AI Lab 2: Connect Four Game

Mohamed Tarek Hussein - 21011161
Asser Ossama Elzeki - 21010241
Youssef Tarek Hussien - 21011595

1 Introduction

Connect Four is a two-player strategy game where the objective is to align four discs of the same color either horizontally, vertically, or diagonally before the opponent. Each player drops discs into a grid, and the discs fall to the lowest available position within the chosen column. The game continues until all slots are filled, and the player with the highest number of connected fours wins.

This project aims to develop a Connect Four game featuring a Human vs. Computer mode. The computer (AI) player uses one of three algorithms:

- Minimax without alpha-beta pruning
- Minimax with alpha-beta pruning
- Expected Minimax with probabilistic disc placement

Each algorithm is designed to evaluate the board state and make optimal moves. A heuristic function guides the AI, improving efficiency by evaluating game states at a specific depth (K levels). The minimax tree for each turn is printed to the console for traceability, with an optional GUI visualization for extra clarity.

2 Project Structure

The project consists of two primary components:

- **Frontend (Svelte):** Provides an interactive graphical user interface (GUI) where players can make moves and select AI algorithms.
- **Backend (Flask):** Contains the AI algorithms and handles game logic, including move evaluation and decision-making.

3 Implementation Overview

Svelte Frontend: The frontend, developed in Svelte, offers an intuitive interface for players to interact with the game. It communicates with the backend server to retrieve AI decisions and displays the current state of the board in real-time.

Flask Backend: The backend is implemented using Flask, a lightweight Python web framework. It hosts the AI algorithms and handles all computational logic. Each AI algorithm operates by generating and evaluating a minimax tree, considering heuristic values to make optimal decisions.

4 System Architecture Overview

The system implements a Connect Four game with AI algorithms running on a Flask backend and a SvelteKit frontend. The backend provides decision-making capabilities based on Minimax, Alpha-Beta pruning, and an Expected Minimax strategy.

4.1 Frontend Interaction

The frontend sends an HTTP POST request to the server endpoint `/api/game/ai-response`. The payload contains:

- **board:** Current state of the Connect Four grid (2D array).
- **algorithm:** Selected AI algorithm (`minimax`, `alphaBeta`, or `expected`).
- **maxDepth:** Maximum search depth for the AI.
- **aiPlayer:** Specifies whether the AI is playing as `'r'` (red) or `'y'` (yellow).

4.2 Backend Processing

The Flask server receives the request and processes it in the `make_move()` function. Key steps include:

1. Parse JSON data from the request.
2. Determine the AI algorithm to use based on the `algorithm` field.
3. Invoke the corresponding decision function:
    - `minimax_decision()`: Implements basic Minimax without optimizations.
    - `alphabeta_decision()`: Enhances Minimax using Alpha-Beta pruning to improve efficiency by pruning branches that do not affect the outcome.
    - `expected_decision()`: Incorporates probabilistic outcomes, evaluating the expected utility of each move.
4. Generate a decision tree rooted at the current game state.

5 Core AI Functions

5.1 Minimax Algorithm

The Minimax algorithm (`minimax_decision()`) evaluates all possible moves up to a specified depth. It aims to maximize the AI's utility while minimizing the opponent's potential utility.

- **Maximize and Minimize Functions:** Recursively generate children nodes, evaluating each state using a heuristic function.
- **Utility Calculation:** If the maximum depth is reached or no further moves are possible, the heuristic value is calculated for the current state.

5.2 Alpha-Beta Pruning

The Alpha-Beta algorithm (`alphabeta_decision()`) is an optimization over Minimax. It maintains two values, α and β, representing the best already-explored options along the path to the root for the maximizer and minimizer, respectively.

- Prunes branches when it determines that they cannot produce a better result than previously evaluated branches.
- Reduces the number of nodes evaluated, improving efficiency significantly.

5.3 Expected Minimax

The Expected Minimax algorithm (`expected_decision()`) introduces probabilistic outcomes:

- Instead of choosing the best move directly, it considers the probability of discs falling into adjacent columns.
- **Utility Calculation:** Computes the weighted average of possible outcomes to determine the expected utility of each move.

