Ashanti Boone and Sydney Brutus

Dr. Josyula

COSC 473: Artificial Intelligence

December 12, 2024

AI Game: Connect Four Paper

**Rules**

Connect Four is a game played by two players. These players are often referred to as Player 1 and Player 2. The game's setup includes a game board consisting of a 7x6 grid of empty spaces. Players drop their discs into one of the seven columns, and the disc falls to the lowest unoccupied row. The game's goal is to be the first player to form a horizontal, vertical, or diagonal line of four of one's discs. Players will either have a yellow disc (if they are Player 1) or a red disc (if they are Player 2). The game ends when a player wins, or all spaces are filled, resulting in a draw. To play the game, players alternate turns, beginning with Player 1. On a turn, a player chooses a column (if it is incomplete) and places their disc in it. A turn is complete once the disc is placed in the selected column. A player wins by aligning four discs consecutively in a horizontal, vertical, or diagonal line. The game is a draw if the board is filled and no player has achieved four consecutive discs.

A zero-sum game is one where one player's gain equals the other player's loss, meaning the total payoff across all players is always zero. If Player 1 wins, Player 2 loses, and vice versa. There is no scenario where both players can win or lose simultaneously. In the case of a draw, both players' net payoff is zero since neither wins or loses. Every advantage one player gains comes at an equal disadvantage to the other player. For example, blocking an opponent's potential Connect Four prevents them from winning, directly reducing their potential score.

Thus, Connect Four satisfies the conditions of a zero-sum game because any gain for one player is a loss for the other, and the total net benefit across both players remains zero.

**Initial State**

The initial state of Connect Four consists of an empty 7x6 grid, where all spaces are unoccupied and available for play. No discs are placed on the board, and it is Player 1's turn to make the first move. The game begins with equal opportunities for both players, as no player has an inherent advantage. The initial state represents a balanced and neutral setup, where strategic planning and execution will determine the outcome.

Each game state in Connect Four can be represented as a 2D array or grid with dimensions 7 (columns) by 6 (rows). Each cell in the grid can hold one of three values:

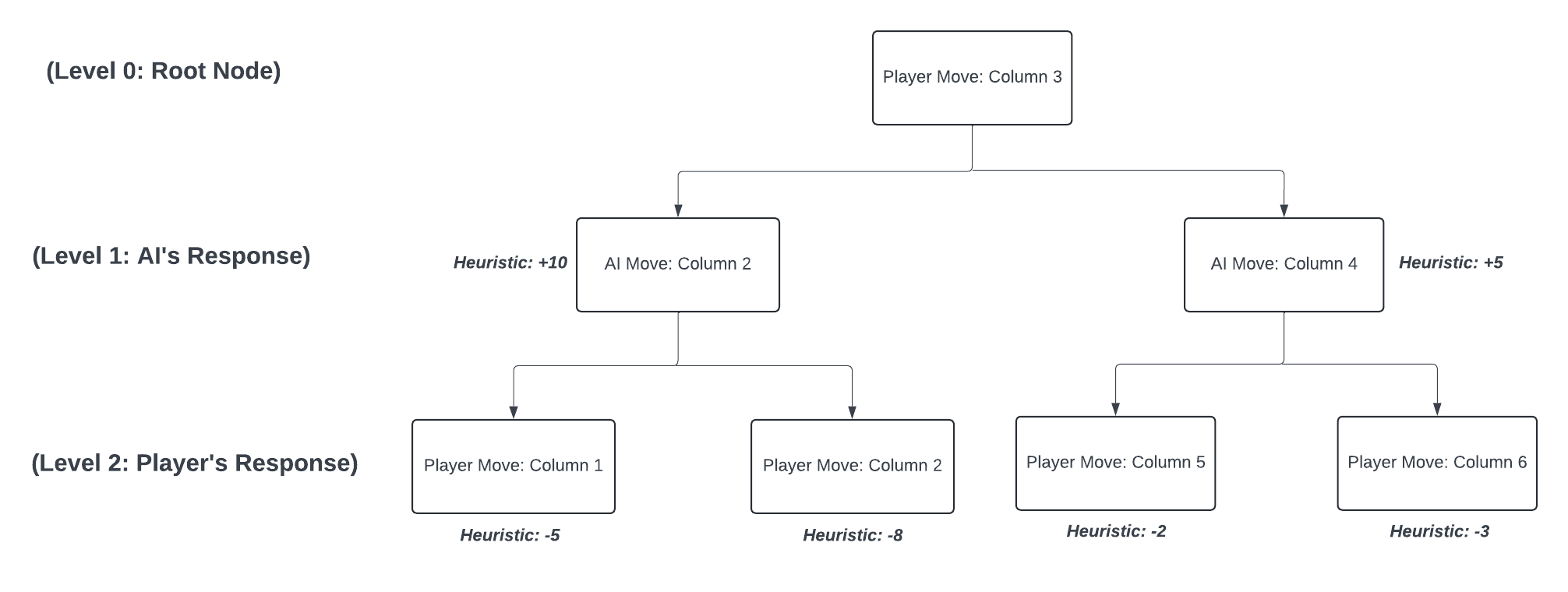
* ⚪ for an empty cell,
* 🟡 for a disc placed by Player 1,
* 🔴 for a disc placed by Player 2.

