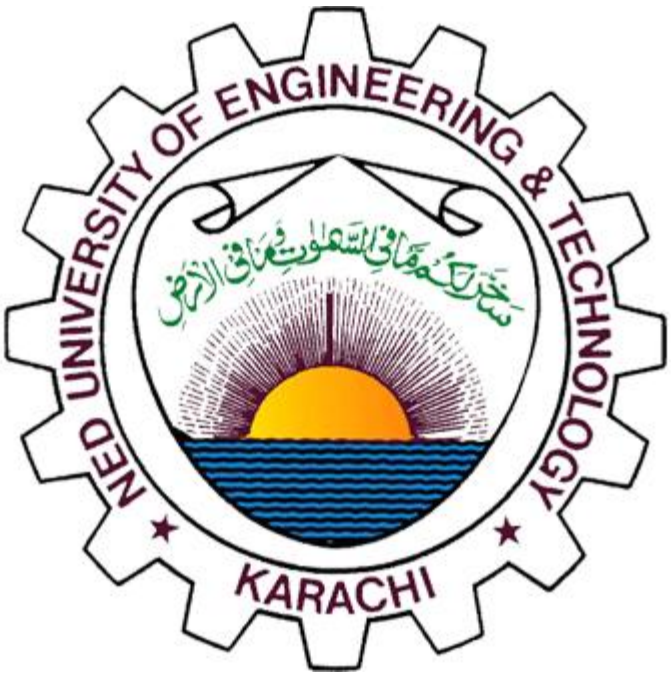
Assignment 2

AIES (CT-361)



Name: Hafsa Ali

Roll no: CR-22006

Discipline: BCIT (Cyber Security)

Teacher: Sir Muhammad Abdullah

TABLE OF CONTENTS

1. Introduction:

This project presents a graphical implementation of the classic game **Tic-Tac-Toe**, enhanced with **Artificial Intelligence (AI)** to demonstrate and compare the performance of two well-known game-playing algorithms: **Minimax** and **Alpha-Beta Pruning**.

The **objective** of the project is twofold:

- To allow a human player to compete against an AI in a game of Tic-Tac-Toe.
- To compare the computational performance (in terms of move quality and execution time) of the Minimax algorithm versus its optimized version, Alpha-Beta Pruning.

2. Methodology:

The project consists of three core components:

2.1 Game Logic

The game board is represented as a list with 9 positions. The logic includes:

- Checking for available moves
- Validating and making a move
- Checking for win conditions or draw
- Undoing moves (for backtracking in AI algorithms)

2.2 AI Algorithms

Two algorithms were implemented to simulate intelligent moves:

- **Minimax Algorithm:**
  Explores all possible future moves and chooses the optimal one assuming the opponent also plays optimally. It recursively evaluates all possible outcomes of moves to determine the best one.
- **Alpha-Beta Pruning:**
  An optimization of Minimax that reduces the number of nodes evaluated in the game tree. It keeps track of two parameters — *alpha* (maximum score the maximizer is assured of) and *beta* (minimum score the minimizer is assured of) — and prunes branches that cannot influence the final decision.

Each algorithm computes the best possible move at the AI's turn and the time taken to compute it is recorded.

2.3 GUI Design

The interface is built using **Tkinter**. Key elements include:

- A 3x3 grid of buttons representing the game board
- Player input handling
- Automated AI response after human move
- Game-over detection with a pop-up message

The AI always plays as 'O', and the human player plays as 'X'.

