Tic Tac Toe

Presented by

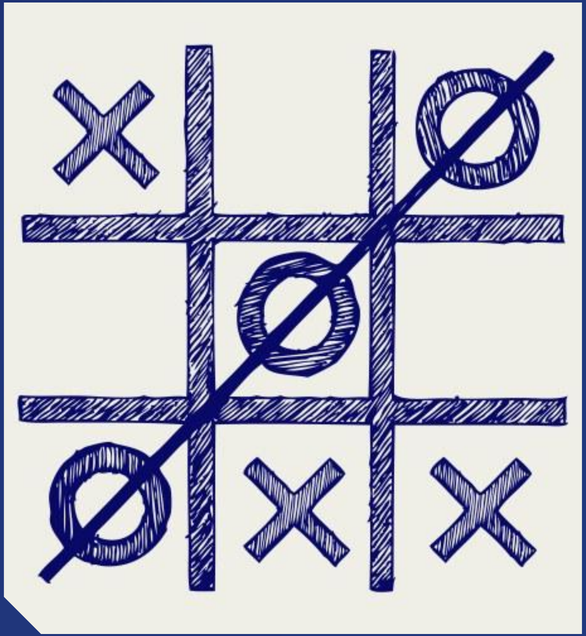
Abdullah Al Mamun
ID: 22101158
Anika Nawer Nabila
ID: 22101152

Presented to

Bidita Sarkar Diba
Lecturer
Department of CSE
University of Asia Pacific

Problem Statement :

The goal of this project is to create a **Tic Tac Toe game** with an **unbeatable AI** using the **Minimax algorithm** optimized by **Alpha-Beta pruning**. The AI should make optimal moves while minimizing computation time. The game should support multiple modes: **Human vs Computer** and **Computer vs Computer**, and provide a modern, interactive GUI using Python's Tkinter library.



Game Modes:

- Human vs. Computer
- Computer vs. Computer

Game Rules and Representation

Rules:

- The game is played on a **3×3 grid**.
- Player X always goes first, followed by Player O.
- Players alternate turns, placing their symbol (X or O) in an empty cell.
- The first player to align **three symbols horizontally, vertically, or diagonally wins**.
- If all cells are filled without a winner, the game ends in a **draw**.

Board Representation:

The board is represented as a **2D list** in Python:

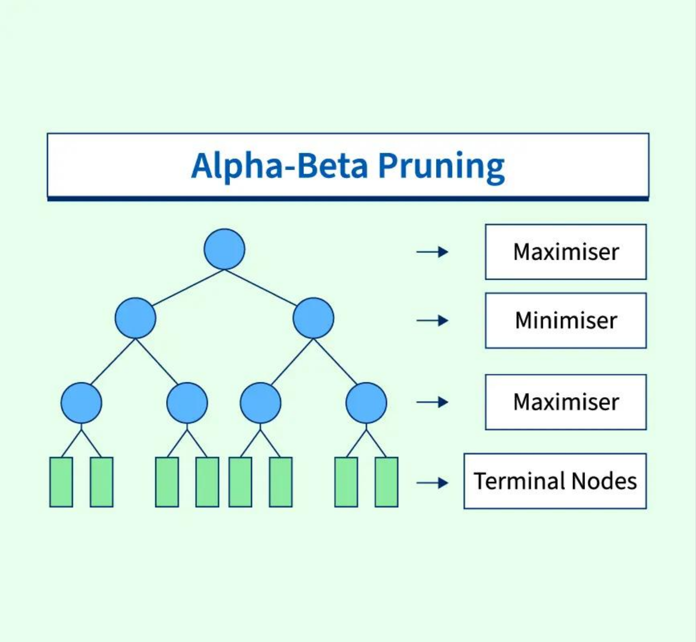
Minimax Algorithm

The Minimax Algorithm is widely used in game theory for decision-making. It
involves evaluating every possible move to determine the optimal strategy for
the AI.

Players:

- **Maximizer (X)**: Aims to maximize the score (win).
- **Minimizer (O)**: Aims to minimize the score (prevent loss).