For example, the initial state is a 7x6 grid filled with ⚪s. As players take turns, the grid is updated by replacing a ⚪in the selected column with the respective player's value (🟡 or 🔴) at the lowest available row. The game state evolves with each move, and this representation allows for easy tracking of the current board configuration, checking for winning conditions and determining validity.

**Transitions**

In Connect Four, players can place their disc in any column numbered 0 to 6, which will fall to the lowest available row in that column. For example, Player 1 might start by placing a yellow disc (🟡) in column 3. The disc will land in the lowest empty cell, the bottom-most row (row 5, column 3, considering rows start at 0). Then, Player 2 could place a red disc (🔴) in column 4, which will fall to the lowest available position, row 5, column 4. If Player 1 continues placing another yellow disc in column 3, the disc will stack on top of the previous one, occupying row 4, column 3. Each subsequent move updates the grid, with discs filling the columns from the bottom up, and the game progresses as players alternate turns. The choice of column determines the transition from one move to the next, and the state of the grid reflects the current position of each disc to the others.

**Partial Game Tree**



At the very top of the tree is the root node, or level 0. This would indicate the initial state, which is when the board is empty and the player makes their first move. In this example, the player moves to column three.

The values in the middle are level 1, or is the state right after the initial state in level 0. This level is reached when it is the AI’s turn to respond to the player’s action. The player put their piece in column three, so now the AI evaluates the next possible moves it can make. To clarify, in terms of connect four hard mode specifically, the heuristic values reflects the strength of the position for the player and are as follows:

* +10: Strong move for the AI (potentially sets up a win or blocks the player).
* +5: Decent move for the AI, but not as strong.
* -5 to -8: Weaker moves for the player, where -8 is a particularly bad choice.
* -2 to -3: Neutral moves for the player, not particularly advantageous/disadvantageous, but they might be setting up something in the future.

At this level, the AI can choose between column two or column four, since

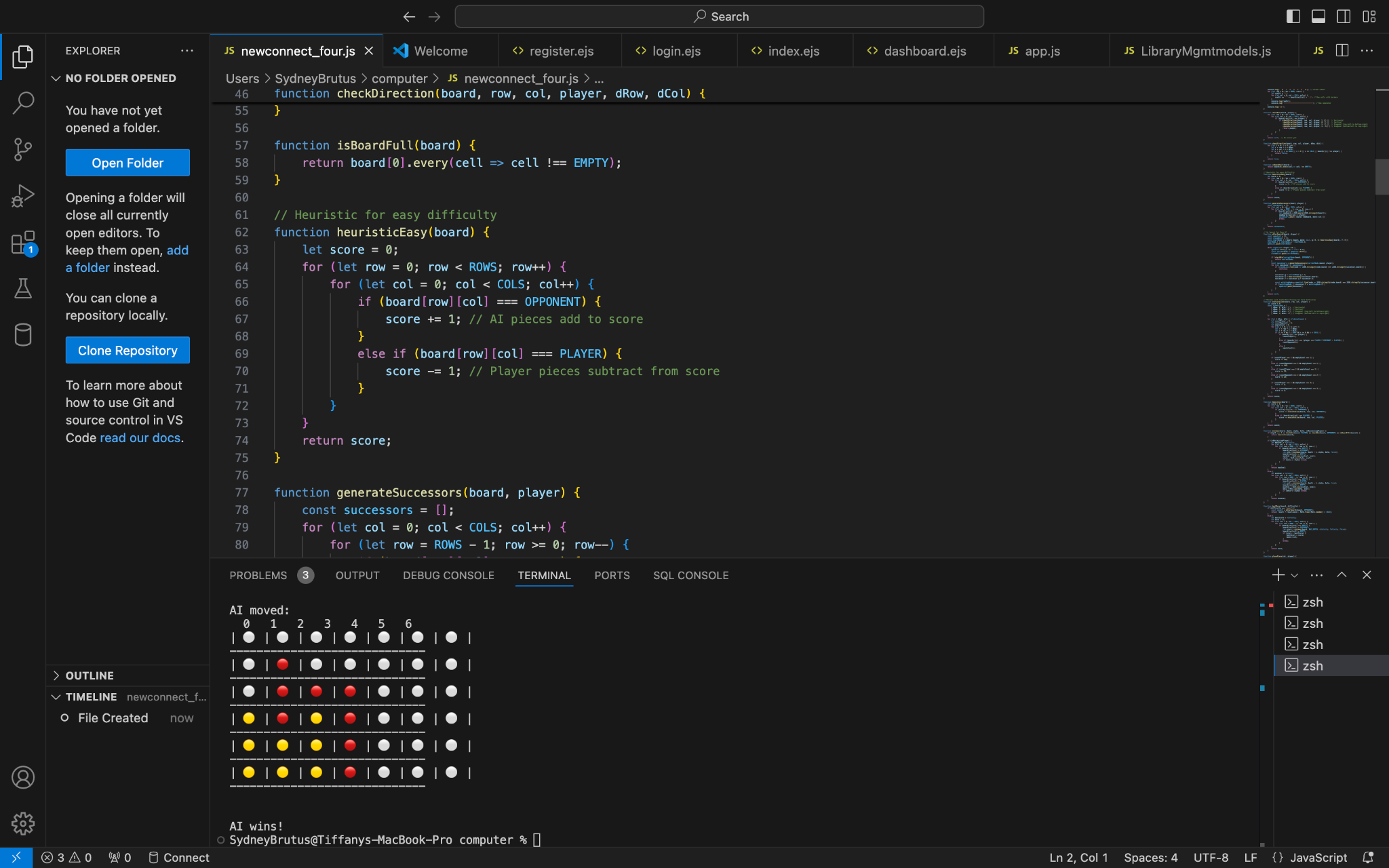
* If the AI moves in column 2, the heuristic value might be +10, which indicates that this is a good move for the AI, as it could block a potential win or set up a future opportunity for itself
* If the AI moves in column 4, the heuristic value might be +5, which indicates that while this is a decent move for the AI, it is not as strong as column two

