|  |  |
| --- | --- |
| **Name** | Manish Shashikant Jadhav |
| **UID no.** | 2023301005 |

|  |  |
| --- | --- |
| **Experiment 4** | |
| **AIM :** | Implement the problem using the Informed searching technique min-max algorithm . Analyze the algorithm with respect to Completeness, Optimality, time and space Complexity  a)     Tic Tac Toe |
| **CODE:** | *import* math  def print\_board(*board*):  *for* i *in* range(3):          print(" | ".join(*board*[i\*3:(i+1)\*3]))  *if* i < 2:              print("---------")  def empty\_cells(*board*):  *return* [i *for* i, cell *in* enumerate(*board*) *if* cell == " "]  def is\_winner(*board*, *player*):      winning\_combinations = [          [0, 1, 2], [3, 4, 5], [6, 7, 8],  *# Rows*          [0, 3, 6], [1, 4, 7], [2, 5, 8],  *# Columns*          [0, 4, 8], [2, 4, 6]  *# Diagonals*      ]  *return* any(all(*board*[i] == *player* *for* i *in* combo) *for* combo *in* winning\_combinations)  def game\_over(*board*):  *return* is\_winner(*board*, "X") or is\_winner(*board*, "O") or len(empty\_cells(*board*)) == 0  def minimax(*board*, *depth*, *is\_maximizing*):  *if* is\_winner(*board*, "X"):  *return* -1  *if* is\_winner(*board*, "O"):  *return* 1  *if* len(empty\_cells(*board*)) == 0:  *return* 0  *if* *is\_maximizing*:          best\_score = -math.inf  *for* move *in* empty\_cells(*board*):  *board*[move] = "O"              score = minimax(*board*, *depth* + 1, False)  *board*[move] = " "              best\_score = max(score, best\_score)  *return* best\_score  *else*:          best\_score = math.inf  *for* move *in* empty\_cells(*board*):  *board*[move] = "X"              score = minimax(*board*, *depth* + 1, True)  *board*[move] = " "              best\_score = min(score, best\_score)  *return* best\_score  def get\_best\_move(*board*):      best\_score = -math.inf      best\_move = None  *for* move *in* empty\_cells(*board*):  *board*[move] = "O"          score = minimax(*board*, 0, False)  *board*[move] = " "  *if* score > best\_score:              best\_score = score              best\_move = move  *return* best\_move  def get\_player\_move(*board*, *player*):  *while* True:  *try*:              move = int(input(f"Player {*player*}, enter your move (0-8): "))  *if* move not in empty\_cells(*board*):  *raise* ValueError  *return* move  *except* ValueError:              print("Invalid move. Try again.")  def play\_game(*mode*):      board = [" " *for* \_ *in* range(9)]      current\_player = "X"    *if* *mode* == "1":          print("You are X, AI is O")  *else*:          print("Player 1: X, Player 2: O")        print\_board(board)  *while* not game\_over(board):  *if* *mode* == "1" and current\_player == "O":              print("AI is making a move...")              move = get\_best\_move(board)  *else*:              move = get\_player\_move(board, current\_player)            board[move] = current\_player          print\_board(board)  *if* game\_over(board):  *break*          current\_player = "O" *if* current\_player == "X" *else* "X"  *if* is\_winner(board, "X"):          print("X wins!")  *elif* is\_winner(board, "O"):          print("O wins!")  *else*:          print("It's a tie!")  def main():      print("Welcome to Tic Tac Toe!")  *while* True:          mode = input("Enter 1 for single player (vs AI) or 2 for two players: ")  *if* mode in ["1", "2"]:  *break*          print("Invalid input. Please enter 1 or 2.")        play\_game(mode)  *if* \_\_name\_\_ == "\_\_main\_\_":      main() |
| **OUTPUT:** | 1. **Two Player Game:**      1. **Single Player Game AI Win:**      1. **Game Tie:** |
| **Analysis of Algorithm** | 1. **Completeness:** Yes, the algorithm is complete and will always find a solution (win, lose, or draw). 2. **Optimality:** Yes, the algorithm is optimal for both players when they play optimally. 3. **Time Complexity:** O(b^d), which is O(9!) for Tic-Tac-Toe, equivalent to O(362,880) in the worst case. 4. **Space Complexity:** O(d), which is O(9) for Tic-Tac-Toe, meaning the space complexity is constant and manageable. |
| **CONCLUSION:** | Hence by completing this experiment I came to know about Informed searching technique min-max algorithm . |