**CENG 3511 Artificial Intelligence Project Report: Connect Four Game**

## 1. Game Definition: In this project, the classic turn-based strategy game **Connect Four** has been developed.

**Game Board:**
The game is played on a vertical grid consisting of 6 rows and 7 columns.

**Gameplay:**
Two players (User – 1 and AI – 2) take turns dropping their pieces into any column of the grid.
Each piece falls to the lowest available row within the chosen column — either on top of existing pieces or to the bottom if the column is empty.

**Winning Condition:**
The goal is to be the first player to connect four of their pieces horizontally, vertically, or diagonally.

**Role of the AI:**
In this game, the artificial intelligence acts as Player 2 (the opponent). It waits for the user's move and then analyzes the current board state to make the most strategic move possible.

## 2. Artificial Intelligence Design

For this project, **"Option 3: AI Opponent for a Turn-Based Game"** was selected from the alternatives provided in the project brief.

**Chosen Method: Minimax Algorithm**

The AI agent is implemented using the **Minimax Algorithm**, which was also recommended in the project instructions.
Minimax evaluates possible future board states by simulating all legal moves and selecting the action that maximizes the AI's winning potential while minimizing the player's advantage.

**- Why Minimax is Suitable for Connect Four?**

Connect Four is:

- Two-player
- Zero-sum (one player's gain = the other player's loss)
- Turn-based
- Perfect-information (no hidden information; the entire board is visible)

Because of these characteristics, Minimax is an ideal approach. It systematically explores possible future game states and chooses the most advantageous move for the AI, assuming the opponent also plays optimally.

### Optimization: Alpha-Beta Pruning

To improve runtime and allow deeper search levels, **Alpha-Beta Pruning** was applied.
Alpha-Beta Pruning eliminates branches of the game tree that cannot possibly influence the final decision.
As a result, the algorithm:

- Searches fewer nodes
- Makes decisions faster
- Can look deeper into the game tree within the same computation limits

This significantly increases the AI's strength without changing the final Minimax result.

### Heuristic Evaluation Function (score_position)

Since the full game tree cannot always be explored (especially in early and mid-game), a heuristic evaluation function named **score_position** was implemented. This function allows the AI to estimate the quality of a board even when the game has not yet ended (non-terminal state).

The heuristic considers several strategic factors:

#### 1. Center Column Priority

- The center column is the strongest position on the board because it provides more opportunities to form vertical and diagonal connections.
- The function rewards AI moves placed in the central column with additional points.

#### 2. Window Evaluation

- The board is scanned in "windows" of 4 cells horizontally, vertically, and diagonally.
- Each window is analyzed to determine if it contains a potential winning pattern.

#### 3. Scoring Strategy

The AI receives positive points for advantageous patterns:

| Pattern in a 4-cell window | Score |
|---|---|
| 4 AI pieces (win) | Very high score |
| 3 AI pieces + 1 empty | Moderate score |
| 2 AI pieces + 2 empty | Small score |

To make the agent defensive as well:

- If the opponent (Player 1) has 3 connected pieces and one empty space, the function gives a large negative scoreso the AI prioritizes blocking.

In this way, the heuristic encourages both offense (winning) and defense (blocking the opponent).

### 3. Development (Game + AI Agent)

Both the game and the AI opponent were developed using the tools and programming environment recommended in the project brief.

- **Programming Language**

  **Python**

Python was chosen due to its simplicity, readability, and strong library support for numerical operations and AI logic.

**Libraries Used:**

| Library | Purpose |
|---|---|
| **numpy** | Used to efficiently represent, copy, and manipulate the game board as a 6x7 matrix. |
| **math** | Provides constants such as **math.inf** and **-math.inf**, which are necessary when defining maximum and minimum search values in the Minimax algorithm. |
| **random** | Used to randomly select one move among multiple best-scoring options. This prevents the AI from playing the exact same sequence every game and introduces variability. |

**Development Stages:**

The implementation process consisted of two main phases:

### 1. Core Game Mechanics

Before adding AI, the fundamental structure of Connect Four was implemented. Key functions include:

- **create_board()** – Initializes an empty 6×7 game grid
- **drop_piece()** – Places a piece in the selected column
- **is_valid_location()** – Checks if a column has at least one available slot
- **winning_state()** – Detects if a player has formed four connected pieces horizontally, vertically, or diagonally

These functions ensure that the board behaves according to official game rules.

