CSCI6649 – Term Project Mid-Status Report

Project Title: *Solving Games with AI: Game Tree and Minimax Algorithm Implementation*
Team Members: Parishad Ajallooeian (00867091),
Jigneshkumar Makawana (00890567)
Submission Date: April 18, 2025

---

1. Research Question

How can two-player, turn-based games be represented using game trees, and how can the Minimax algorithm enhanced with Alpha-Beta pruning be used to develop an AI that plays optimally in competitive games such as Tic-Tac-Toe?

---

2. Introduction

To apply AI in solving games, we use a concept called a game tree, followed by the minimax algorithm. In a game tree, each node represents a possible state of the game, like how states are represented in planning problems. The structure differs slightly in that the nodes are organized into levels based on which player's turn it is.

The topmost node, known as the root, represents the starting state of the game, for example, an empty tic-tac-toe board. The next level contains nodes that represent all the possible moves the first player can make. These are known as the children of the root node.

Each of those children then branches out into new children that show how the opponent might respond. This pattern continues, with each level alternating between players, until the tree reaches terminal nodes—states where the game ends either in a win for one of the players or in a draw when the board is full.

---

3. Related Literature

- **GeeksforGeeks: Minimax Algorithm**: Explains the fundamentals of Minimax in adversarial games.

- **Brilliant: Alpha-Beta Pruning**: Describes optimization techniques to improve Minimax performance.

Approach: Game Setup

- Use a 3x3 grid to represent the board.
- Players take turns (X and O).

Build Game Tree

- Create a tree of all possible moves.
- Each node = one game state.

Apply Minimax

- AI checks all possible moves and picks the best one.
- Win = +1, Loss = -1, Draw = 0.

Add Alpha-Beta Pruning

- Speeds up the Minimax by skipping moves that won't help.

Make AI Move

- When it's AI's turn, use Minimax to choose the best move.

Test and Improve

- Make sure AI never loses.
- Compare speed with and without pruning.

How it Work: The algorithm search, recursively, the best move that leads the *Max* player to win or not lose (draw). It considers the current state of the game and the available moves at that state, then for each valid move it plays (alternating *min* and *max*) until it finds a terminal state (win, draw or lose).

Achievements: minimax(state, depth, player)

if (player = max) then
best = [null, -infinity]

```
    else
        best = [null, +infinity]
    if (depth = 0 or gameover) then
        score = evaluate this state for player
        return [null, score]
    for each valid move m for player in state s do
      execute move m on s
      [move, score] = minimax(s, depth - 1, -player)
        undo move m on s
      if (player = max) then
          if score > best.score then best = [move, score]
      else
        if score < best.score then best = [move, score]
 return best
end
```

Challenges:

Detecting and Handling Draws

Problem:
The AI sometimes fails to detect a drawing and continues evaluating further moves.
Solution:
Enhance your `game over` condition to explicitly check for draw conditions (i.e., the board is full, and no winner exists).

Optimal Depth Limiting

- **Problem:**
  The algorithm evaluates the entire tree even when early outcomes are obvious, leading to unnecessary computation.
- **Solution:**
  Introduce a dynamic depth limit based on the remaining number of empty cells.

5. GitHub Repository

jigneshParishad/game-tree-minimax

(*Collaborator added: shivanjali.khare@gmail.com*)