At the very bottom of the tree is level 2, or is the state right after level 1. This level is reached when it is the Player’s turn to respond to the AI’s action. The player can still choose any column as none of them are full to the very top, but to win the game, the player can choose from columns 1,2,5, and 6

* Player Move in Column 1: The heuristic value is -5, indicating that this move is not optimal for the player, but it doesn't completely hinder them.
* Player Move in Column 2: The heuristic value is -8, indicating that this could be a weaker move (perhaps because the AI's piece is already in column 2, making it harder for the player to capitalize on this column).
* Player Move in Column 5: The heuristic value is -2, meaning this move is somewhat neutral, but the player might be trying to set up a future winning opportunity.
* Player Move in Column 6: The heuristic value is -3, meaning this is also a decent move, but not particularly strong for the player.

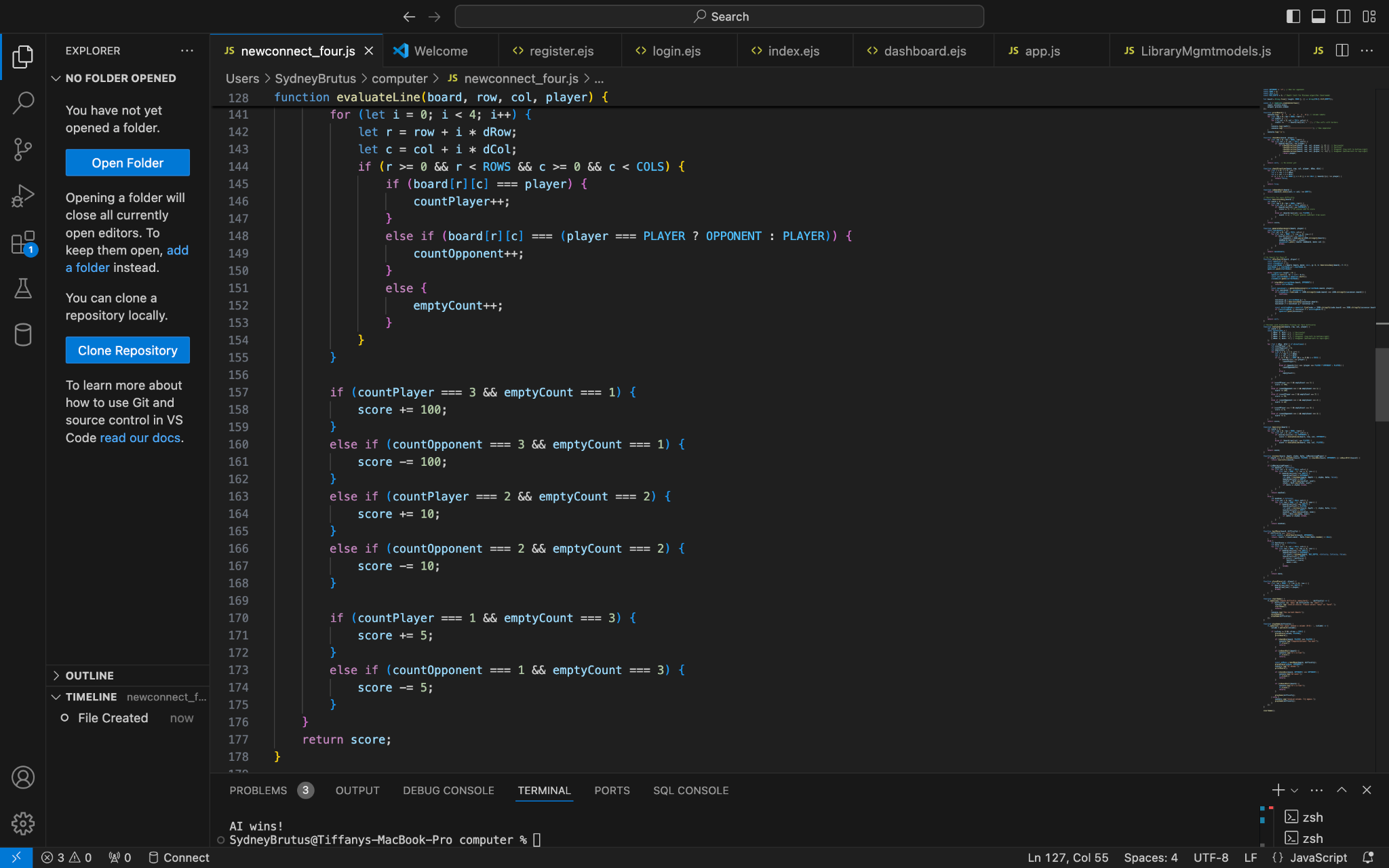
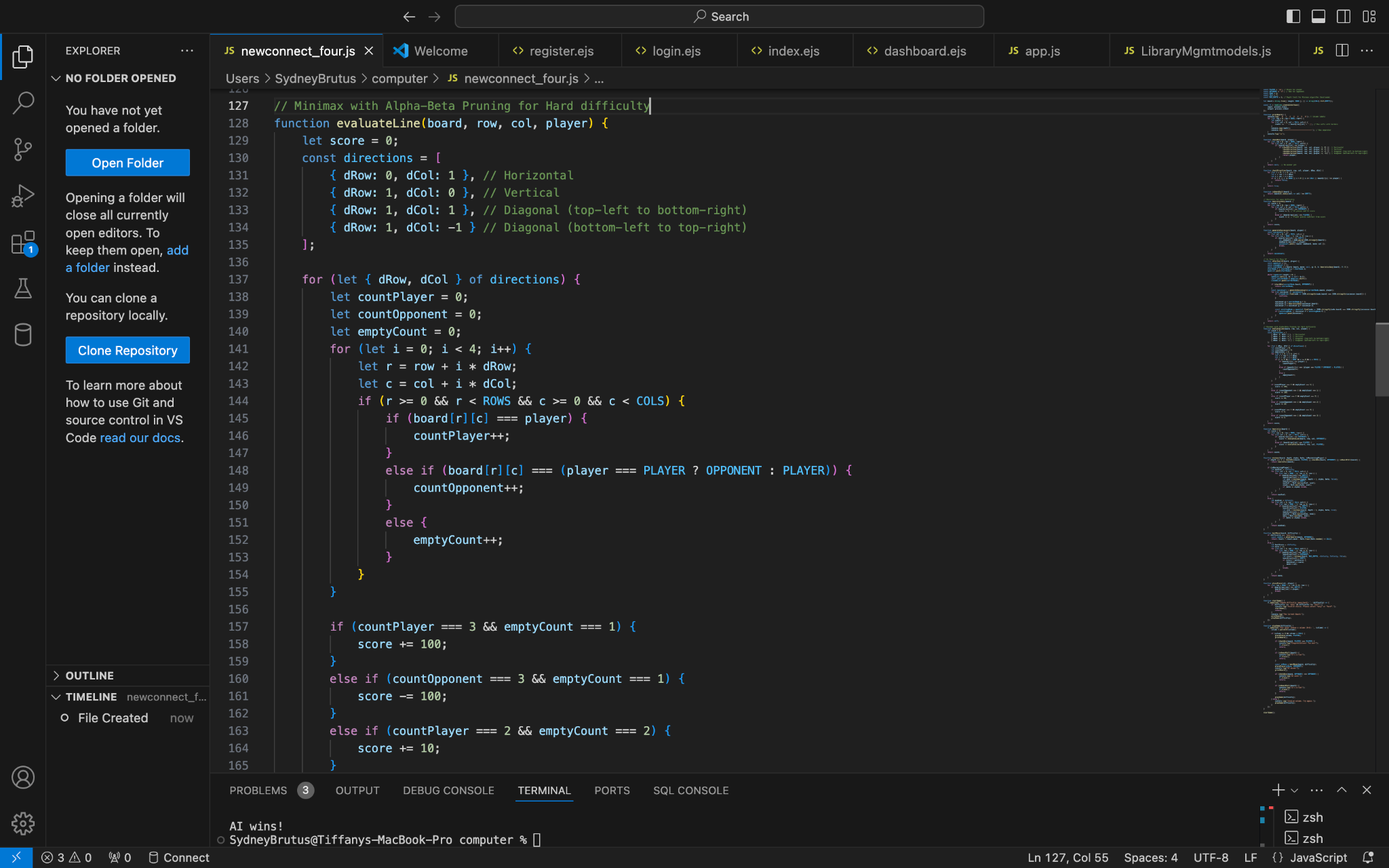
**Heuristics**

*Easy Mode*



Facing against the easy AI, we used a simpler heuristic that is based on counting the number of pieces belonging to each player on the board. So each red piece on the board contributes to a positive score, and each yellow piece contributes to a negative score. The AI will prefer game states where there are more of its own pieces and fewer of the player’s pieces, but it’s a simple evaluation. As a result, when you play on easy mode, the game goes very quickly because the AI decides its moves quickly, but not strategically.

