UNIVERSITY INSTITUTE of COMPUTING
Asia's Fastest Growing University

NAAC GRADE A+ ACCREDITED UNIVERSITY

Student Name: LEVRY NAKA HENRI-MICHEL **UID: 25MCI10388**
Branch: MCA [AI/ML] **Section/Group:6B**
Semester: 1st **Date of Performance: 31-10-2025**

Subject Name: AI **Subject code: 25CAT-613**

MINI PROJECT REPORT

Title: Artificial Intelligence Mini Project – Tic Tac Toe using Minimax Algorithm with Pygame

ABSTRACT:

This project presents an interactive implementation of the Tic-Tac-Toe game using Artificial Intelligence concepts. The AI opponent is developed using the Minimax algorithm, a popular decision-making technique in game theory. The game allows players to choose between Player vs Player (PVP) and Player vs AI (Minimax) modes. The interface is designed using Python's Pygame library, providing a simple and engaging graphical user experience. The AI ensures optimal gameplay, making it unbeatable when playing against a human. The project demonstrates the practical application of AI algorithms in real-time games and decision-making systems.

1. Introduction

This mini project focuses on implementing the Minimax Algorithm in a Tic Tac Toe game using Python and Pygame.

The project demonstrates how Artificial Intelligence can be applied to make optimal decisions in turn-based games.

2. Objectives

- To understand and implement the Minimax algorithm.
- To apply AI techniques in a real-time environment using Pygame.
- To create an interactive two-player and AI-based game.

The main objective of this mini project is to:

- Implement an intelligent Tic-Tac-Toe game where the computer can make optimal decisions.
- Provide an option for both Player vs Player and Player vs AI modes.
- Utilize the Minimax algorithm to simulate human-like strategic thinking.
- Develop a visually appealing interface using Pygame.

3. Tools and Technologies

Component	Description
Programming Language	Python 3.13.7
Libraries Used	Pygame
Algorithm	Minimax Algorithm
Platform	Cross-platform (Windows/Linux/macOS)
IDE (Optional)	VS Code / PyCharm / IDLE

4. Methodology

The system uses Pygame for graphical rendering and the Minimax algorithm for AI
decision-making. The board is represented as a 3x3 matrix, where the AI and player take turns until a terminal state (win/draw) is reached.

5. Algorithm Design and Flowchart

5.1 Minimax Algorithm (Step-by-Step Explanation)

The **Minimax algorithm** is a **recursive search algorithm** used in decision-making and game theory.
Its goal is to minimize the possible loss while maximizing the potential gain.
It is commonly used in **two-player games** like Tic Tac Toe, Chess, and Checkers.

Step-by-Step Working:

1. **Input:**
   - Current board state
   - Current player (Maximizing or Minimizing)
2. **Base Conditions:**
   - If a player has won → return a positive or negative score.
   - If the board is full (draw) → return 0.
3. **For Maximizing Player (AI – 'O'):**
   - Initialize best score = -∞
   - For every empty cell:
     - Place the move.
     - Call Minimax recursively for the Minimizing player ('X').
     - Undo the move.
     - Update best score = max(best score, returned score).
4. **For Minimizing Player (Human – 'X'):**
   - Initialize best score = +∞
   - For every empty cell:
     - Place the move.
     - Call Minimax recursively for the Maximizing player ('O').
     - Undo the move.
     - Update best score = min(best score, returned score).

5. **Return the best score** to the upper recursive call.
6.
7. **The AI chooses the move with the highest score** among all possible moves.

