

---

# Project Report: Connect-4 AI & 2 Player System

Submitted by:

**Kazi Shamil Akram (1518)**

**Md. Khayrul Islam Sajib (1552)**

Team: **Roboman**

Date: 11-12-25

Subject: Python Implementation of Connect Four with AI and Two-Player Modes

Language: Python 3 (Object-Oriented)

GitHub: <https://github.com/sajib91/Connect-4-Ai-and-2-player-system.git>

---

## 1. Abstract

This project implements a fully functional version of the classic strategy game *Connect Four*. The application features two distinct game modes: a **Player vs. Player (PvP)** mode for local multiplayer and a **Player vs. AI** mode. The AI opponent is powered by the Minimax algorithm, optimized with Alpha-Beta pruning to ensure efficient and challenging gameplay. The system is architected using Object-Oriented Programming (OOP) principles in Python, ensuring modularity, scalability, and ease of maintenance. A graphical user interface (GUI) built with the `pygame` library handles all user interactions.

## 2. Game Description

Connect Four is a two-player zero-sum game played on a vertically suspended grid of six rows and seven columns.

- **Gameplay:** Players choose a color (Red or Yellow) and take turns dropping tokens into one of the seven columns.
- **Mechanics:** Tokens fall straight down, occupying the lowest available space within the chosen column.
- **Objective:** The game ends when a player successfully forms a horizontal, vertical, or diagonal line of four of their own tokens. If the board fills up before either player achieves this, the game is a draw.

## 3. System Architecture

The project is refactored into a modular structure comprising four distinct files, adhering to OOP best practices:

1. **settings.py (Configuration):**
  - Acts as a centralized repository for game constants.
  - Stores dimensions (6x7 grid), colors (RGB tuples), search depth settings, and player identifiers.
  - Allows for easy tuning of game difficulty and visuals without altering logic code.
2. **board.py (The Model):**
  - Encapsulates the state of the game grid using a NumPy 2D array for performance.
  - **Responsibilities:** Validating moves, executing piece drops, finding the next open row, and running algorithms to detect win conditions (horizontal, vertical, diagonal).
3. **ai.py (The Intelligence):**
  - Contains the decision-making logic for the computer opponent.
  - **Core Components:** The `Connect4AI` class which implements the Minimax algorithm, Alpha-Beta pruning optimizations, and a custom heuristic evaluation function.
4. **main.py (The Controller & View):**
  - The entry point of the application.
  - **Responsibilities:** managing the Pygame window, rendering graphics, handling user input (mouse clicks/keyboard), and managing the game loop.
  - **New Feature:** Implements a start menu allowing users to select between 'PvP' and 'AI' modes.

## 4. Artificial Intelligence Logic

### 4.1 Minimax Algorithm

The AI uses the Minimax algorithm to look ahead `N` moves (depth set to 5 by default) to predict the outcome of the game.

- **Maximizer (AI):** Attempts to choose the move that leads to the highest score.
- **Minimizer (Human):** The AI assumes the human will play perfectly to minimize the AI's score.

### 4.2 Alpha-Beta Pruning

To make the search computationally feasible for real-time play, Alpha-Beta pruning is applied.

- The algorithm maintains two values, `alpha` (best max option) and `beta` (best min option).
- If a branch is found to be worse than a previously examined option (`alpha >= beta`), that branch is "pruned" (ignored), significantly speeding up decision-making.

### 4.3 Heuristic Evaluation

Since the game tree cannot be fully traversed to the end state, leaf nodes at the depth limit are scored using a heuristic function:

- **Streaks:** Points are awarded for having 2 or 3 consecutive pieces with open slots (potential to become 4).
- **Blocking:** Negative scores are applied if the opponent has 3 in a row, prioritizing defense.
- **Center Control:** Weighted points are given for pieces in the center column, which statistically offers more winning combinations.

## 5. User Interface (GUI) Features

The application meets the "Functional User Interface" requirement and the "Aesthetic GUI" extra credit criteria via `pygame`.

- **Start Menu:** A text-based graphical menu prompts the user to press '1' for AI mode or '2' for Two-Player mode.
- **Interactive Gameplay:**
  - **Hover Effect:** A game piece (Red or Yellow) follows the mouse cursor at the top of the screen to indicate the current player's drop column.
  - **Turn Indicators:** In PvP mode, the piece color changes dynamically between Red (Player 1) and Yellow (Player 2).
- **Visual Feedback:** The board updates instantly upon a click. Win messages ("Player 1 Wins!", "AI Wins!") are displayed in large, colored text overlaying the board.

## 6. Implementation Summary

Component	Implementation Detail
Language	Python 3
Libraries	<code>pygame</code> (Graphics), <code>numpy</code> (Grid management), <code>math</code> , <code>random</code>
Search Depth	5 (Adjustable in <code>settings.py</code> )

<b>Evaluation</b>	Custom heuristic (Window-based scoring)
<b>Modes</b>	Human vs. AI, Human vs. Human

## 7. Conclusion

This project successfully delivers a robust implementation of Connect Four. It meets all specified requirements:

- **Search Algorithm:** Minimax is fully operational.
- **Optimization:** Alpha-Beta pruning effectively reduces search time.
- **Evaluation:** The heuristic function provides intelligent, challenging gameplay.
- **Versatility:** The addition of the 2-Player mode expands the utility of the program beyond a simple AI demo.
- **User Experience:** The graphical interface is intuitive, responsive, and aesthetically pleasing.