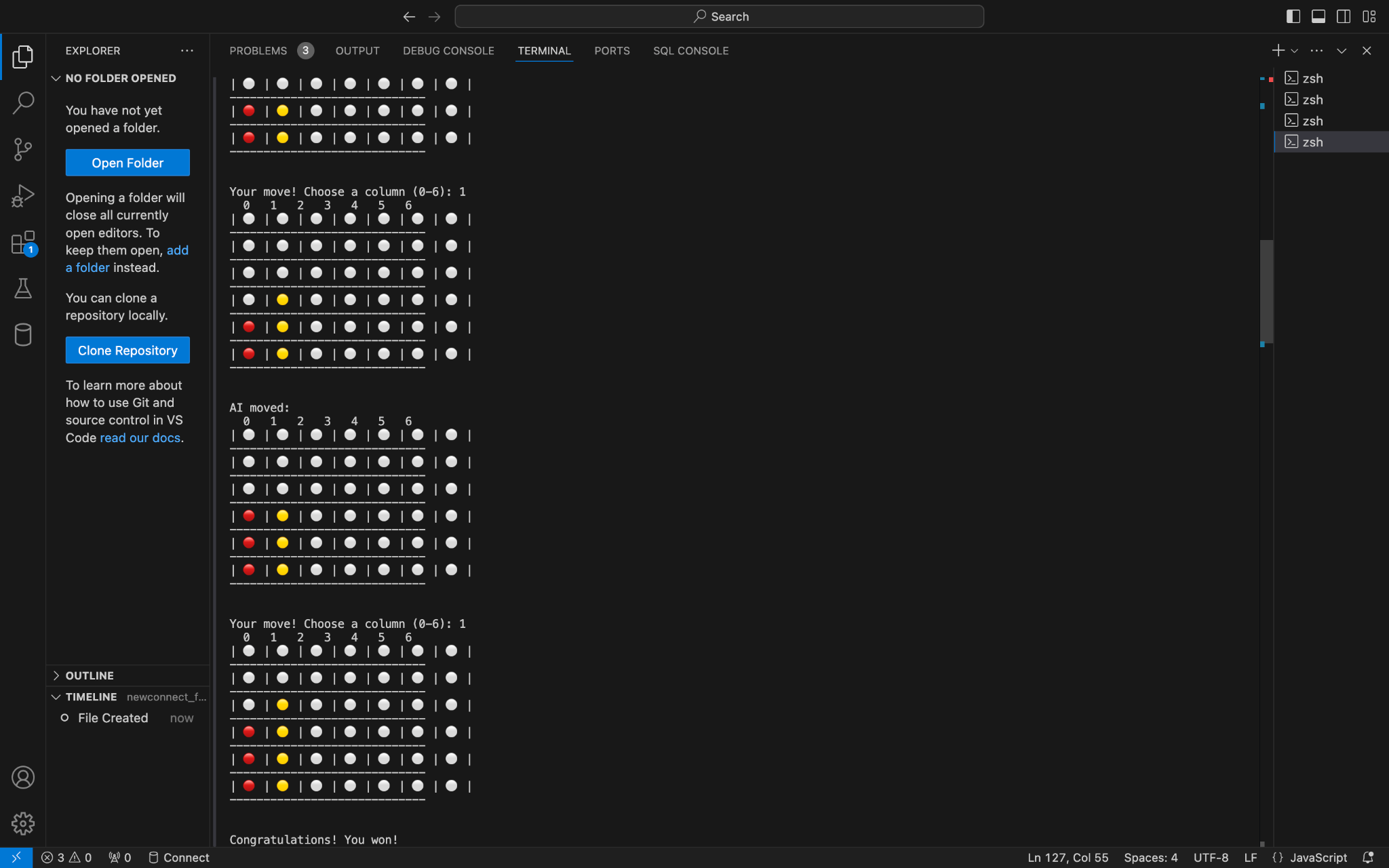
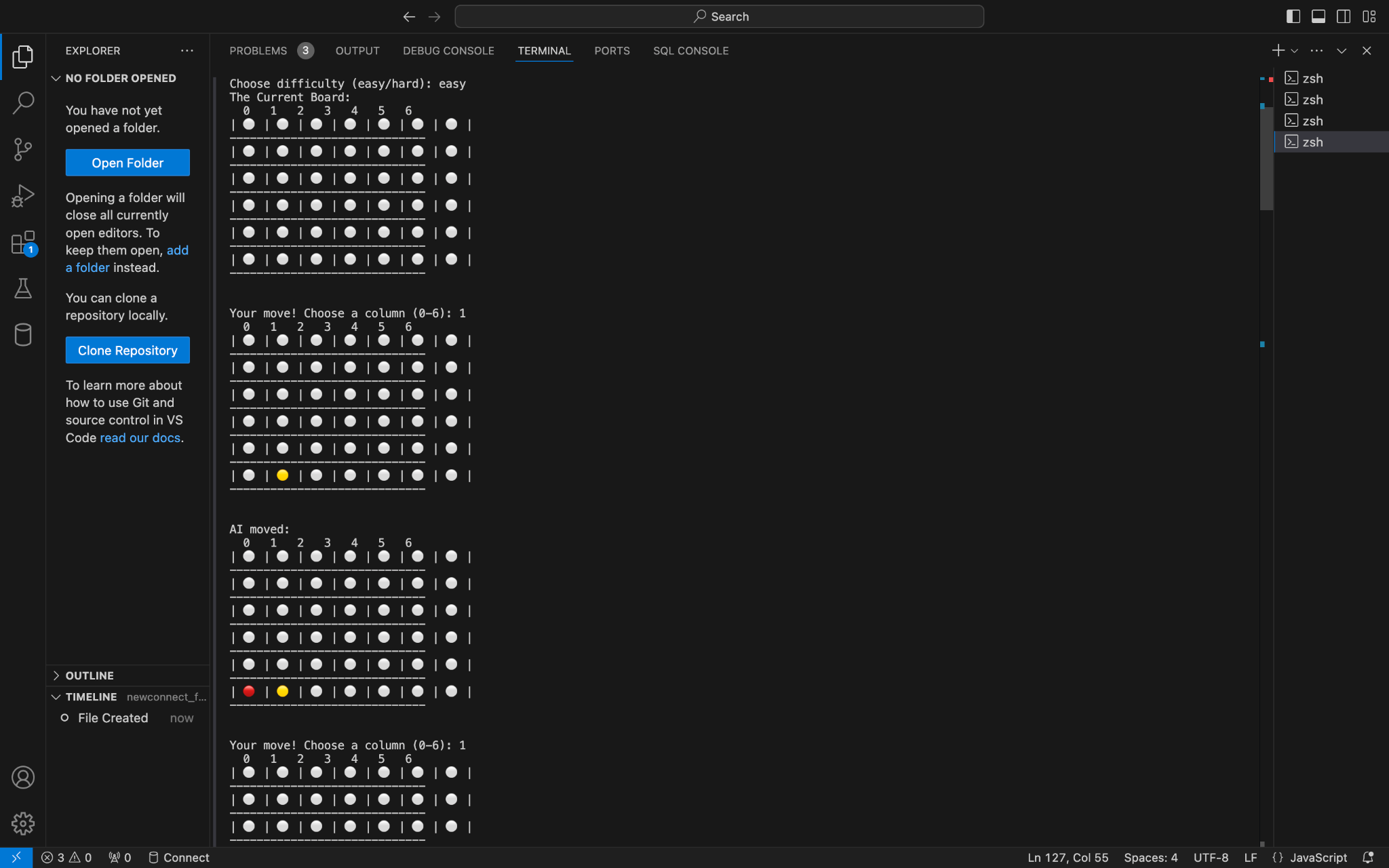
*Hard Mode*



