**Table of contents**

[Problem Statement : Gaming 2](#_Toc196753876)

[Overview 2](#_Toc196753877)

[Features 2](#_Toc196753878)

[Game Mechanics 2](#_Toc196753879)

[AI Strategy (Minimax Algorithm) 2](#_Toc196753880)

[User Interaction 2](#_Toc196753881)

[Edge Cases Handled 3](#_Toc196753882)

[Visualization 3](#_Toc196753883)

[Analysis and Justification of Static Evaluation in Minimax 4](#_Toc196753884)

[Static Evaluation Design 4](#_Toc196753885)

[Problem Statement 2: Logic 6](#_Toc196753886)

## Problem Statement : Gaming

### Overview

This project implements a Connect Four game with a human player versus an AI opponent. The AI utilizes the Minimax algorithm for decision-making. The game board is visualized using Matplotlib, and gameplay is handled through a command-line interface.

### Features

* Players take turns dropping pieces into a 7x6 grid.
* The game checks for winning moves automatically.
* AI employs the Minimax algorithm with a depth-limited search to make strategic moves.
* The board state is visualized using Matplotlib.
* Graceful handling of user input and game interruptions.

### Game Mechanics

1. The board is initialized as a 7-row by 6-column grid.

2. The player and AI take turns dropping pieces into valid columns.

3. A piece falls to the lowest available row in the selected column.

4. The game checks for four consecutive pieces in a row, column, or diagonal to determine the winner.

5. If the board is full and no winner is found, the game results in a draw.

### 

### AI Strategy (Minimax Algorithm)

* The Minimax algorithm evaluates possible moves up to a set depth limit.
* If a winning move is found for the AI, it is chosen immediately.
* If a blocking move against the player is detected, the AI prioritizes it.
* Each board state is scored based on advantageous positions for the AI.
* The AI recursively explores move sequences and selects the optimal one based on the evaluation function.

### User Interaction

* The user is prompted to enter a column number (0-5) to place their piece.
* If the column is full, the user is asked to choose another one.
* The AI then makes its move using the Minimax algorithm.
* The board is displayed after each turn.

### Edge Cases Handled

* Invalid input (non-numeric or out-of-range values) prompts the user to retry.
* Full columns cannot be selected.
* The game gracefully exits when interrupted (e.g., via keyboard interruption).

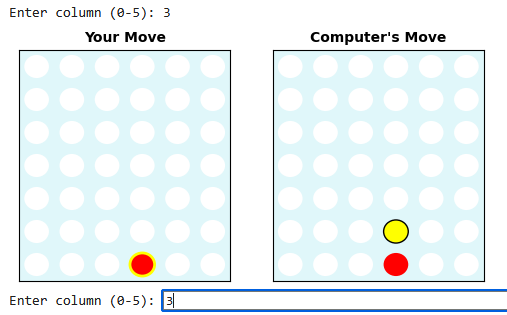
### Visualization

* The game board is rendered using Matplotlib.
* The player's and AI's moves are highlighted for better clarity.
* The latest move is marked distinctly to show progress.

When the program is run, it first shows **Enter Column (0-5):** input box. The user’s disc is red and the AI’s disc is yellow. The user's latest move disc has a yellow border and the AI’s latest move has black border to show to the user about the latest move of the respective players.

After the user enters the column number, the board with label Your Move shows the user’s move and immediately AI counters with its move using min-max algorithm. This is shown in the board with label Computer’s Move to the right of the User’s board. Another inputbox is shown below for the user to enter the next column number for the next move.

When the 4 discs of the same color are together (row-wise or column-wise or diagonally), then that player wins and the game ends. Suppose if there are no 4 discs together, then the game comes to a draw.



### Analysis and Justification of Static Evaluation in Minimax

The static evaluation function in this Connect Four implementation is designed to score a board state heuristically when the minimax algorithm reaches the depth limit or a terminal state (win, loss, or draw). The evaluation considers various strategic aspects of the game, such as control of the center, potential winning patterns, and blocking the opponent.

#### Static Evaluation Design

The function evaluate\_board(board) calculates the evaluation score based on:

1. Center Column Control (+3 per AI piece in center column)

* The center column is prioritized because placing pieces here increases future connectivity chances.
* If multiple pieces are in the center, they can contribute to diagonal and vertical wins.

