Vivek Poojari
2019130050
TE COMPS
AIML Lab

Experiment 3

Aim: To implement tic tac toe game in python

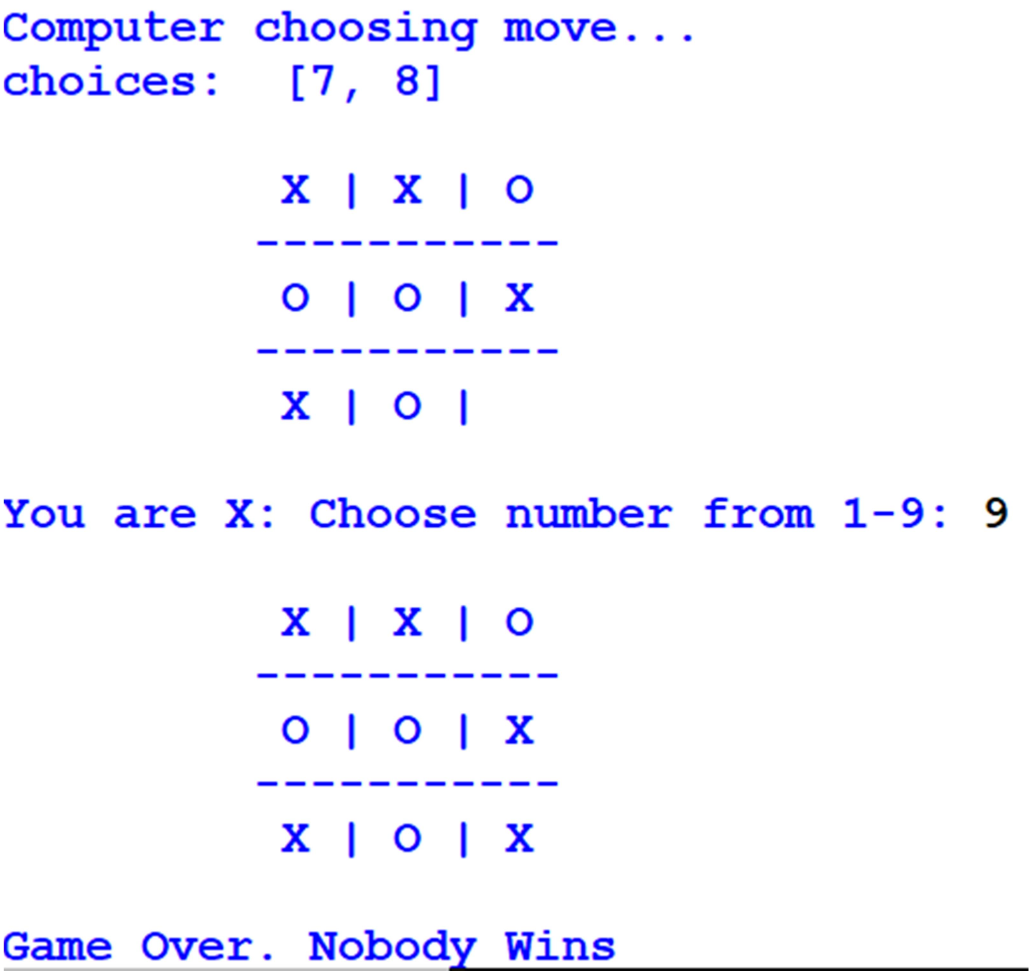
Code:

```
import random

class TicTacToe:

    def __init__(self):
        """Initialize with empty board"""
        self.board = [" ", " ", " ",
                      " ", " ", " ",
                      " ", " ", " "]

    def show(self):
        """Format and print board"""
        print("""
          {} | {} | {}
         -----------
          {} | {} | {}
         -----------
          {} | {} | {}
        """.format(*self.board))

    def clearBoard(self):
        self.board = [" ", " ", " ",
                      " ", " ", " ",
                      " ", " ", " "]

    def whoWon(self):
        if self.checkWin() == "X":
            return "X"
        elif self.checkWin() == "O":
            return "O"
        elif self.gameOver() == True:
            return "Nobody"

    def availableMoves(self):
        """Return empty spaces on the board"""
        moves = []
        for i in range(0, len(self.board)):
            if self.board[i] == " ":
                moves.append(i)
        return moves

    def getMoves(self, player):
        """Get all moves made by a given player"""
        moves = []
        for i in range(0, len(self.board)):
            if self.board[i] == player:
                moves.append(i)
        return moves

    def makeMove(self, position, player):
        """Make a move on the board"""
        self.board[position] = player

    def checkWin(self):
        """Return the player that wins the game"""
        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])

        for player in ("X", "O"):
            positions = self.getMoves(player)
            for combo in combos:
                win = True
```

```
         for pos in combo:
            if pos not in positions:
               win = False
         if win:
            return player

def gameOver(self):
   """Return True if X wins, O wins, or draw, else return False"""
   if self.checkWin() != None:
      return True
   for i in self.board:
      if i == " ":
         return False
   return True

def astar(self, node, depth, player):
   """
   Recursively analyze every possible game state and choose
   the best move location.
   node - the board
   depth - how far down the tree to look
   player - what player to analyze best move for (currently setup up ONLY for "O")
   """
   if depth == 0 or node.gameOver():
      if node.checkWin() == "X":
         return 0
      elif node.checkWin() == "O":
         return 100
      else:
         return 50

   if player == "O":
      bestValue = 0
      for move in node.availableMoves():