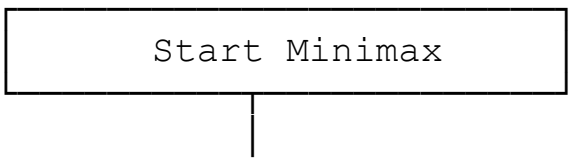
⚙ **Pseudocode:**

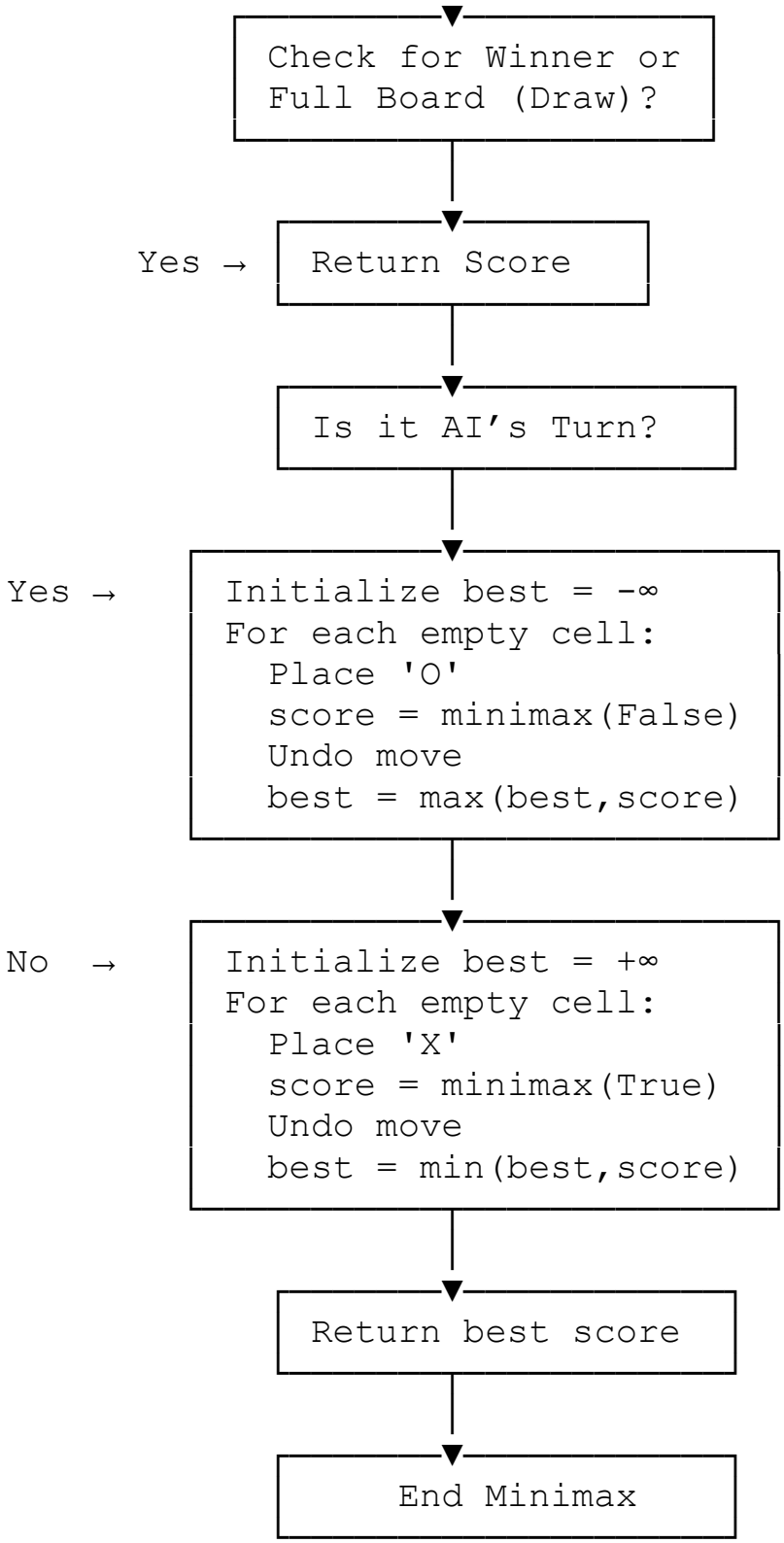
```
function minimax(board, depth, isMaximizing):
    if check_winner('O'):
        return +1
    if check_winner('X'):
        return -1
    if board is full:
        return 0

    if isMaximizing:
        best = -infinity
        for each empty cell:
            place 'O'
            score = minimax(board, depth + 1, False)
            undo move
            best = max(best, score)
        return best
    else:
        best = +infinity
        for each empty cell:
            place 'X'
            score = minimax(board, depth + 1, True)
            undo move
            best = min(best, score)
        return best
```