Alpha Beta Pruning



Key Values:

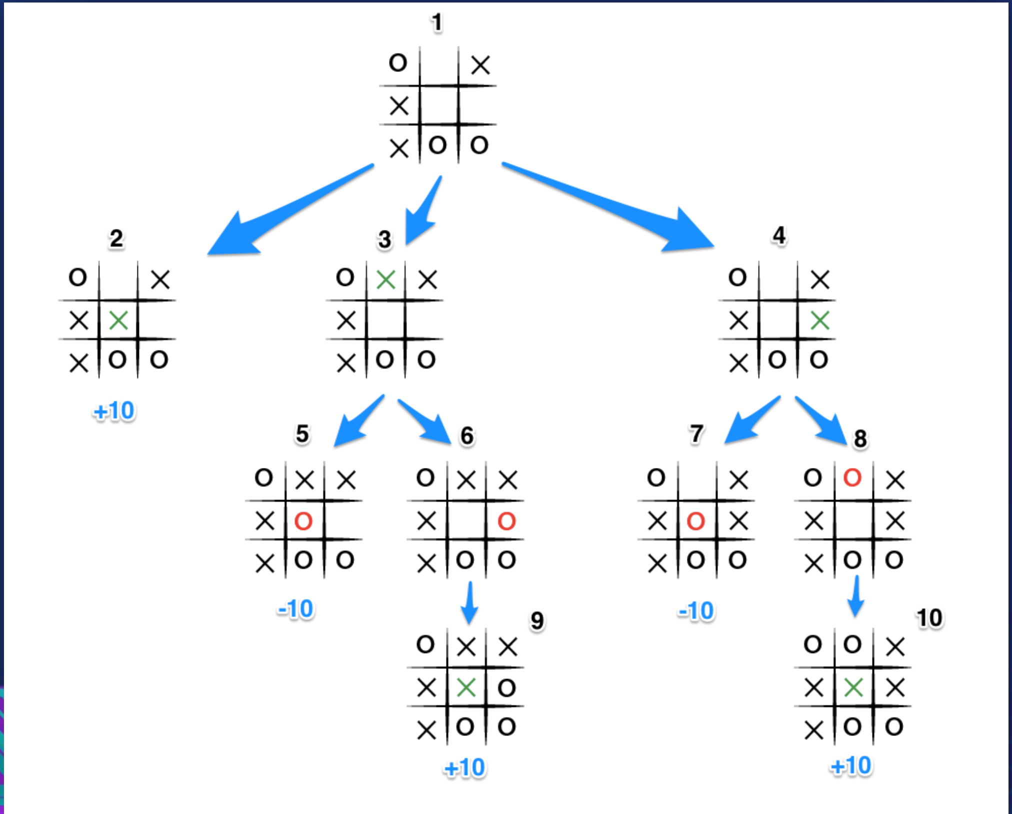
- **Alpha**: Best score Maximizer can guarantee.
- **Beta**: Best score Minimizer can guarantee.

Effect:

- Avoids unnecessary calculations.
- Enables deeper exploration of the game tree.

Outcome: Faster and more efficient decision-making.

Minimax Algorithm



Minimax Execution (Step-by-Step)

