CSAI 301 - Project Phase 2

Connect 4 Game

Ahmed Wael 202201415

Game Description: Connect Four

Game Overview

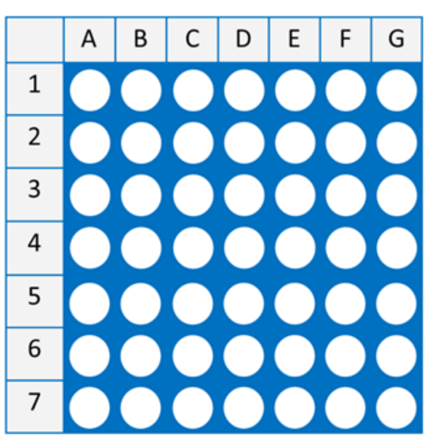
Connect Four is a classic two-player deterministic game played on a vertical 6×7 grid. Players alternate dropping colored discs into columns, with gravity pulling each disc to the lowest available position. The objective is to form a horizontal, vertical, or diagonal line of four consecutive discs before the opponent does.

Why Connect Four for Adversarial Search?

- **Deterministic:** No randomness; perfect information available to both players
- **Two-player zero-sum:** One player's gain is the other's loss
- **Finite state space:** Game always terminates within 42 moves
- **Strategic complexity:** Requires lookahead planning and threat analysis
- **Computational tractability:** Suitable for depth-limited search with pruning

State Representation

Board Structure:



Implementation:

- **Data Structure:** 6×7 NumPy array
- **Cell Values:**
  - `0` = Empty cell
  - `1` = Player 1 (Maximizing player/AI)
  - `2` = Player 2 (Minimizing player/Opponent)

Initial State

The game begins with:

- An empty 6×7 board (all cells = 0)
- Player 1 designated to move first
- No pieces on the board
- Empty move history

Actions and Moves

Valid Actions: At any states, the set of legal actions consists of all columns that are not filled

Properties:

- Minimum actions: 0 (board full - terminal state)
- Maximum actions: 7 (empty board)

Move Execution: When a player selects column c:

1. Find lowest empty row in column c
2. Place player's disc at position (row, c)
3. Update game state
4. Switch player

End game States and Utility

End game Conditions:

1. Win Condition: Four consecutive discs of the same player in any direction:

- **Horizontal, Vertical**, **Diagonal**

2. Draw Condition:

- Board full and No player has achieved win condition

3. Loss Condition:

- Opponent achieves four-in-a-row

Game Tree Complexity

Branching Factor Analysis:

- **Initial state:** b = 7 (all columns available)
- **Average state:** b ≈ 4-5 (some columns filled)
- **Late game:** b ≈ 2-3 (most columns near full)
- **Effective branching factor:** $\bar{b} \approx 4$

Depth Analysis:

- **Maximum depth:** d = 42 (all cells filled)
- **Average game length:** $\bar{d} \approx$ 35-40 moves

Why Appropriate for Minimax & Alpha-Beta:

1. **Finite horizon:** Game always terminates
2. **Perfect information:** Both players see entire board
3. **Zero-sum property:** Enables minimax principle
4. **Reasonable branching:** b ≈ 4 is manageable with pruning
5. **Evaluation feasible:** Heuristics can approximate position strength

Minimax Algorithm

Description

Minimax is a recursive algorithm that explores the game tree to find the optimal move for the current player. It operates on the principle that:

- **MAX player** (Player 1) tries to maximize the evaluation score
- **MIN player** (Player 2) tries to minimize the evaluation score

Core Concept: MAX assumes MIN will play optimally, and vice versa. The algorithm recursively evaluates all possible move sequences up to a depth limit, then selects the move leading to the best guaranteed outcome.

Implementation Details

Key Features:

1. **Depth Limiting:** Search stops at predetermined depth
2. **Terminal Detection:** Checks for win/loss/draw conditions
3. **Move Ordering:** Prioritizes center columns for better efficiency
4. **Performance Tracking:** Counts nodes explored and time taken

Advantages and Limitations

Advantages:

- ✓ Guaranteed to find optimal move
- ✓ Simple to understand and implement
- ✓ Always correct with sufficient depth

Limitations:

- ✗ Explores all nodes (no pruning)
- ✗ Exponential time complexity: O(b^d)
- ✗ Slow for deeper searches
- ✗ Examines obviously bad moves

Alpha-Beta Pruning Algorithm

[escription

Alpha-Beta Pruning is an optimization of Minimax that eliminates branches in the game tree that cannot possibly influence the final decision. It maintains two values:

- **Alpha (α):** Best value MAX can guarantee so far
- **Beta (β):** Best value MIN can guarantee so far

Key Insight: If MIN finds a move worse (for MAX) than what MAX can already achieve elsewhere, there's no need to explore further in that branch.

