Title: Noughts and Crosses with Alpha-Beta Pruning

A PROJECT REPORT

by

VISHAL SINGH (202410116100251)

VINAYAK CHAUHAN (202410116100245)

VIBHOR TYAGI (202410116100240)

YATHARTH TYAGI (202410116100254)

Session:2024-2025(II Semester)

Under the supervision of

Ms. Komal Salgotra

KIET Group of Institutions, Delhi-NCR, Ghaziabad

Department Of Computer Applications

KIET GROUP OF INSTITUTIONS, DELHI-NCR, GHAZIABAD-201206

(2024-2025)

2. Abstract

This project focuses on implementing an AI-powered version of Tic-Tac-Toe using the Minimax algorithm with Alpha-Beta Pruning. The objective is to develop an efficient and intelligent game opponent that minimizes computation while maintaining optimal gameplay. The AI uses the Minimax decision-making strategy enhanced with Alpha-Beta pruning to reduce unnecessary calculations, making the game both challenging and efficient. This report covers the methodology, implementation, results, and possible improvements in future iterations.

3. Introduction

Overview

Noughts and Crosses, commonly known as Tic-Tac-Toe, is a well-known two-player game that involves a simple yet strategic decision-making process. Played on a 3x3 grid, players alternate turns marking their symbols (either 'X' or 'O'). The primary objective is to form a consecutive line of three marks in a row, column, or diagonal before the opponent does.

This project implements an AI-driven version of Tic-Tac-Toe using the **Minimax Algorithm with Alpha-Beta Pruning**. The AI opponent intelligently determines the best possible moves by evaluating future board states. Alpha-Beta Pruning optimizes the Minimax algorithm by eliminating unnecessary computations, enhancing efficiency.

Objective

- Develop an AI that plays Tic-Tac-Toe optimally using the Minimax algorithm.
- Enhance performance using Alpha-Beta pruning to reduce unnecessary computations.

- Provide an interactive command-line interface where users can play against the AI.
- Explore further improvements such as GUI development and adaptive difficulty levels.

Scope of the Project

- The project aims to provide a fundamental understanding of AI in turn-based games.
- It demonstrates the use of decision trees and pruning techniques in real-world AI applications.
- Future improvements may include graphical user interfaces (GUIs), larger board sizes, and machine learning enhancements.

---

4. Literature Review

Various AI algorithms have been explored for Tic-Tac-Toe, including:

- **Rule-Based Systems**: Hardcoded strategies that make predefined moves.
- **Minimax Algorithm**: A recursive decision-making process evaluating possible future moves.
- **Alpha-Beta Pruning**: An enhancement to Minimax that skips irrelevant branches in the decision tree.
- **Machine Learning Approaches**: Using reinforcement learning to adapt strategies dynamically.

The Minimax algorithm with Alpha-Beta pruning is preferred due to its efficiency and guaranteed optimal play.

---

5. Methodology

Algorithms Used

1. Minimax Algorithm

- A recursive algorithm that evaluates all possible moves to determine the optimal strategy.
- Assigns values to board states: +10 for AI win, -10 for opponent win, and 0 for a draw.
- The AI maximizes its score, while the human player minimizes it.

2. Alpha-Beta Pruning

- Optimizes Minimax by skipping unnecessary computations.
- Uses two threshold values (Alpha and Beta) to prune branches that will not affect the decision.
- Reduces execution time without affecting the accuracy of the AI's decisions.

Game Rules

- Played on a 3x3 grid.
- Players alternate turns placing their marks.
- The game ends when a player wins or when all cells are filled (resulting in a draw).

Implementation Approach

1. **Board Representation:** A 3x3 list stores the game state.
2. **User Input Handling:** Players enter moves using row and column indices.
3. **AI Decision Making:** The AI evaluates board positions and makes optimal moves.
4. **Game Loop:** The game continues until a win or draw condition is met.

