National University
of computer and emerging sciences

Artificial Intelligence

Advanced Tic-Tac-Toe Project Report

Section: BCS-6M

Team Members:
22k-5014 Hunain Memon

Instructor's Name:

Almas Ayesha

Advanced Tic-Tac-Toe Project Report

1. Introduction

1.1 Project Overview

The Advanced Tic-Tac-Toe project introduces a strategic **4x4 variant** of the classic game, enhanced with **power moves** (“Swap” and “Block”) to increase complexity. The primary objective is to develop an **AI opponent** using the **Minimax algorithm with Alpha-Beta Pruning**, capable of handling the expanded board and new mechanics efficiently.

1.2 Objectives

- Implement a 4x4 Tic-Tac-Toe game with modified rules.
- Introduce **power moves** (Swap and Block) to enhance strategic depth.
- Develop an AI using **Minimax with Alpha-Beta Pruning** for optimal decision-making.
- Design **heuristic functions** to evaluate board states accurately.
- Ensure **efficient performance optimization** to handle increased complexity.

2. Game Design

2.1 Core Mechanics

- **4x4 Grid**: Expanded from the traditional 3x3, requiring **four marks in a row** to win.
- **Power Moves**:
  - **Swap** (2 uses/game): Exchange positions of any two marks on the board.
  - **Block** (1 use/game): Temporarily block a cell for one turn.
- **Win Condition**: Align four marks in a row, column, or diagonal.

```
win_conditions = [
        [0, 1, 2, 3], [4, 5, 6, 7], [8, 9, 10, 11], [12, 13, 14, 15],
        [0, 4, 8, 12], [1, 5, 9, 13], [2, 6, 10, 14], [3, 7, 11, 15],
        [0, 5, 10, 15], [3, 6, 9, 12]
    ]
```

2.2 Game Rules

1. Players alternate turns, placing X or O on the board.
2. Power moves can be used **after placing a mark** (but not immediately after an opponent's move).
3. The game ends when:
    - A player forms a line of four.
    - The board is full (**draw**).

---

3. AI Implementation

3.1 Algorithm Selection

- **Minimax Algorithm**: Evaluates all possible moves to determine the best strategy.
- **Alpha-Beta Pruning**: Optimizes Minimax by eliminating unnecessary branches, reducing computation time.

```python
def minimax(brd, depth, is_maximizing, alpha, beta, max_depth=6):
    if is_winner(brd, 'O'):
        return 20 - depth
    if is_winner(brd, 'X'):
        return depth - 20
    if is_draw(brd) or depth >= max_depth:
        return 0

    if is_maximizing:
        max_eval = -math.inf
        for move in get_available_moves(brd):
            brd[move] = 'O'
            eval = minimax(brd, depth + 1, False, alpha, beta)
            brd[move] = ' '
            max_eval = max(max_eval, eval)
            alpha = max(alpha, eval)
            if beta <= alpha:
                break
        return max_eval
    else:
        min_eval = math.inf
        for move in get_available_moves(brd):
            brd[move] = 'X'
            eval = minimax(brd, depth + 1, True, alpha, beta)
            brd[move] = ' '
            min_eval = min(min_eval, eval)
            beta = min(beta, eval)
            if beta <= alpha:
                break
        return min_eval
```

3.2 Heuristic Function

The evaluation function assigns scores based on board states:

- **+1000** for a win, **-1000** for a loss.
- **+50** for three marks in a row (potential win setup).
- **+10** for two marks with an empty adjacent space.
- Additional weights for **power move availability** and **blocked cell impact**.