Pruning Conditions

Beta Cutoff (in MAX node):

```
if β ≤ α:
    prune remaining branches
    (MIN already has better option elsewhere)
```

Alpha Cutoff (in MIN node):

```
if β ≤ α:
    prune remaining branches
    (MAX already has better option elsewhere)
```

Implementation Features

Optimizations:

1. **Move Ordering:** Center columns examined first (better pruning)
2. **Pruning Counter:** Tracks number of cutoffs
3. **Metrics Tracking:** Performance comparison with Minimax
4. **Same Interface:** Drop-in replacement for Minimax

3.5 Complexity Analysis

Time Complexity:

- **Best case:** O(b^(d/2)) - perfect move ordering

- **Average case:** O(b^(3d/4)) - random ordering
- **Worst case:** O(b^d) - worst possible ordering

Space Complexity:

- O(d) - recursive call stack

Practical Impact:

- Depth 6 Minimax: $\sim 4^6$ = 4,096 nodes
- Depth 6 Alpha-Beta: $\sim 4^3$ = 64 nodes (best case)
- **Effective doubling of searchable depth**

Evaluation Function

Purpose

Since searching to terminal states is infeasible (depth 40+), we need a heuristic evaluation function to estimate the "goodness" of non-terminal positions.

Design Goals:

1. Fast computation (called millions of times)
2. Accurate position assessment
3. Captures strategic features
4. Differentiates winning vs. losing positions

Evaluation Components

1. Window Counting

A "window" is any sequence of 4 consecutive positions (horizontal, vertical, or diagonal).

Scoring:

```
Four-in-a-row (win):     +1000
Three-with-one-empty:      +10
Two-with-two-empty:         +2
Blocked (mixed pieces):      0
```

Center Column Control

The center column (column 3) is strategically valuable as it participates in more potential winning lines.

Scoring:

```
Each piece in center column: +3 points
```

Rationale:

- Center creates more four-in-a-row possibilities
- Controls board geography
- Empirically strong in Connect Four strategy

2. Opponent Evaluation

The evaluation considers both players:

```
Score = (Player_Features) - (Opponent_Features)
```

This creates a relative advantage metric rather than absolute position strength.

Why This Function Works

1. Captures Immediate Threats:

- Three-in-a-row scores +10, making it valuable
- AI will prioritize creating threats or blocking opponent threats

2. Strategic Positioning:

- Two-in-a-row gets +2
- Center control bonus promotes strong positions

3. Balanced Evaluation:

- Considers both offensive and defensive features
- Relative scoring prevents bias

4. Computational Efficiency:

- Linear scan through board: O(rows × cols)
- No complex pattern matching

Limitations and Future Improvements

Current Limitations:

- Doesn't detect complex forced-win sequences beyond depth
- Equal weight to all three-in-a-row (some are stronger)
- No consideration of "traps" (multi-threat positions)

Potential Improvements:

- **Threat detection:** Immediate win/block recognition
- **Pattern library:** Known strong/weak configurations
- **Dynamic weights:** Adjust based on game phase

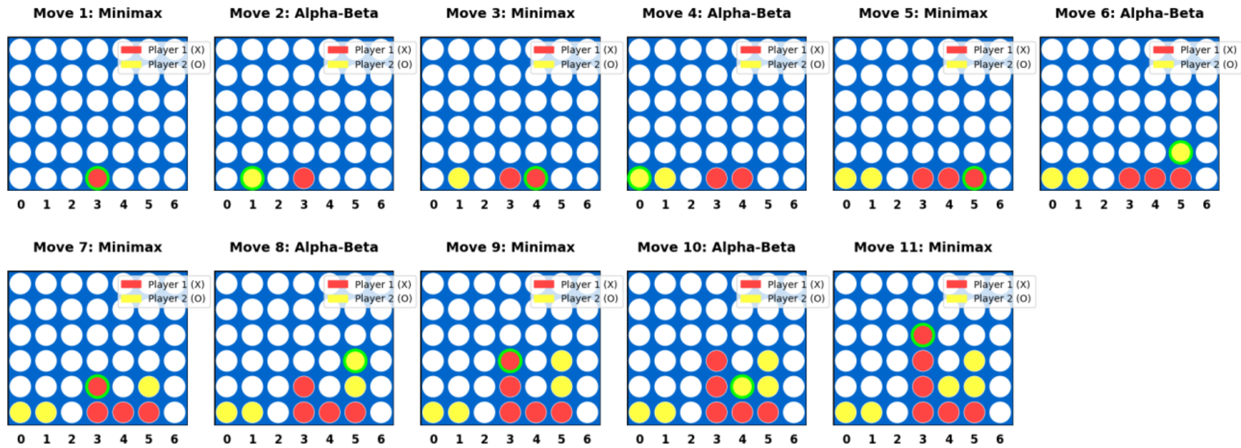
Performance Comparison

```
============================================
Test 1:  Early game (few moves)
============================================================
RESULTS
============================================================

Criterion                 Minimax             Alpha-Beta
------------------------------------------------------------
Best Move                 Column 3            Column 3
Time Taken (s)            16.9490             0.5710
Nodes Explored            19608               778
Max Depth Reached         5                   5
Best Score                12                  12
Branches Pruned           N/A                 131
============================================
EFFICIENCY ANALYSIS
============================================

Time Improvement: 96.63% faster
Nodes Reduction: 96.03% fewer nodes
Pruning Efficiency: 131 branches pruned
Same Best Move: True
============================================
```

