1. Introduction and Overview

1.1 Project Idea and Overview

This project presents the development of an AI agent capable of playing the Connect-6 game, a strategic turn-based game and an extension of Connect-4, where players alternate placing stones on a customizable n x n board. The first player to connect six pieces in a row—horizontally, vertically, or diagonally—wins. The AI implementation combines classical algorithms from game theory, specifically Minimax and Alpha-Beta Pruning, supported by heuristic evaluation functions.

1.2 Objectives

- Develop a playable version of Connect-6 with GUI using Python and Pygame
- Implement and compare Minimax and Alpha-Beta algorithms
- Design and evaluate two heuristic scoring systems (H1 and H2)
- Assess game performance under various configurations

2. Literature Review

2.1 Connect Games and AI

Games like Connect-4, Go, and Chess have long served as benchmarks for artificial intelligence research due to their well-defined rules and strategic complexity. Connect-6 extends Connect-4 by requiring six consecutive pieces to win, increasing the branching factor and difficulty.

2.2 Minimax Algorithm in Games

The Minimax algorithm is a foundational technique in game-playing AI. It has been successfully applied in two-player games such as Tic-Tac-Toe, Checkers, and Othello. However, its performance can decline in games with large state spaces unless optimization techniques like Alpha-Beta Pruning are used.

2.3 Alpha-Beta Pruning Efficiency

Research shows that Alpha-Beta Pruning can reduce the number of nodes evaluated by the Minimax algorithm by up to 75%, enabling deeper search within the same computational limits. This makes it particularly useful for games like Connect-6 where the board can be large.

2.4 Heuristic Evaluation

Heuristic evaluation functions are used to estimate the utility of a game state when the full tree cannot be explored. Studies indicate that well-crafted heuristics can significantly enhance AI performance by guiding the search towards promising paths.

2.5 Dynamic AI in Modern Games

Modern AI systems, including those in commercial video games and research simulators, often combine deterministic strategies (like Minimax) with learning-based components. This hybrid approach is gaining attention for adaptive and context-aware gameplay.

3. Proposed Solution and Dataset

3.1 Game Representation

The game state is stored in a 2D NumPy array where:

- 0 represents an empty cell
- 1 represents the human player
- 2 represents the AI

3.2 Dataset

No external dataset is required. All game data (board states and moves) are dynamically generated during gameplay and self-play testing sessions.

4. Applied Algorithms

4.1 Minimax Algorithm

Minimax is a recursive decision-making algorithm used to choose an optimal move assuming the opponent also plays optimally. It simulates all possible moves up to a certain depth and selects the move that maximizes the AI's minimum gain.

4.2 Alpha-Beta Pruning

Alpha-Beta Pruning is an optimization over Minimax that skips branches that won’t influence the final decision, improving time efficiency significantly.

4.3 Heuristic Functions

Two different evaluation functions were developed:

- **H1**: Simple and fast; favors immediate win/loss detection
- **H2**: More complex; evaluates both offensive and defensive positions with higher granularity

5. Representation of States, Actions, and the State Space

5.1 States

The game state is represented by a two-dimensional NumPy array (matrix) of size n x n. Each element in the matrix can have one of the following values:

- 0 for an empty cell
- 1 for a player's piece
- 2 for an AI's piece

5.2 Actions

An action consists of selecting a column (within board limits) in which a piece can be dropped. The system calculates the lowest available row in that column and places the piece there.

5.3 State Space

The total state space of the Connect-6 game grows exponentially with board size. For a board of size n x n, the maximum number of unique states is approximately $3^{(n*n)}$, considering each cell can be empty, contain a player's piece, or an AI's piece.

5.4 Transition Model

The transition model defines the outcome of performing a valid action in a given state. It takes the current board and the chosen action (column) and returns a new board state with the piece added at the appropriate position.

6. Experiments and Results

6.1 Setup

- Variable board sizes tested (6x6 to 12x12)
- Depths: 2, 3, 4
- Heuristic used: H1 and H2
- Algorithms: Minimax, Alpha-Beta

6.2 Evaluation Metrics

- **Win rate**
- **Draw rate**
- **Average decision time**
- **Responsiveness on different board sizes**

6.3 Observations

- Alpha-Beta reduced execution time up to 40%
- H2 produced better strategies but was slower in pure Minimax
- H2 with Alpha-Beta provided the best balance of performance and intelligence

7. Analysis, Discussion, and Future Work

7.1 Heuristic Comparison

To evaluate the effectiveness of the heuristic functions, we tested each under both Minimax alone and Minimax with Alpha-Beta Pruning. Below is a comprehensive comparison:

evaluate_window (H1)

evaluate_window was designed to be lightweight and efficient, focusing on detecting immediate winning conditions and basic offensive strategies.