5.2 Flowchart of the Minimax Algorithm

Below is a text-based flow representation (for your printed report, you can include this diagrammatically):




Check for Winner or Full Board (Draw)?
Yes →
Return Score
Is it AI's Turn?
Yes →
Initialize best = -∞
For each empty cell:
Place 'O'
score = minimax(False)
Undo move
best = max(best,score)
No →
Initialize best = +∞
For each empty cell:
Place 'X'
score = minimax(True)
Undo move
best = min(best,score)
Return best score
End Minimax

6. Implementation

The game offers two modes: Player vs Player and Player vs AI.

The AI uses Minimax to determine its optimal move. The board updates dynamically using Pygame visuals, and a scoreboard tracks wins and draws.
This mini project is a Tic Tac Toe game built using Python, Pygame, and the Minimax algorithm.
It allows the user to:

- Play against another human (2-Player Mode), or
- Play against an AI opponent that uses the Minimax algorithm to make optimal decisions.

The AI aims to maximize its winning chances and minimize losses, ensuring it never loses if played correctly.

6.2 System Architecture

The system is composed of four main modules:

1. Game Interface (Pygame)
    - Responsible for drawing the board, displaying moves, and handling player inputs.
    - Includes event handling for mouse clicks and key presses.
2. Game Logic
    - Maintains the game state in a 2D list (`board`).
    - Checks for valid moves, wins, and draws.
3. Minimax Algorithm (AI Engine)
    - Recursively evaluates all possible future moves.
    - Chooses the move that maximizes the AI's advantage and minimizes the opponent's advantage.
4. Game Mode Controller
    - Allows the player to select between "Play vs AI" or "Play vs Player" modes.
    - Handles game restarts and score tracking.

6.3 How Minimax Works in the Game

The Minimax algorithm is used by the AI to decide its next move.
It works as follows:

1. Generate all possible moves for the AI (player 'O').

2. For each move, simulate the game:
    - If the move leads to a win, it gets a positive score (+1).
    - If the move leads to a loss, it gets a negative score (-1).
    - If it results in a draw, the score is 0.
3. The AI then chooses the move with the maximum score (best possible outcome).

Meanwhile, during simulation, the human player ('X') is treated as a minimizing player, trying to lower the score.

6.4 FLOW OF EXECUTION

1. Game starts → user chooses mode (vs AI or vs Player).
2. The Pygame window displays a 3x3 grid.
3. Player 1 (X) makes a move by clicking a cell.
4. If vs AI mode:
    - The AI uses Minimax to decide its best move.
    - The move is updated automatically on the board.
5. The game checks for a winner or a draw.
6. The result (X Wins, O Wins, or Draw) is displayed.
7. The player can restart or go back to the menu.

7. Results and Discussion

- The AI plays optimally and never loses.
- The interface is interactive and user-friendly.
- The algorithm performs efficiently for 3x3 Tic Tac Toe.

8. Limitations

- Works only for 3x3 board sizes.
- No learning capability.
- Simple graphics.

9. GitHub Link

michellevry18-source/TIC-TAC-TOE: Project

10. Future Enhancements

- Add Alpha-Beta pruning for faster computation.
- Support larger grids (4x4, 5x5).
- Include animations and sound effects.

11. Conclusion

This project demonstrates the practical use of Artificial Intelligence in game strategy.
Using the Minimax algorithm, we successfully built a smart Tic Tac Toe game where the AI makes optimal decisions.

12. References

1. Russell, S., & Norvig, P. (2020). Artificial Intelligence:
A Modern Approach. 2. Pygame Documentation –
https://www.pygame.org/docs/
3. GeeksforGeeks – Minimax Algorithm in Game Theory.