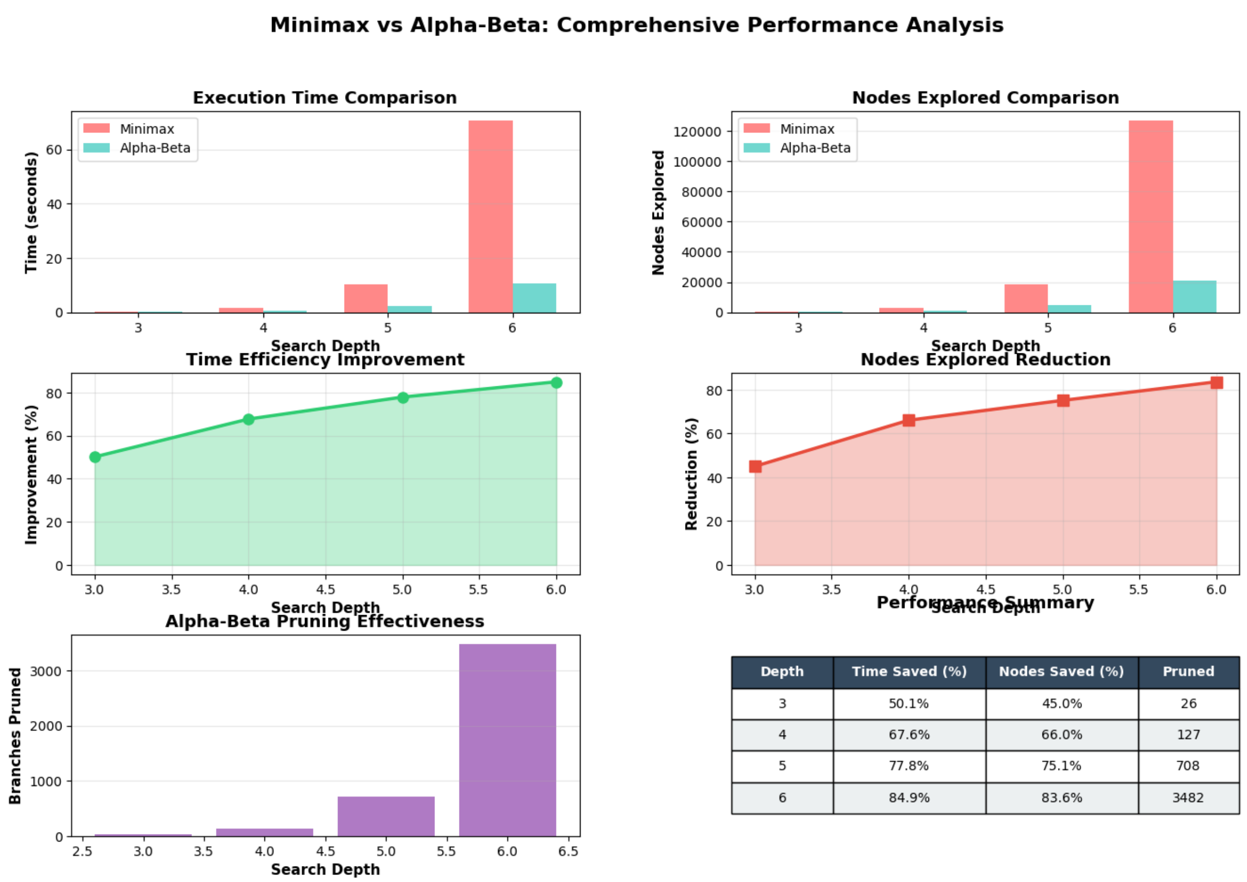
```
Test 2: Mid game
============================================================
RESULTS
============================================================

Criterion                 Minimax             Alpha-Beta
------------------------------------------------------------
Best Move                 Column 5            Column 5
Time Taken (s)            14.1387             4.4714
Nodes Explored            15699               5374
Max Depth Reached         5                   5
Best Score                -8                  -8
Branches Pruned           N/A                 662
============================================
EFFICIENCY ANALYSIS
============================================

Time Improvement: 68.37% faster
Nodes Reduction: 65.77% fewer nodes
Pruning Efficiency: 662 branches pruned
Same Best Move: True
============================================
Test 3: Complex position
============================================================
RESULTS
============================================================

Criterion                 Minimax             Alpha-Beta
------------------------------------------------------------
Best Move                 Column 2            Column 2
Time Taken (s)            112.5486            11.6636
Nodes Explored            130607              14210
Max Depth Reached         6                   6
Best Score                2                   2
Branches Pruned           N/A                 2458
============================================
EFFICIENCY ANALYSIS for Alpha-Beta
============================================
Time Improvement: 89.64% faster
Nodes Reduction: 89.12% fewer nodes
Pruning Efficiency: 2458 branches pruned
Same Best Move: True
```

Visual Results

Minimax vs Alpha-Beta



Performance Comparison Visualization



Depth	Time Saved (%)	Nodes Saved (%)	Pruned
3	50.1%	45.0%	26
4	67.6%	66.0%	127
5	77.8%	75.1%	708
6	84.9%	83.6%	3482