evaluate_window_2 (H2)

evaluate_window_2 is more detailed, offering scores for 2- to 6-in-a-row patterns and also punishing potential opponent threats earlier.

evaluate_window vs evaluate_window_2 :

Aspect	evaluate_window	evaluate_window_2
Number of Scenarios	Limited (Only 6, 5+1 empty, 4+2 empty)	Covers more scenarios (2 to 6 pieces, both for player and opponent)
Evaluation Depth	Basic and minimal	Much deeper and smarter evaluation
Winning Score	+1000	+100000 → Strong emphasis on winning
Threat Handling	Only checks one threat (opponent has 5 pieces + 1 empty)	Handles multiple threat levels (2 to 5 pieces for opponent)
Simplicity	Simple and easy to understand	More complex but more powerful
Recommended Usage	Suitable for simple or early versions of the game AI	Better for serious AI and competitive gameplay

evaluate_window vs evaluate_window_2 - With Mini-Max:

Aspect	evaluate_window	evaluate_window_2
Evaluation Granularity	Coarse — only checks 6, 5+1 empty, 4+2 empty	Fine-grained — evaluates from 2 to 6 pieces with weight scaling
Depth Sensitivity	Shallow: only rewards near-win conditions	Deep: rewards build-up from earlier game states
Defensive Awareness	Minimal: checks only 1 specific threat	Strong: checks multiple levels of opponent threats
Minimax Decision Quality	May overlook strategic setups or early threats	More likely to choose moves that prevent threats or set up traps
AI Behavior	Short-sighted, may react too late	More proactive, sets up long-term plans
Performance (Speed)	Faster (fewer conditions to evaluate)	Slightly slower but more intelligent

Best Used For	Simple AI, early testing	Smarter AI, competitive gameplay

evaluate_window vs evaluate_window_2 - With Alpha-Beta VS. Without:

	evaluate_window	**evaluate_window_2**
Minimax (basic)	Basic logic, low awareness	More awareness, but slower
Minimax + Alpha-Beta	Same logic, much faster	Same logic, smarter and faster

Full Comparison

Aspect	evaluate_window + Minimax	evaluate_window_2 + Minimax	evaluate_window + Alpha-Beta	evaluate_window_2 + Alpha-Beta
Speed	Fast	Slower	Much faster	Fast enough with smarter logic
Evaluation Quality	Weak	Smart	Weak	Smart
Threat Awareness	Minimal	Multi-level threat aware	Minimal	Multi-level threat aware
Best Depth (Real-Time)	Depth 5-6	Depth 3-4	Depth 7-8+	Depth 5-7
AI Decision Intelligence	Medium (reactive)	Better (some planning)	Medium (reactive)	Best (deep planning + speed)
Real Game Efficiency	OK for simple play	More strategic, slower	Great balance for simple AI	Best for serious AI

Conclusion

- H1 is suitable for lightweight systems and quick AI behavior.
- H2, especially when paired with Alpha-Beta Pruning, offers strategic depth and high performance.
- Alpha-Beta Pruning significantly boosts efficiency, making H2 feasible without large delays.

7.2 Challenges

- Longer computation time with increasing board size
- Balancing speed and intelligence in real-time decisions

7.3 Future Enhancements

- Implement reinforcement learning for adaptive strategy
- Support multi-AI tournaments (self-play)
- Allow custom heuristics or AI difficulty toggles

8. Development Tools and Environment

- **Language**: Python 3.x
- **Libraries**:
    - pygame: Game interface
    - numpy: Matrix operations
    - tkinter: Input dialogs
    - math, random, sys: Logic and utilities
- **Editor**: Pycharm and Jupyter

9. Conclusion

The Connect-6 project successfully demonstrates the implementation of game AI using Minimax and Alpha-Beta Pruning, supported by tailored heuristic evaluation strategies. Experimental results show that Alpha-Beta significantly improves performance, and heuristic H2 yields superior gameplay quality. The project not only builds a playable AI opponent but sets a foundation for future extensions such as learning-based agents or adaptive difficulty systems.

9. References

- https://www.pygame.org/docs
- https://www.geeksforgeeks.org
- https://www.researchgate.net/publication/332944251_Design_and_Implementation_of_connect6_Intelligent_Game_System
- https://numpy.org
- https://www.w3schools.com

Drive Link:

https://drive.google.com/drive/folders/1105fk7bKPypZXmepnRkTwQqnPEbJEKwc?usp=drive_link

Shortened Drive Link:

https://shorturl.at/yKwdt