6 Data Structure: Node Representation

Each game state is represented by a `Node` class, encapsulating:

- **Board State:** 2D array representing the current grid.
- **Children:** List of possible next states.
- **Utility Value:** Numeric score evaluating the desirability of the state.
- **Best Move:** Optimal move determined during the evaluation process.

6.1 Serialization to JSON

The AI decision tree is serialized into a dictionary using the `to_dict()` method. This method structures the data into a format that the frontend can interpret:

- **Root Node:** Contains metadata like player, depth, and utility.
- **Child Nodes:** Recursively structured to form a tree.

7 Server Response

The Flask server returns the serialized decision tree to the frontend as a JSON object. This data is used to visualize the decision process, allowing players to understand the AI's reasoning and strategy.

Heuristic Function Description

The heuristic function `heuristic(board, player, opponent)` evaluates the quality of a Connect Four board state for a given player by calculating a numerical score. This score is derived from two main components: **positional weighting** and **line potential scoring**.

Positional Weighting

The board is represented as a 6×7 grid, where each cell has a predefined weight based on its strategic importance. Central positions are given higher weights because they offer more potential connections. The weights are defined as follows:

$$\text{weights} = \begin{bmatrix} 3 & 4 & 5 & 7 & 5 & 4 & 3 \\ 4 & 6 & 8 & 10 & 8 & 6 & 4 \\ 5 & 8 & 11 & 13 & 11 & 8 & 5 \\ 5 & 8 & 11 & 13 & 11 & 8 & 5 \\ 4 & 6 & 8 & 10 & 8 & 6 & 4 \\ 3 & 4 & 5 & 7 & 5 & 4 & 3 \end{bmatrix}$$

The positional score is calculated by iterating through each cell of the board and adding or subtracting the cell's weight depending on whether it contains the current player's piece or the opponent's:

$$\text{score}_{\text{positional}} = \sum_{i=0}^{5} \sum_{j=0}^{6} \begin{cases} w_{i,j} & \text{if board}(i,j) = \text{player} \\ -w_{i,j} & \text{if board}(i,j) = \text{opponent} \\ 0 & \text{otherwise} \end{cases}$$

Line Potential Scoring

The line potential scoring evaluates sequences of pieces in various directions: horizontal, vertical, and diagonal. It rewards the formation of consecutive pieces by assigning higher scores to longer sequences. The following directions are considered:

$$\text{directions} = \{(0,1), (1,0), (1,1), (-1,1)\}$$

The sequence scores are defined as:

$$\text{sequence\_scores} = \{2 : 10, \quad 3 : 50, \quad 4 : 1000\}$$

For each sequence length k, the score is calculated by counting the number of sequences of length k for both the player and the opponent:

$$\text{score}_{\text{sequence}} = \sum_{k=2}^{4} \left(s_k^{\text{player}} \cdot \text{sequence\_scores}(k) - s_k^{\text{opponent}} \cdot \text{sequence\_scores}(k) \right)$$

where s_k^{player} and s_k^{opponent} represent the counts of sequences of length k for the player and the opponent, respectively.

Total Heuristic Score

The total heuristic score combines the positional weighting and line potential scoring:

$$\text{score} = \text{score}_{\text{positional}} + \text{score}_{\text{sequence}}$$

This heuristic function helps the AI evaluate board states by considering both piece placement and potential winning lines, thereby making strategic decisions.

8 Algorithm Runtime Comparison

To evaluate the performance of the implemented algorithms—Minimax, Alpha-Beta Pruning, and Expected Minimax—a series of runtime tests were conducted at various depths. The tests measured the time taken by each algorithm to compute the optimal move for a base Connect Four board state. The results are summarized in the following table:

Table 1: Runtime Comparison of Algorithms at Different Depths (in seconds)

Depth	**Minimax**	**Alpha-Beta Pruning**	**Expected Minimax**
2	0.011	0.000	0.003
3	0.029	0.021	0.036
4	0.229	0.086	0.208
5	1.489	0.367	1.600
6	11.862	1.127	15.102
7	102.368	5.397	111.249