---

6. Implementation - Code

```
import math

# Tic-Tac-Toe Board
board = [
    [' ', ' ', ' '],
    [' ', ' ', ' '],
    [' ', ' ', ' ']
]
```

```
# Function to print the board
def print_board(board):
    for row in board:
        print("|".join(row))
        print("-" * 5)

# Check if game is over
def is_winner(board, player):
    # Check rows, columns, and diagonals
    for row in board:
        if all(cell == player for cell in row):
            return True
    for col in range(3):
        if all(board[row][col] == player for row in range(3)):
            return True
    if all(board[i][i] == player for i in range(3)) or
all(board[i][2 - i] == player for i in range(3)):
        return True
    return False

# Check if board is full
def is_full(board):
    return all(cell != ' ' for row in board for cell in row)

# Minimax with Alpha-Beta Pruning
def minimax(board, depth, is_maximizing, alpha, beta):
    if is_winner(board, 'X'):
        return -10 + depth
    if is_winner(board, 'O'):
        return 10 - depth
    if is_full(board):
        return 0

    if is_maximizing:
        max_eval = -math.inf
        for i in range(3):
            for j in range(3):
                if board[i][j] == ' ':
                    board[i][j] = 'O'
```

```
                    eval = minimax(board, depth + 1, False, alpha,
beta)
                    board[i][j] = ' '
                    max_eval = max(max_eval, eval)
                    alpha = max(alpha, eval)
                    if beta <= alpha:
                        break
        return max_eval
    else:
        min_eval = math.inf
        for i in range(3):
            for j in range(3):
                if board[i][j] == ' ':
                    board[i][j] = 'X'
                    eval = minimax(board, depth + 1, True, alpha,
beta)
                    board[i][j] = ' '
                    min_eval = min(min_eval, eval)
                    beta = min(beta, eval)
                    if beta <= alpha:
                        break
        return min_eval

# AI Move
def best_move(board):
    best_score = -math.inf
    move = (-1, -1)
    for i in range(3):
        for j in range(3):
            if board[i][j] == ' ':
                board[i][j] = 'O'
                score = minimax(board, 0, False, -math.inf,
math.inf)
                board[i][j] = ' '
                if score > best_score:
                    best_score = score
                    move = (i, j)
    return move

# Main Game Loop
```

```python
def play():
    print("Welcome to Tic-Tac-Toe (Noughts & Crosses) with AI!")
    print_board(board)

    while True:
        # Player Move
        row, col = map(int, input("Enter row and column (0-2): 
").split())
        if board[row][col] != ' ':
            print("Invalid move! Try again.")
            continue
        board[row][col] = 'X'
        print_board(board)

        if is_winner(board, 'X'):
            print("You win!")
            break
        if is_full(board):
            print("It's a draw!")
            break

        # AI Move
        print("AI is thinking...")
        ai_move = best_move(board)
        board[ai_move[0]][ai_move[1]] = 'O'
        print_board(board)

        if is_winner(board, 'O'):
            print("AI wins!")
            break
        if is_full(board):
            print("It's a draw!")
            break

# Start the game
play()
```

---

7. Results and Output

```
Welcome to Tic-Tac-Toe (Noughts & Crosses) with AI!
 | |
-----
 | |
-----
 | |
-----
Enter row and column (0-2): 0 0
X| |
-----
 | |
-----
 | |
-----
AI is thinking...
X| |
-----
 |O|
-----
 | |
-----
Enter row and column (0-2): 0 2
```

```
Enter row and column (0-2): 0 2
X| |X
-----
 |O|
-----
 | |
-----
AI is thinking...
X|O|X
-----
 |O|
-----
 | |
-----
Enter row and column (0-2): 1 0
X|O|X
-----
X|O|
-----
 | |
-----
AI is thinking...
```

```
AI is thinking...
X|O|X
-----
 |O|
-----
 | |
-----
Enter row and column (0-2): 1 0
X|O|X
-----
X|O|
-----
 | |
-----
AI is thinking...
X|O|X
-----
X|O|
-----
 |O|
-----
AI wins!
```

---

8. Analysis and Discussion

- The AI efficiently blocks opponent moves and plays optimally.
- Alpha-Beta pruning significantly reduces computation time compared to Minimax alone.
- The AI never loses if it plays optimally, demonstrating the effectiveness of the algorithm.

---

9. Future Enhancements

- Implement a **Graphical User Interface (GUI)** for a more interactive experience.
- Introduce difficulty levels by modifying evaluation criteria.
- Extend the game to **5x5 or 7x7 grids** with modified winning conditions.
- Explore **machine learning** techniques to make AI adaptive.

---

10. Conclusion

This project successfully implements an AI-driven Tic-Tac-Toe game using Minimax with Alpha-Beta Pruning. The AI makes optimal moves and efficiently handles decision-making, ensuring a challenging opponent for players. The use of Alpha-Beta Pruning significantly improves performance. Future work can explore graphical interfaces, larger board sizes, and more advanced AI techniques.