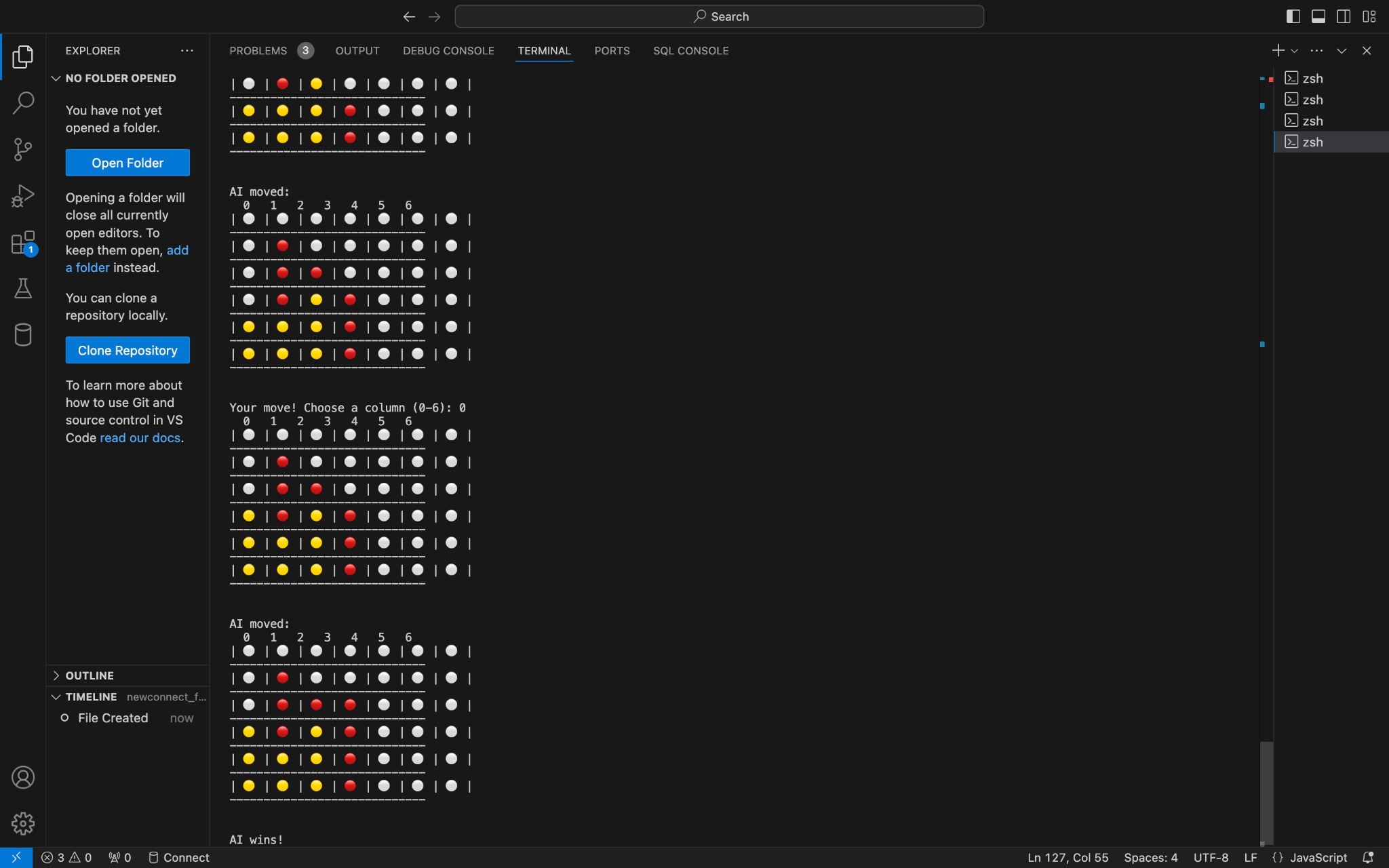
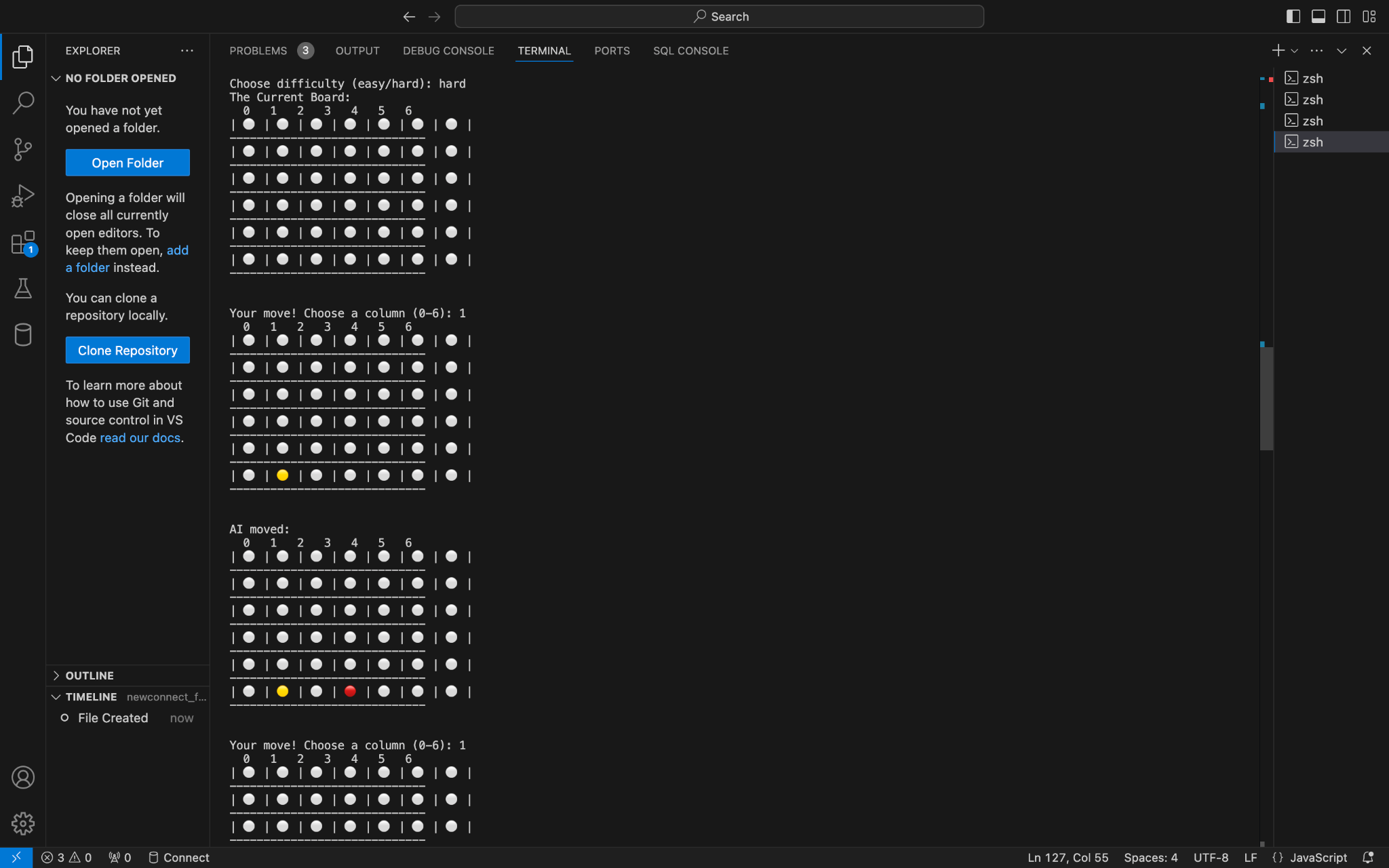
Facing against the hard AI, the heuristic is more sophisticated and takes into account potential winning and blocking opportunities. The idea is to evaluate each possible move by considering how close it is to forming a line of four pieces in various directions. If the AI has three pieces in a row with one empty spot, or the potential to complete a line, it gets a high positive score and moves offensively. If the player has three pieces in a row with one empty spot, it gets a negative score, and encourages the AI to move defensively to block the threat.

