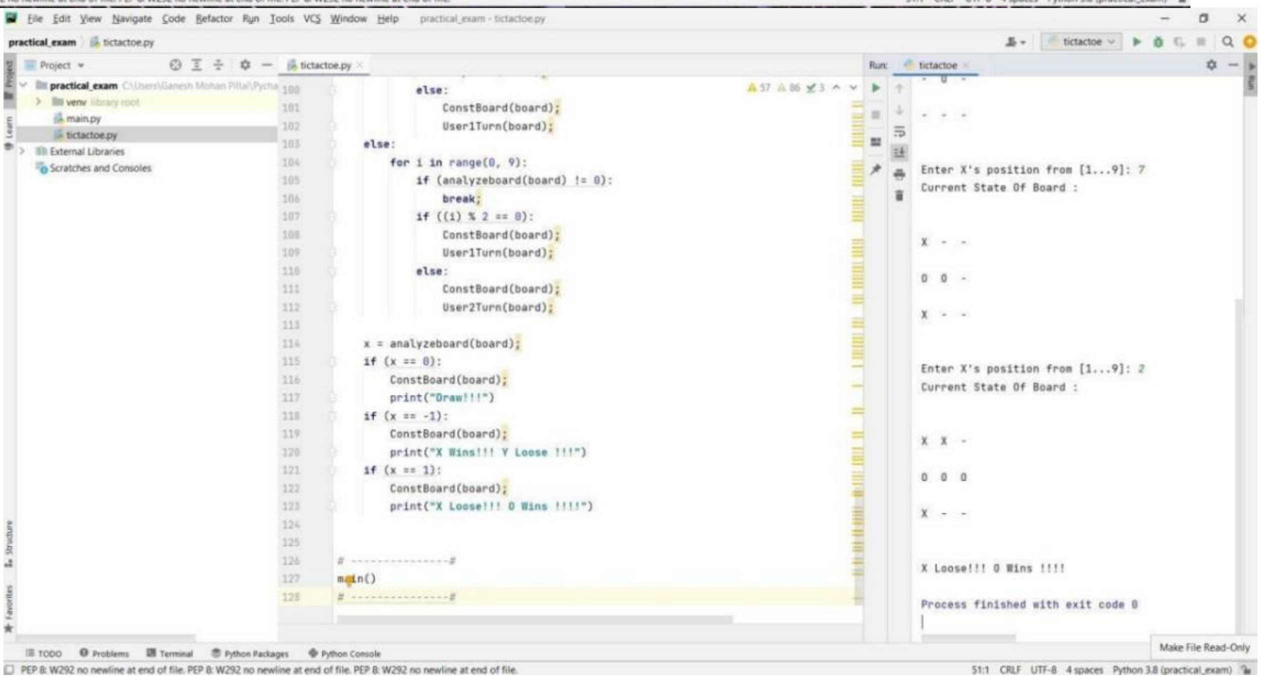
```
# The broad is considered in the form of a single dimentional
array.  # One player moves 1 and other move -1.
board = [0, 0, 0, 0, 0, 0, 0, 0, 0]; if (choice == 1):
print("Computer : O Vs. You : X"); player = input("Enter
to play 1(st) or 2(nd) : "); player = int(player); for i in
range(0, 9):
if (analyzeboard(board) != 0): break;
if ((i + player) % 2 == 0):
CompTurn(board);
else:
ConstBoard(board);
User1Turn(board); else:
for i in range(0, 9):
if (analyzeboard(board) != 0):
break; if ((i) % 2 == 0):
ConstBoard(board);
User1Turn(board);
else:
ConstBoard(board); User2Turn(board);

x = analyzeboard(board); if
(x == 0):
ConstBoard(board); print("Draw!!!")
if (x == -1):
ConstBoard(board); print("X
Wins!!! Y Loose
!!!")  if
(x == 1):
ConstBoard(board); print("X Loose!!!
O Wins !!!!")

#--------------- # main()
# ---------------#
```

Output:



```
            else:
                ConstBoard(board);
                User1Turn(board);
        else:
            for i in range(0, 9):
                if (analyzeboard(board) != 0):
                    break;
                if ((i) % 2 == 0):
                    ConstBoard(board);
                    User1Turn(board);
                else:
                    ConstBoard(board);
                    User2Turn(board);

        x = analyzeboard(board);
        if (x == 0):
            ConstBoard(board);
            print("Draw!!!")
        if (x == -1):
            ConstBoard(board);
            print("X Wins!!! Y Loose !!!")
        if (x == 1):
            ConstBoard(board);
            print("X Loose!!! O Wins !!!!")


# ---------------#
main()
# ---------------#
```

```
"C:\Users\Ganesh Mohan Pillai\PycharmProjects\pra
Enter 1 for single player, 2 for multiplayer: 1
Computer : O Vs. You : X
Enter to play 1(st) or 2(nd) : 1
Current State Of Board :


- - -

- - -

- - -


Enter X's position from [1...9]: 1
Current State Of Board :


X - -

- O -

- - -


Enter X's position from [1...9]: 7
Current State Of Board :


X - -
```



```
            else:
                ConstBoard(board);
                User1Turn(board);
        else:
            for i in range(0, 9):
                if (analyzeboard(board) != 0):
                    break;
                if ((i) % 2 == 0):
                    ConstBoard(board);
                    User1Turn(board);
                else:
                    ConstBoard(board);
                    User2Turn(board);

        x = analyzeboard(board);
        if (x == 0):
            ConstBoard(board);
            print("Draw!!!")
        if (x == -1):
            ConstBoard(board);
            print("X Wins!!! Y Loose !!!")
        if (x == 1):
            ConstBoard(board);
            print("X Loose!!! O Wins !!!!")


# ---------------#
main()
# ---------------#
```

```
- - -


Enter X's position from [1...9]: 7
Current State Of Board :


X - -

O O -

X - -


Enter X's position from [1...9]: 2
Current State Of Board :


X X -

O O O

X - -


X Loose!!! O Wins !!!!

Process finished with exit code 0
```

Conclusion:

We have successfully implemented adversarial search in python.