The following plot illustrates the above data for a visual comparison:

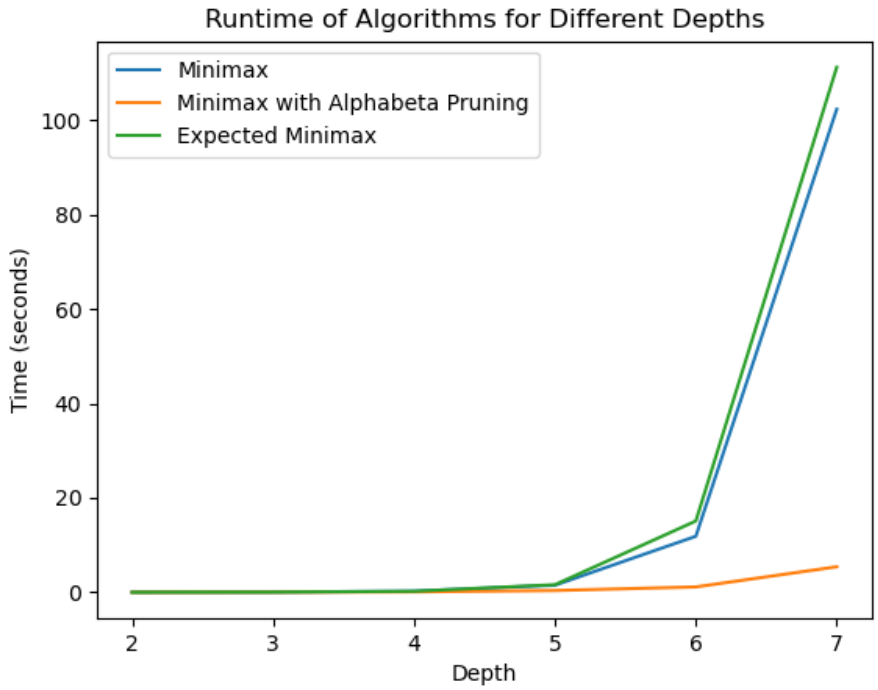

Figure 1: Runtime of Algorithms for Different Depths

Analysis and Observations

The results indicate that the Alpha-Beta Pruning algorithm significantly outperforms the standard Minimax and Expected Minimax algorithms. As the depth increases, the runtime difference becomes more pronounced:

- At shallow depths (2-3), all algorithms perform relatively quickly, though Alpha-Beta Pruning is still the fastest.
- At deeper levels (5 and beyond), the performance gap widens. For example, at depth 7, Alpha-Beta Pruning is nearly 20 times faster than Minimax and Expected Minimax.
- The Expected Minimax algorithm, which incorporates probabilistic disc placement, generally takes longer than the standard Minimax due to its additional complexity.

8.1 Conclusion

The Alpha-Beta Pruning algorithm provides the most efficient performance, especially at deeper search levels. This makes it the preferred choice for real-time gameplay scenarios where computational efficiency is crucial.

9 User Interface and Sample Runs

Our Connect Four project features a user-friendly graphical interface (GUI) built using SvelteKit. Users can easily:

- Select player roles and AI algorithms.
- Set board dimensions and other game parameters.

- Visualize the full decision tree generated by the AI, offering insight into the decision-making process.