3. Code:

```
import tkinter as tk

from tkinter import messagebox

import time


# ---------- GAME LOGIC ----------

class TicTacToe:

    def __init__(self):

        self.board = [' ' for _ in range(9)]


    def available_moves(self):

        return [i for i, spot in enumerate(self.board) if spot == ' ']


    def make_move(self, pos, player):

        if self.board[pos] == ' ':

            self.board[pos] = player

            return True

        return False
```

```
    def undo_move(self, pos):
        self.board[pos] = ' '

    def check_winner(self):
        wins = [(0,1,2),(3,4,5),(6,7,8),
                (0,3,6),(1,4,7),(2,5,8),
                (0,4,8),(2,4,6)]
        for a,b,c in wins:
            if self.board[a] == self.board[b] == self.board[c] and self.board[a] != ' ':
                return self.board[a]
        return None

    def is_full(self):
        return ' ' not in self.board

# ---------- AI ALGORITHMS ----------
def minimax(board, is_max, player, opponent):
    winner = board.check_winner()
    if winner == player:
        return 1
    elif winner == opponent:
        return -1
    elif board.is_full():
        return 0

    if is_max:
        best = -float('inf')
```

```
        for move in board.available_moves():
            board.make_move(move, player)
            val = minimax(board, False, player, opponent)
            board.undo_move(move)
            best = max(best, val)
        return best
    else:
        best = float('inf')
        for move in board.available_moves():
            board.make_move(move, opponent)
            val = minimax(board, True, player, opponent)
            board.undo_move(move)
            best = min(best, val)
        return best

def alphabeta(board, depth, alpha, beta, is_max, player, opponent):
    winner = board.check_winner()
    if winner == player:
        return 1
    elif winner == opponent:
        return -1
    elif board.is_full():
        return 0

    if is_max:
        max_eval = -float('inf')
        for move in board.available_moves():
            board.make_move(move, player)
```

```
            eval = alphabeta(board, depth+1, alpha, beta, False, player, opponent)
            board.undo_move(move)
            max_eval = max(max_eval, eval)
            alpha = max(alpha, eval)
            if beta <= alpha:
                break
        return max_eval
    else:
        min_eval = float('inf')
        for move in board.available_moves():
            board.make_move(move, opponent)
            eval = alphabeta(board, depth+1, alpha, beta, True, player, opponent)
            board.undo_move(move)
            min_eval = min(min_eval, eval)
            beta = min(beta, eval)
            if beta <= alpha:
                break
        return min_eval

# ---------- MOVE COMPARISON ----------
def compare_algorithms(board, ai_player):
    opponent = 'O' if ai_player == 'X' else 'X'

    # Minimax
    start = time.time()
    best_val_mm = -float('inf')
    best_move_mm = -1
    for move in board.available_moves():
```

```
        board.make_move(move, ai_player)
        val = minimax(board, False, ai_player, opponent)
        board.undo_move(move)
        if val > best_val_mm:
            best_val_mm = val
            best_move_mm = move
    duration_mm = time.time() - start

    # Alpha-Beta
    start = time.time()
    best_val_ab = -float('inf')
    best_move_ab = -1
    for move in board.available_moves():
        board.make_move(move, ai_player)
        val = alphabeta(board, 0, -float('inf'), float('inf'), False, ai_player, opponent)
        board.undo_move(move)
        if val > best_val_ab:
            best_val_ab = val
            best_move_ab = move
    duration_ab = time.time() - start

    return {
        'minimax': {'move': best_move_mm, 'time': duration_mm},
        'alphabeta': {'move': best_move_ab, 'time': duration_ab}
    }

# ---------- GUI ----------
class TicTacToeGUI:
```

```
    def __init__(self, root):
        self.game = TicTacToe()
        self.root = root
        self.buttons = []
        self.ai_player = 'O'
        self.human_player = 'X'
        self.turn_count = 0
        self.build_gui()

    def build_gui(self):
        self.root.title("Tic-Tac-Toe AI Comparison")
        for i in range(9):
            b = tk.Button(self.root, text=' ', font=('Arial', 24), width=5, height=2,
                          command=lambda i=i: self.on_click(i))
            b.grid(row=i//3, column=i%3)
            self.buttons.append(b)

    def on_click(self, index):
        if self.game.board[index] == ' ':
            self.game.make_move(index, self.human_player)
            self.update_buttons()
            if self.check_end():
                return
            self.root.after(500, self.ai_move)

    def ai_move(self):
        self.turn_count += 1
        print(f"\nTurn {self.turn_count}")
```