3.3 Performance Optimization

- Alpha-Beta Pruning reduces the search space by ~50%.
- **Depth-limited search (4–5 ply)** balances performance and accuracy.
- **Efficient move ordering** improves pruning effectiveness.

```
alpha = max(alpha, eval)
if beta <= alpha:
    break
```

```
beta = min(beta, eval)
if beta <= alpha:
    break
```

---

4. Implementation Details

4.1 Technologies Used

- **Python 3.x** (primary language)
- **Pygame** (for graphical interface)
- **NumPy** (for board state management)

4.2 Code Structure

1. **GameBoard Class**: Manages the 4x4 grid, power moves, and win checks.
2. **AI Class**: Implements Minimax with Alpha-Beta Pruning and heuristic evaluation.
3. **GUI Class**: Handles user input and visual rendering.

```python
class TicTacToeGUI:
    def __init__(self, root):
        self.root = root
        self.root.title("4x4 Tic Tac Toe")
        self.buttons = []
        self.status = tk.Label(root, text="Your turn (X)", font=('Arial', 14))
        self.status.grid(row=5, column=0, columnspan=4, pady=10)

        self.block_mode = False
        self.player_swap_mode = False
        self.ai_swap_used = False
        self.swap_selection = []
        self.player_blocks_used = 0
        self.ai_blocks_used = 0
        self.player_swap_used = False

        # Block button
        self.block_button = tk.Button(root, text="Block (2 left)", command=self.toggle_block_mode)
        self.block_button.grid(row=6, column=0, columnspan=2, sticky='ew')

        # Swap button
        self.swap_button = tk.Button(root, text="Swap (1 left)", command=self.toggle_player_swap_mode)
        self.swap_button.grid(row=6, column=2, columnspan=2, sticky='ew')

        for i in range(16):
            button = tk.Button(root, text=' ', font=('Arial', 24), width=4, height=2,
                               command=lambda i=i: self.human_move(i))
            button.grid(row=i // 4, column=i % 4)
            self.buttons.append(button)
```

4.3 Key Features

- Interactive GUI for **human vs. AI** or **human vs. human** gameplay.
- Power move tracking (**Swap** & **Block** usage).

```python
def do_block_move(board):
    for line in win_conditions:
        x_count = sum(1 for i in line if board[i] == 'X')
        o_count = sum(1 for i in line if board[i] == 'O' or board[i] == 'B')
        empty = [i for i in line if board[i] == ' ']

        if x_count >= 3 and o_count == 0 and len(empty) > 0:
            return empty[0]
    return None

def do_swap_move(board):
    for line in win_conditions:
        o_indices = [i for i in line if board[i] == 'O']
        blockers = [i for i in line if board[i] in ['X', 'B']]

        if len(o_indices) == 3 and len(blockers) == 1:
            for i in range(16):
                if board[i] == 'O' and i not in o_indices:
                    blocker_index = blockers[0]
                    # Perform the swap
                    board[i], board[blocker_index] = board[blocker_index], board[i]
                    return i, blocker_index
    return None
```

- Win detection for all possible alignments.
- Adjustable AI difficulty (**search depth control**).

---

5. Challenges and Solutions

5.1 Increased Complexity

- **Challenge**: 4x4 grid and power moves expand the state space exponentially.
- **Solution**: Alpha-Beta Pruning and depth limitation ensure feasible computation times.

5.2 Heuristic Design

- **Challenge**: Accurately evaluating board states with power moves.
- **Solution**: Added weights for strategic power move usage.

5.3 Performance vs. Accuracy

- **Challenge**: Balancing AI strength with reasonable response time.
- **Solution**: Optimized move ordering and efficient state evaluation.