2. Scoring Four-Space Windows:

* The board is analyzed using sliding windows of four cells (horizontally, vertically, and diagonally).
* Each window is evaluated using score\_window(window, piece), which assigns scores based on the number of AI or Player pieces in that segment.

**Scoring Rules in score\_window(window, piece):**

* AI pieces in a row: +100 (Winning move)
* AI pieces + 1 empty space: +5 (Strong opportunity)
* AI pieces + 2 empty spaces: +2 (Moderate opportunity)
* Opponent pieces + 1 empty space: -4 (Blocking move, important to prevent losing)

**Sample Board Before AI Move**

|  |
| --- |
| **0 0 0 0 0 0 0**  **0 0 0 0 0 0 0**  **0 0 1 1 1 0 0**  **0 0 2 2 0 0 0**  **0 0 0 1 2 0 0**  **0 0 2 1 1 0 0** |

* 1 → Player (Red)
* 2 → AI (Yellow)

**Step-by-Step Scoring:**

1. Center Control:

* AI has 1 piece in column 3 → Score = 3.

2. Horizontal Windows:

* **[1, 1, 1, 0]** (Row 3, Col 2-5): Player advantage (+5 for Player, -5 AI)
* **[2, 2, 0, 0]** (Row 4, Col 2-5): Potential AI move (+2 AI)
* **[2, 1, 1, 0]** (Row 5, Col 2-5): No immediate advantage
* Subtotal: -3 (since Player's score reduces AI's position)

3. Vertical Windows:

* **[2, 2, 1, 1]** in Column 2: No clear advantage
* **[1, 2, 1, 0]** in Column 3: No immediate advantage
* Subtotal: 0

4. Diagonal Windows:

* **[2, 1, 1, 0]** (Diagonal bottom-left to top-right): AI can form a sequence, +2
* **[0, 1, 2, 1]** (Diagonal bottom-right to top-left): Player advantage
* Subtotal: -3

**Final Static Evaluation Score:**

Total Score = 3 + (-3) + 0 + (-3) = -3

Since the score is negative, the AI currently has a weaker position, and it should prioritize blocking Player’s next move in Column 5 while attempting to improve its own diagonal or vertical alignments.

**Justification of Evaluation Heuristic**

* **Strategic Prioritization:** The evaluation function balances offensive play (winning sequences) and defensive play (blocking Player’s winning moves).
* **Avoiding Short-Sighted Play:** AI does not blindly maximize piece count but considers board control and blocking moves.
* **Tunable Weights:** The values (100 for a win, 5 for a strong sequence, -4 for blocking) reflect the strategic impact of each scenario.

This heuristic design helps the computer to make intelligent moves, ensuring that it doesn't just prioritize short-term gains but also prepares for future threats.

## Problem Statement 2: Logic

[Steps given here are based on Windows OS]

The logic is implemented using Prolog rules and structures. In order to run this, the SWI Prolog compiler has to be installed. After installing, go to the directory where medical\_diagnosis.pl is saved and enter **swipl** command as shown below:

|  |
| --- |
| C:\ACI\Assignments\Assignment 2>**swipl**  Welcome to SWI-Prolog (threaded, 64 bits, version 9.2.9)  SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.  Please run ?- license. for legal details.  For online help and background, visit https://www.swi-prolog.org  For built-in help, use ?- help(Topic). or ?- apropos(Word).  1 ?- |

Enter [medical\_diagnosis]. (each command should end with .) in the command prompt as shown below:

|  |
| --- |
| 1 ?- **[medical\_diagnosis].**  true. |

If it outputs true, it means that the program has been loaded by the SWI compiler successfully. Now enter, **decision\_tree.** command. This will show the questions and answer **yes.** or **no.** accordingly. Final diagnosis will be shown by the program.

|  |
| --- |
| 2 ?- **decision\_tree.**  Do you have a fever? (yes/no): **no.**  Do you have a runny nose and sneezing? (yes/no): |: **yes.**  Do you also have itchy eyes and a skin rash? (yes/no): |: **yes.**  Diagnosis: You may have allergy  true. |