         node.makeMove(move, player)
         moveValue = self.astar(node, depth-1, changePlayer(player))
         node.makeMove(move, " ")
```

```
                bestValue = max(bestValue, moveValue)
            return bestValue

        if player == "X":
            bestValue = 100
            for move in node.availableMoves():
                node.makeMove(move, player)
                moveValue = self.astar(node, depth-1, changePlayer(player))
                node.makeMove(move, " ")
                bestValue = min(bestValue, moveValue)
            return bestValue

def changePlayer(player):
    """Returns the opposite player given any player"""
    if player == "X":
        return "O"
    else:
        return "X"

def make_best_move(board, depth, player):
    """
    Controllor function to initialize mm and keep track of optimal move choices
    board - what board to calculate best move for
    depth - how far down the tree to go
    player - who to calculate best move for (Works ONLY for "O" right now)
    """
    neutralValue = 50
    choices = []
    for move in board.availableMoves():
        board.makeMove(move, player)
        moveValue = board.astar(board, depth-1, changePlayer(player))
        board.makeMove(move, " ")

        if moveValue > neutralValue:
            choices = [move]
            break
        elif moveValue == neutralValue:
            choices.append(move)
    print("choices: ", choices)

    if len(choices) > 0:
        return random.choice(choices)
    else:
        return random.choice(board.availableMoves())




#Actual game
if __name__ == '__main__':
    game = TicTacToe()
    game.show()

    while game.gameOver() == False:
        person_move = int(input("You are X: Choose number from 1-9: "))
        game.makeMove(person_move-1, "X")
        game.show()

        if game.gameOver() == True:
            break

        print("Computer choosing move...")
        ai_move = make_best_move(game, -1, "O")
        game.makeMove(ai_move, "O")
        game.show()

    print("Game Over. " + game.whoWon() + " Wins")
```

Output:



```
Computer choosing move...
choices:  [7, 8]

          X | X | O
         -----------
          O | O | X
         -----------
          X | O |

You are X: Choose number from 1-9: 9

          X | X | O
         -----------
          O | O | X
         -----------
          X | O | X

Game Over. Nobody Wins
```

Conclusion:

In this experiment we learnt how to implement the tic tac toe game in python using the A* algorithm. A major drawback of the A* algorithm is its space and time complexity. It takes a large amount of space to store all possible paths and a lot of time to find them.