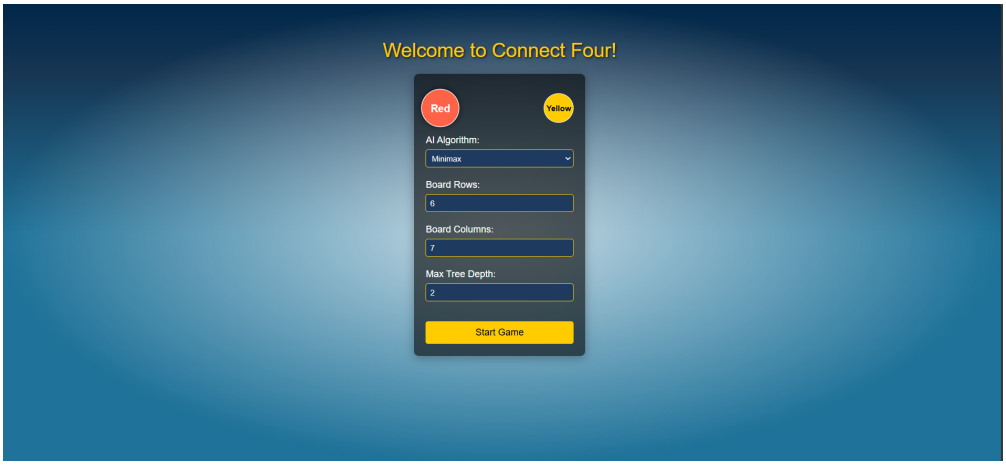

Figure 2: Welcome Page of the GUI

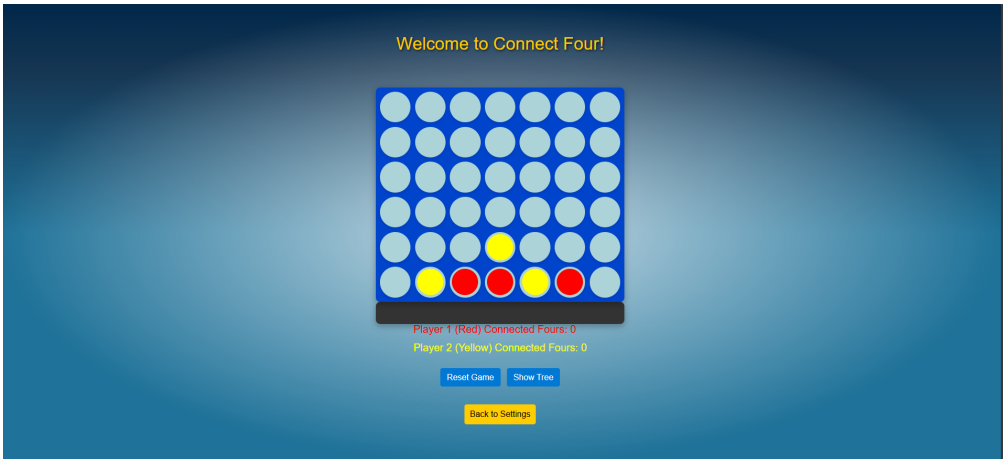

Figure 3: Board of the game

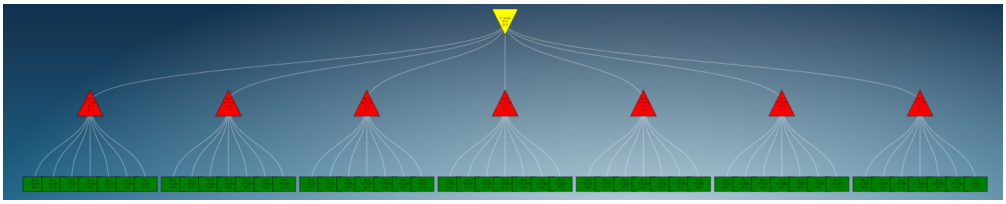

Figure 4: Minimax Decision Tree

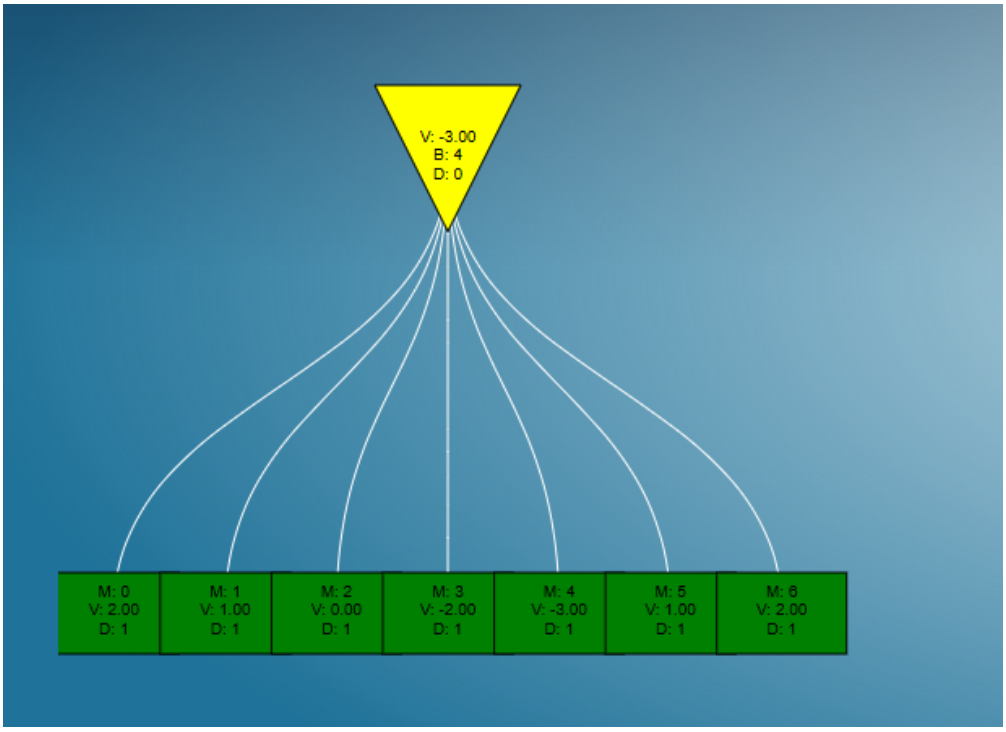

Figure 5: Closer look for Minimax Decision Tree

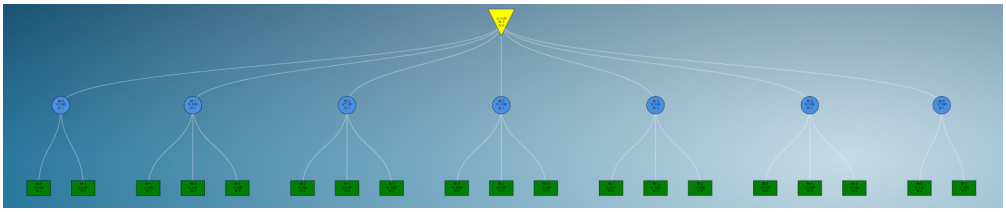