State 1: X’s turn → generates States 2, 3, 4

State 2: End state → pushes **+10**

State 3: Generates States 5 & 6

State 5 → pushes **−10**

State 6 → leads to win → pushes **+10**

O picks min(−10, +10) = −10

State 4: Generates States 7 & 8

State 7 → pushes **−10**

State 8 → leads to win → pushes **+10**

O picks min(−10, +10) = −10

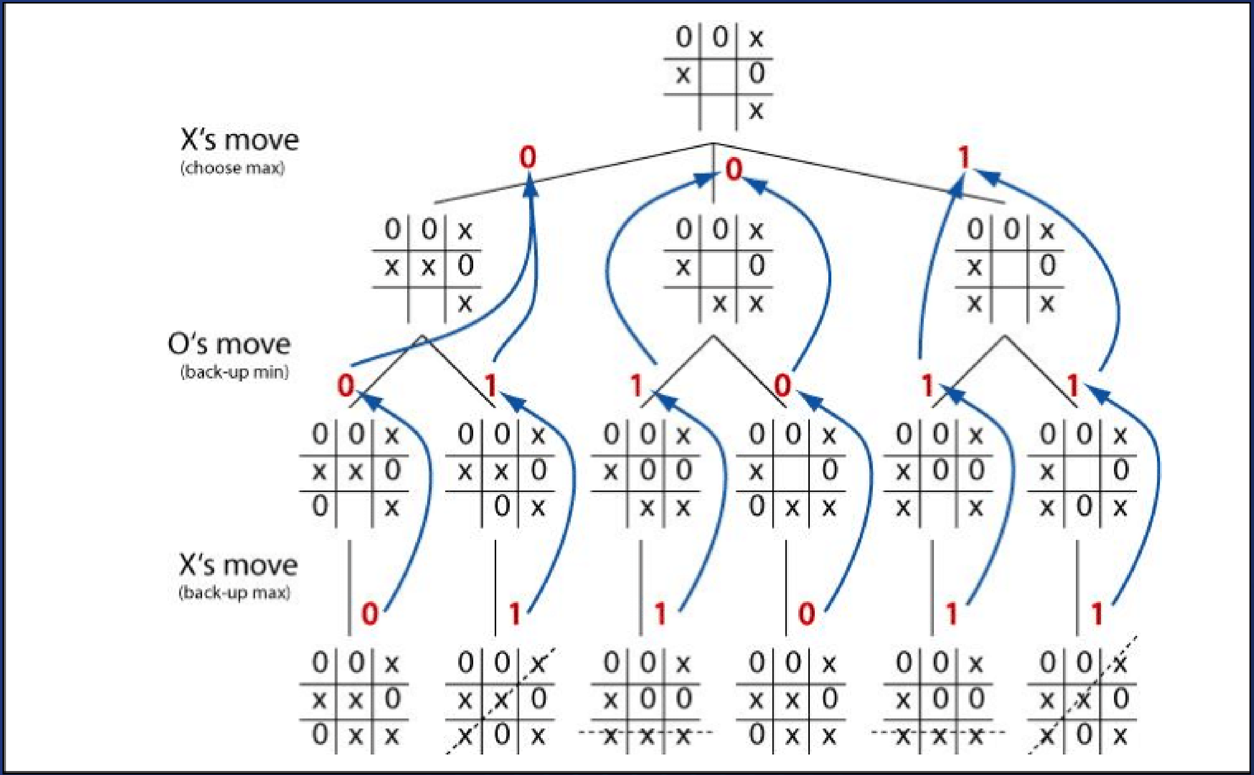
Backtrack to State 1:

Scores → [+10, −10, −10]

X picks max(+10, −10, −10) = +10 ✅

Best Move → **State 2**

Alpha Beta Pruning



Important Links

Source Code link

Report link

Proof of Alpha-Beta Pruning Efficiency

```
# Partial Board Setup
test_board = [
    ['X','O',' '],
    [' ','X',' '],
    [' ',' ',' ']
]

# Run Without Pruning
minimax_counter = 0
minimax_proof(test_board, depth=6,
              alpha=-inf, beta=inf,
              maximizing_player=True,
              player_symbol=player(test_board),
              use_pruning=False)
nodes_minimax = minimax_counter

# Run With Alpha-Beta Pruning
minimax_counter = 0
minimax_proof(test_board, depth=6,
              alpha=-inf, beta=inf,
              maximizing_player=True,
              player_symbol=player(test_board),
              use_pruning=True)
nodes_alpha_beta = minimax_counter

print("Pure Minimax:", nodes_minimax)
print("Alpha-Beta:", nodes_alpha_beta)
```

Explanation:

First, Minimax was executed **without pruning** → explored **all nodes**.

Then, Minimax was executed **with Alpha-Beta Pruning** → explored **fewer nodes**.

Comparing the **node counts proves** that Alpha-Beta Pruning is **faster and more efficient**.

SOURCE CODE

Comparison and Findings

Feature	Minimax	Alpha-Beta Pruning
Decision Quality	Optimal	Optimal
Computation	Explores all nodes	Skips unnecessary branches
Time Complexity	$O(b^d)$	$O(b^{\frac{d}{2}})$ on average
Memory Usage	Higher	Lower
Gameplay Result	Same	Same

Findings:

- Both Minimax and Alpha-Beta pruning produce the same **optimal decisions**.
- Alpha-Beta pruning **reduces computation** and improves efficiency, especially useful in larger games.

Conclusion

Building a Tic Tac Toe game with the Minimax Algorithm and Alpha-Beta Pruning is a great way to learn about AI and game development. Alpha-Beta Pruning helps make the AI more efficient, saving time while still playing well

Thank You