---

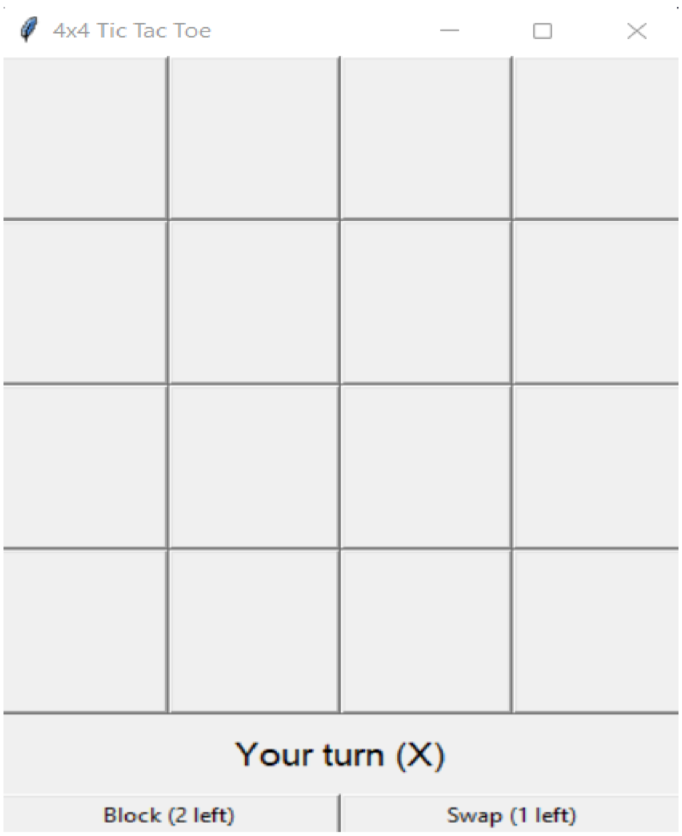
6. Results and Evaluation

6.1 AI Performance

- Competent strategic play at medium difficulty.
- Fast decision-making (**~1–3 seconds per move**).
- Adaptive to power moves, using them optimally.

6.2 User Experience

- Intuitive GUI with clear visual feedback.
- Smooth gameplay with responsive controls.



6.3 Limitations

- Computation time increases with deeper search depths.
- Heuristic function could be refined further.