### 2. Integration of the AI Logic

After the basic playable game was completed, the AI agent was implemented. This phase included the following functions:

- **minimax()** – Searches possible future game states and selects the most optimal move for the AI
- **score_position()** – Heuristic evaluation function to score non-terminal board states
- **evaluate_window()** – Analyzes each 4-cell window and assigns points based on advantageous or dangerous patterns

Together, these functions allow the AI to:

- Predict future outcomes
- Prioritize winning moves
- Block the opponent’s threats
- Make strategic and varied decisions

### 4. Testing the AI

The Minimax-based AI used in this project does not require training like Reinforcement Learning (RL) or Machine Learning (ML) models.
Instead of learning from data, it is an algorithmic AI, whose performance depends primarily on:

- The search depth parameter
- The quality of the heuristic evaluation function (score_position)

**Testing Method**

The AI was evaluated through actual gameplay simulations.
Multiple games were played directly against the AI to observe its offensive and defensive capabilities in real conditions.

Rather than theoretical tests, this method shows how the AI behaves in dynamic, unpredictable human gameplay.

**Evaluation and Results**

**-Defensive Behavior**

During testing, the AI consistently detected situations where the player had three connected pieces and was one move away from winning.
In these cases, the AI correctly blocked the winning position, demonstrating effective defense.

**-Offensive Behavior**

The AI actively created opportunities to win by forming three-in-a-row sequences and leaving an open space to complete four-in-a-row on the next move.
It did not only defend—it pursued winning strategies.

**-Search Depth**

The algorithm was tested with a search depth of 4.

- A deeper search improves decision quality but increases computation time
- A shallower search is faster but weaker strategically

A depth of 4 provided an effective balance:

| Depth | Result |
|---|---|
| **Too low** | AI becomes easy to beat |
| **Too high** | Game becomes very slow (As the search depth increases, the number of possible future board states that the AI must evaluate grows **exponentially**.<br>Because of this exponential growth, the |

| | |
|---|---|
| | computational workload on the system becomes much heavier, and the AI requires more time to calculate and return its next move.) |
| **Depth = 4** | Good difficulty + smooth performance |

With this depth, the AI analyzes:

- The player's next possible move
- Its own counter-move
- The consequences of those moves

This gives the agent strategic foresight without slowing the game.

## 5. Project Documentation and Evaluation

### Methodology:

This project followed a rule-based, algorithmic approach to build an AI opponent for a traditional board game.
Instead of machine learning or reinforcement learning, the AI was implemented using the Minimax algorithm, enabling it to analyze future game states and make strategic decisions.

### Results:

The project was successfully completed.
Using Python and NumPy, a fully functional Connect Four game was developed, along with a competitive AI agent capable of playing against the user.
The AI demonstrates intelligent behavior, including blocking threats and creating winning opportunities, making the game challenging and engaging.

### Challenges Encountered:

**1. Tuning the Heuristic Evaluation Function**
One of the main challenges was balancing the scoring parameters inside the score_position function.
Values such as:

- +5 points for three-in-a-row
- +2 points for two-in-a-row
- –4 points when the opponent is close to winning

needed careful adjustment.
If these values leaned too high or too low, the AI became either overly aggressive or

too passive. Finding the right numerical balance was crucial to achieve realistic gameplay.

**2. Performance vs. Search Depth**
Increasing the Minimax search depth makes the AI more accurate and strategic. However, it also increases computation time exponentially, slowing down move generation.

After testing different values, **depth = 4 was chosen as the ideal balance—**
- Strong and strategic gameplay
- Fast and responsive turns
- No noticeable lag

**3. Debugging the Recursive Algorithm**
Debugging a recursive Minimax implementation, especially after adding Alpha-Beta Pruning, was challenging.
To solve this:

- The board state was printed at each step
- Scores were logged and inspected
- Branch decisions were traced

This allowed mistakes in the evaluation logic and recursion flow to be identified and corrected.