**Screenshots**

*Easy Mode*



*Hard Mode*



**Difficulties Faced**

*Implementing Minimax with Alpha-Beta Pruning:*

The Minimax algorithm, even with Alpha-Beta pruning, can be computationally expensive, especially for a game like Connect Four, where the state space grows exponentially with each turn. Implementing this algorithm correctly while ensuring the program overall still performs well, especially for the hard difficulty, took time and required careful edits. Correctly applying the Alpha-Beta pruning logic so that the AI would prune unimportant branches without affecting the optimal outcome was a challenge. We implemented the basic Minimax first to test the waters, and then slowly added Alpha-Beta pruning to the equation. We also adjusted the search depth to four and used optimizations to improve performance and reduce search time.

*Handling AI Difficulty Balancing:*

Ensuring that the AI behaved reasonably across different difficulty levels required adjusting the complexity of the decision-making process. The easy AI used A\*, while the hard AI used Minimax with Alpha-Beta pruning. The easy AI needed to be simple enough where the player didn’t have to play too strategically, but was still able to make intelligent decisions. The hard AI needed to be strong enough to provide a tough challenge where the player had to think hard themselves before making their next move, but couldn’t be unbeatable. We created a heuristic function for each difficulty. For easy mode, we kept the AI’s strategy simple, and for hard mode, we used Minimax with a sophisticated evaluation function to analyze potential threats.

*Debugging and Testing Game Logic:*

Debugging the game’s complex logic and ensuring the AI made optimal moves required extensive testing. A bug in the win-checking function or AI's move generation could make the game behave unpredictably. The win-checking algorithm had to evaluate multiple directions and ensure that it detected every possible way a player could win. For the AI, ensuring it didn't make invalid or suboptimal moves was also a challenge. We wrote comprehensive test cases to simulate different game scenarios and thoroughly tested edge cases. We also used *console.log()* to track the flow of the AI's decision-making process to ensure each line of the program was working as it should.

**Lessons Learned**

*Balancing Difficulty Levels*

We realized that game difficulty should not only depend on the AI’s decision-making complexity but also on making sure the game remains enjoyable. The AI should feel challenging but not unbeatable, which requires fine-tuning both the AI and the gameplay experience. A poorly balanced game could frustrate players or make the game too easy. The right balance made our game fun and rewarding, regardless of skill level. We learned the importance of playtesting at different levels and adjusting difficulty dynamically.

*Debugging is Key in Complex Games*

Developing games with AI requires diligent debugging, especially if you plan to have the AI function a certain way. We learned that systematic testing and logging are essential when working with complex logic such as those that existed within our game. A small bug in the AI’s logic or the win condition check could completely break the game and prevent the player from making a single move (or vice versa). We learned the importance of breaking down problems into smaller, manageable parts and using debugging tools to trace through the program to find errors.

*Efficient Algorithms Are Crucial*

One of the key lessons learned was the importance of using efficient algorithms to ensure that the game runs smoothly, especially as the search space grows in environments like Connect Four. Without optimizations like Alpha-Beta pruning, the Minimax algorithm would be too slow to be playable. Without pruning, the AI would need to search a vast number of possibilities, which would cause performance issues. Optimizing the code and its algorithms was crucial to providing a good player experience without compromising the AI’s ability to make strategic decisions in a strategic setting. We learned that the trade-off between complexity and performance is essential in game development, and should always look for opportunities to optimize time-consuming processes.

**Github**

<https://github.com/DaBombShan11/COSC-473---AI-Game-Connect-Four->