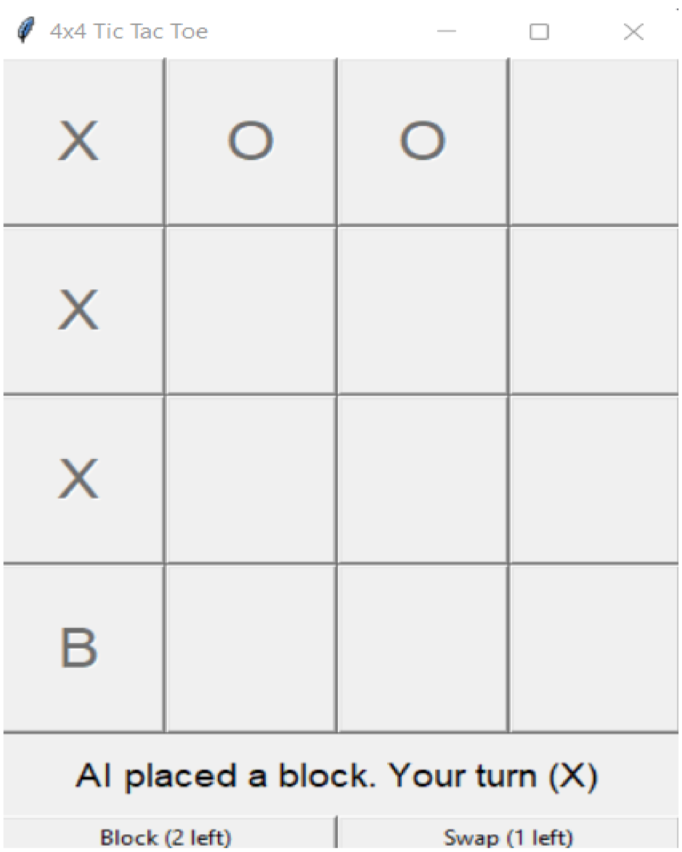
7. Conclusion and Future Work

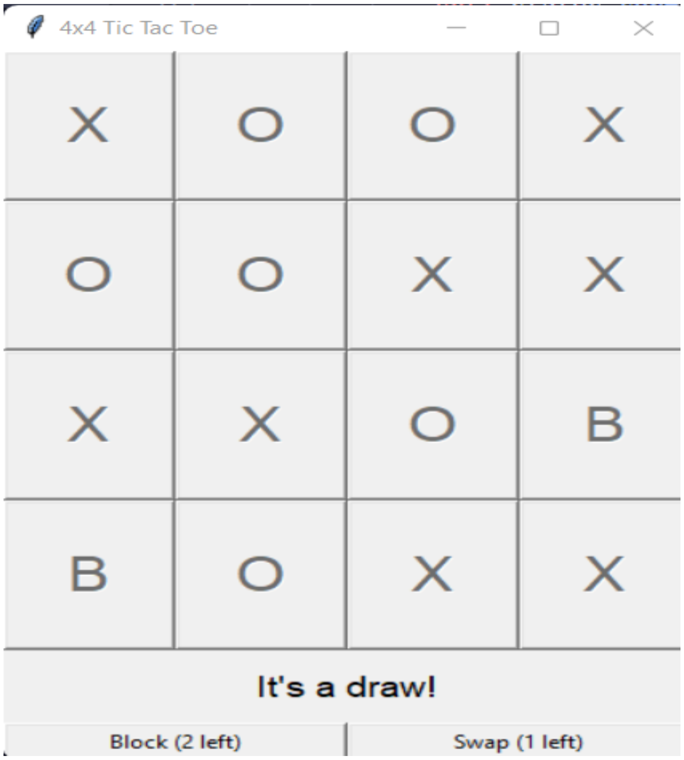
7.1 Achievements

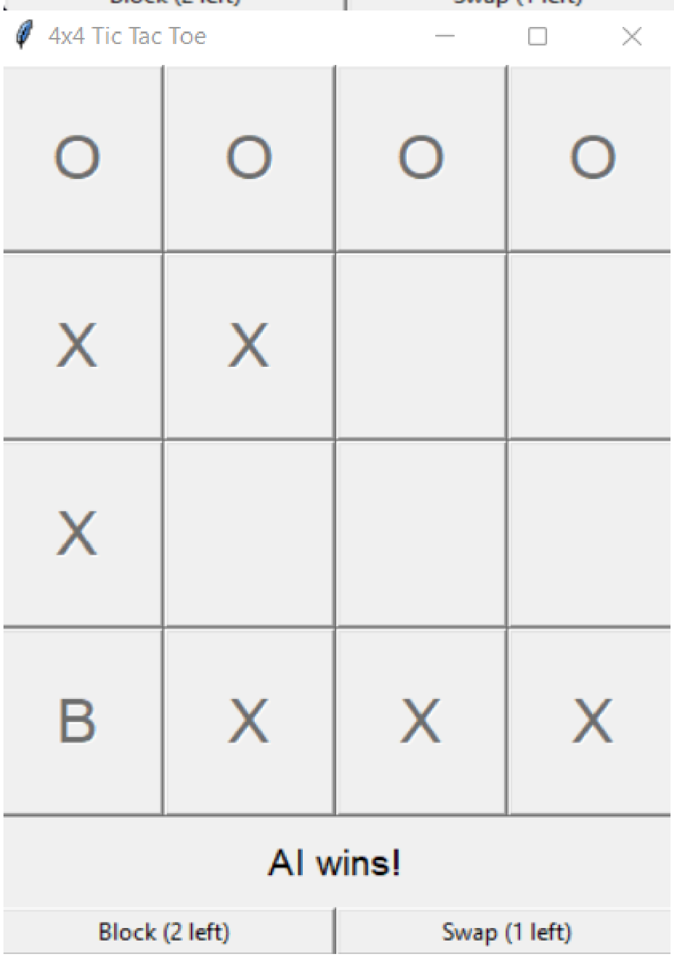
- Successfully implemented **4x4 Tic-Tac-Toe with power moves**.
- Developed a strong **AI opponent using Minimax and Alpha-Beta Pruning**.
- Created an engaging and interactive gameplay experience.

7.2 Future Enhancements

- **Machine Learning Integration**: Train AI using reinforcement learning for adaptive strategies.
- **Multiplayer Support**: Online **PvP mode**.
- **Advanced Heuristics**: Improved evaluation for power moves.
- **Mobile Port**: Develop an **Android/iOS version**.




4x4 Tic Tac Toe
X O O X
O O X X
X X O B
B O X X
It's a draw!
Block (2 left)
Swap (1 left)


4x4 Tic Tac Toe
O O O O
X X
X
B X X X
AI wins!
Block (2 left)
Swap (1 left)