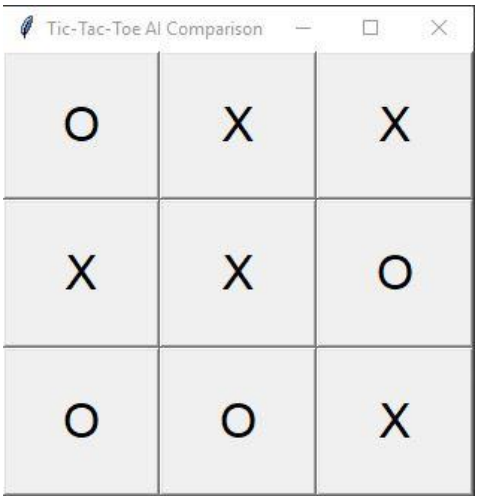
```
        result = compare_algorithms(self.game, self.ai_player)


        print(f"Minimax move: {result['minimax']['move']}, Time:
{result['minimax']['time']:.6f}s")
        print(f"Alpha-Beta move: {result['alphabeta']['move']}, Time:
{result['alphabeta']['time']:.6f}s")


        move = result['alphabeta']['move']
        self.game.make_move(move, self.ai_player)
        self.update_buttons()
        self.check_end()


    def update_buttons(self):
        for i in range(9):
            self.buttons[i].config(text=self.game.board[i])


    def check_end(self):
        winner = self.game.check_winner()
        if winner:
            messagebox.showinfo("Game Over", f"{winner} wins!")
            self.root.quit()
            return True
        elif self.game.is_full():
            messagebox.showinfo("Game Over", "It's a draw!")
            self.root.quit()
            return True
        return False


# ---------- RUN ----------
```

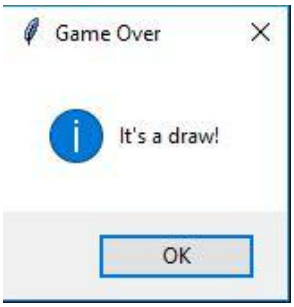
```
if __name__ == "__main__":
    root = tk.Tk()
    app = TicTacToeGUI(root)
    root.mainloop()
```

4. Results:

During each AI turn, the following data is printed in the terminal:

- The move selected by Minimax and Alpha-Beta algorithms
- The time each algorithm took to compute its move






```
Turn 1
Minimax move: 0, Time: 0.121710s
Alpha-Beta move: 0, Time: 0.010033s

Turn 2
Minimax move: 6, Time: 0.006329s
Alpha-Beta move: 6, Time: 0.002542s

Turn 3
Minimax move: 5, Time: 0.000436s
Alpha-Beta move: 5, Time: 0.000307s

Turn 4
Minimax move: 7, Time: 0.000033s
Alpha-Beta move: 7, Time: 0.000018s
```

4.1 Observations:

- Both algorithms consistently chose the same optimal moves.
- **Alpha-Beta Pruning** was significantly faster in most cases due to its pruning strategy.
- As the board filled up, the decision times for both algorithms decreased, but Alpha-Beta maintained a noticeable advantage.

This confirms that Alpha-Beta Pruning is a more efficient algorithm for decision-making in deterministic turn-based games like Tic-Tac-Toe, without sacrificing the quality of decisions.

5. Conclusion:

This project successfully demonstrates the practical implementation and comparison of Minimax and Alpha-Beta Pruning algorithms within a simple game environment. The results validate the efficiency gains of Alpha-Beta Pruning, particularly in search-heavy decision spaces.

5.1 Key Takeaways

- Both algorithms make optimal moves, but Alpha-Beta is faster.
- GUI integration provides a user-friendly way to visualize and interact with AI decisions.
- The project highlights the importance of optimization in AI algorithms for real-time applications.

5.2 Future Enhancements

- Add difficulty levels by limiting Minimax depth.
- Implement a leaderboard or scoring system.
- Allow player to choose whether to play first or second.
- Expand to more complex games (e.g., Connect Four or Chess with limited depth).