Figure 6: Expected Minimax Decision Tree

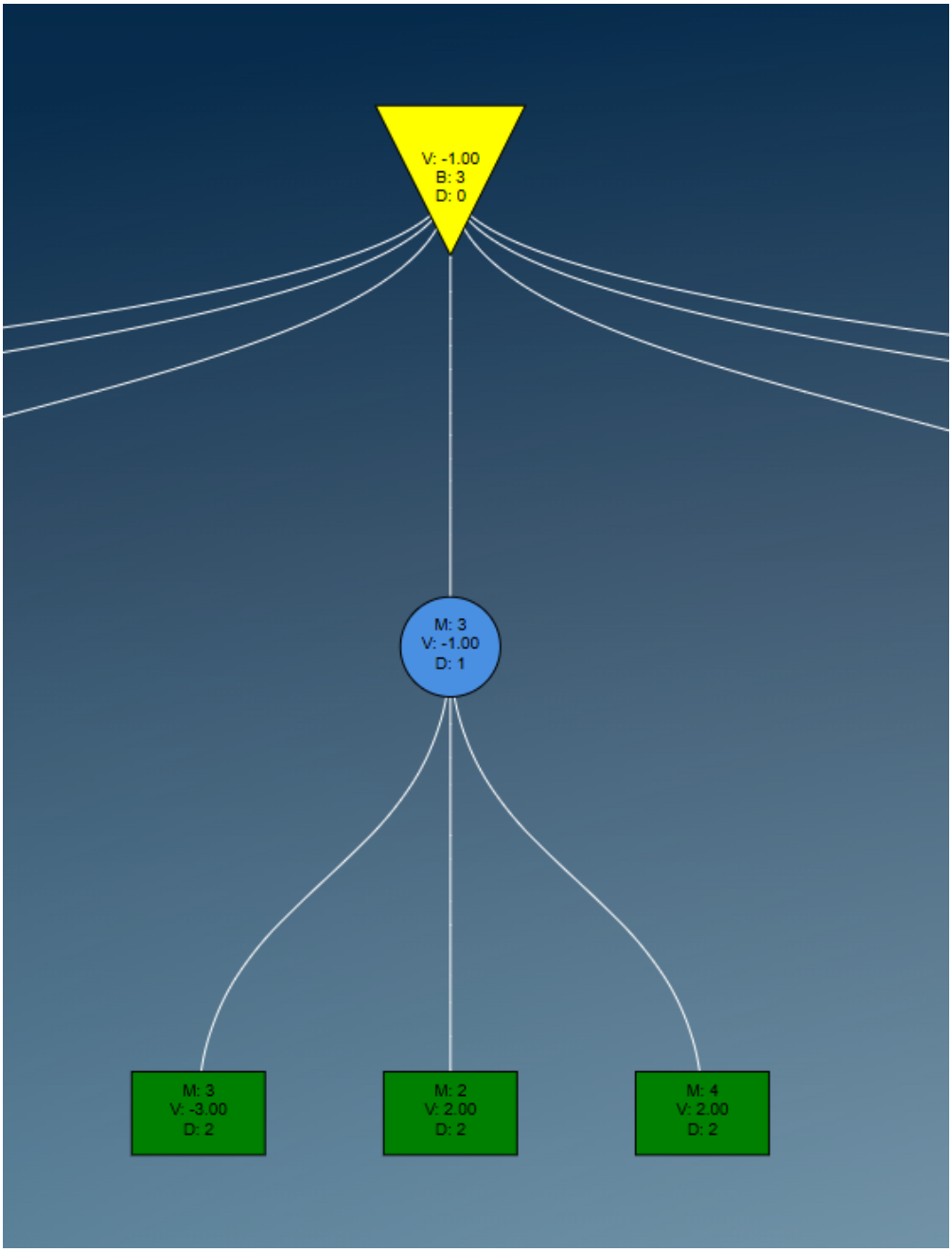

Figure 7: Closer look for Expected Minimax Decision Tree

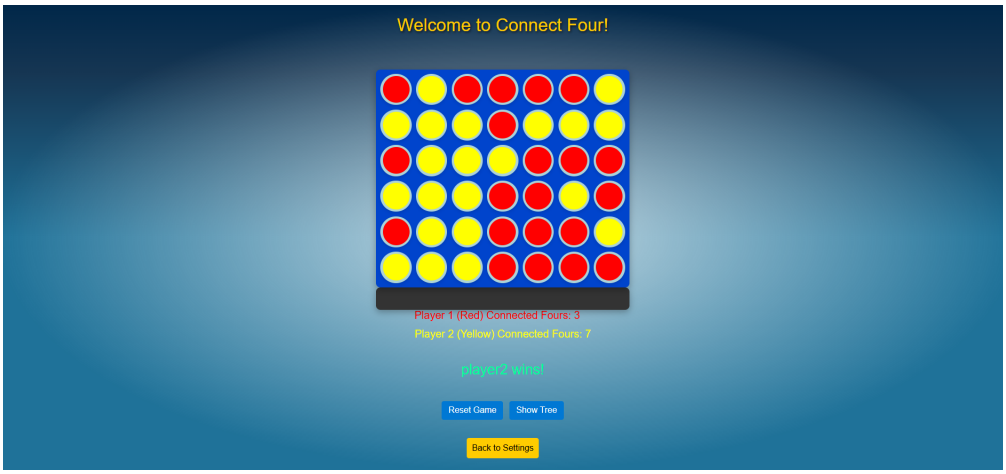

Figure 8: Endgame Board

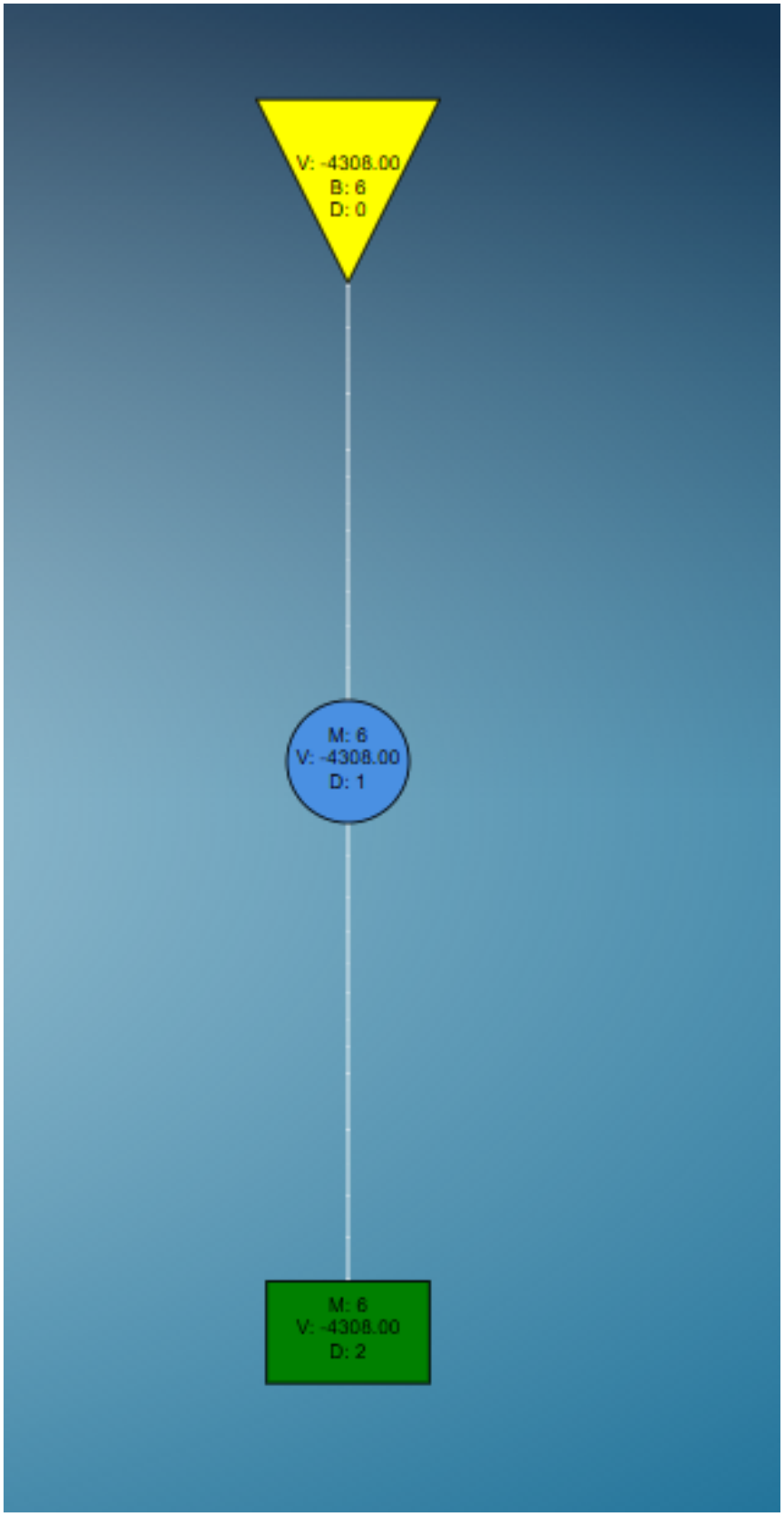

Figure 9: Endgame Expected Minimax Decision Tree

The interface displays the AI's move calculations in a clear, accessible format, allowing players to understand and learn from the AI's strategy. sample runs demonstrates how the AI evaluates different moves and selects the optimal one based on the chosen algorithm. Users can view the minimax decision tree directly within the interface, providing transparency into the AI's decision-making process.

10 Final Remarks

The Connect Four project combines robust AI algorithms with an intuitive and visually appealing interface. The real-time visualization of the decision tree enhances the user experience, making it both educational and engaging. This project showcases the effectiveness of various AI strategies and highlights the benefits of combining advanced algorithms with user-centric design.