Connect4

ARTIFICIAL INTELLIGENCE

7427 | 7806 | 7861

Introduction:

Connect Four is a two-player board game where the objective is to be the first to form a horizontal, vertical, or diagonal line of four pieces of the same color on a grid. In this report we present the implementation of a Connect Four game using two different algorithms: Minimax with Alpha-Beta Pruning , without pruning and Heuristic Pruning.

In this game the player is the opponent(minimizer) while the AI is the maximizer.

We divided our code into GUI and engine connected together through agent function, also we implemented several helping functions like is_terminal to check if it is a draw game and get_valid_location to check if the tile is valid to put the piece in it.

Implemented Algorithms:

1. Minimax with Alpha-Beta Pruning:

- Minimax is a decision-making algorithm commonly used in two-player turn-based games.
- It explores the game tree recursively, alternating between maximizing and minimizing players, to determine the optimal move.
- Alpha-Beta Pruning is applied to improve the efficiency of the Minimax algorithm by pruning branches of the tree that are guaranteed to be worse than previously explored branches.

2. Heuristic Pruning:

- Heuristic Pruning is an enhancement to the traditional Minimax algorithm.
- Instead of exploring the entire game tree to a certain depth, it uses a heuristic function to evaluate the game state at each depth and decides which branches to prune based on the heuristic score.
- This approach aims to reduce the number of nodes expanded while still making reasonably good decisions.

3. Without Pruning:

- In this approach, the game tree is explored to the specified depth without any pruning techniques.
- It serves as a baseline comparison to assess the effectiveness of pruning techniques in reducing the search space.

Data Structures Used:

1. **Grid Representation:**

- The game board is represented as a 6x7 grid where each cell can contain one of three values: empty (0), player 1 piece (1), or player 2 piece (2).
- The grid is stored as a 2D list in Python.

2. **Tree Structure:**

- For the Minimax and Heuristic Pruning algorithms, a tree structure is used to represent the game tree.
- Each node in the tree corresponds to a game state, and child nodes represent possible future game states resulting from different moves.
- The tree is implemented as a nested dictionary, where each node contains information such as depth, value, and child nodes.

Comparison:

- Minimax without Alpha-Beta Pruning:

K Value	Time Taken (seconds) (avg of 21 iterations)	Nodes Expanded
3	0.03613494691394624	399
4	0.3159065700712658	2800

- ExpectedMinimax:

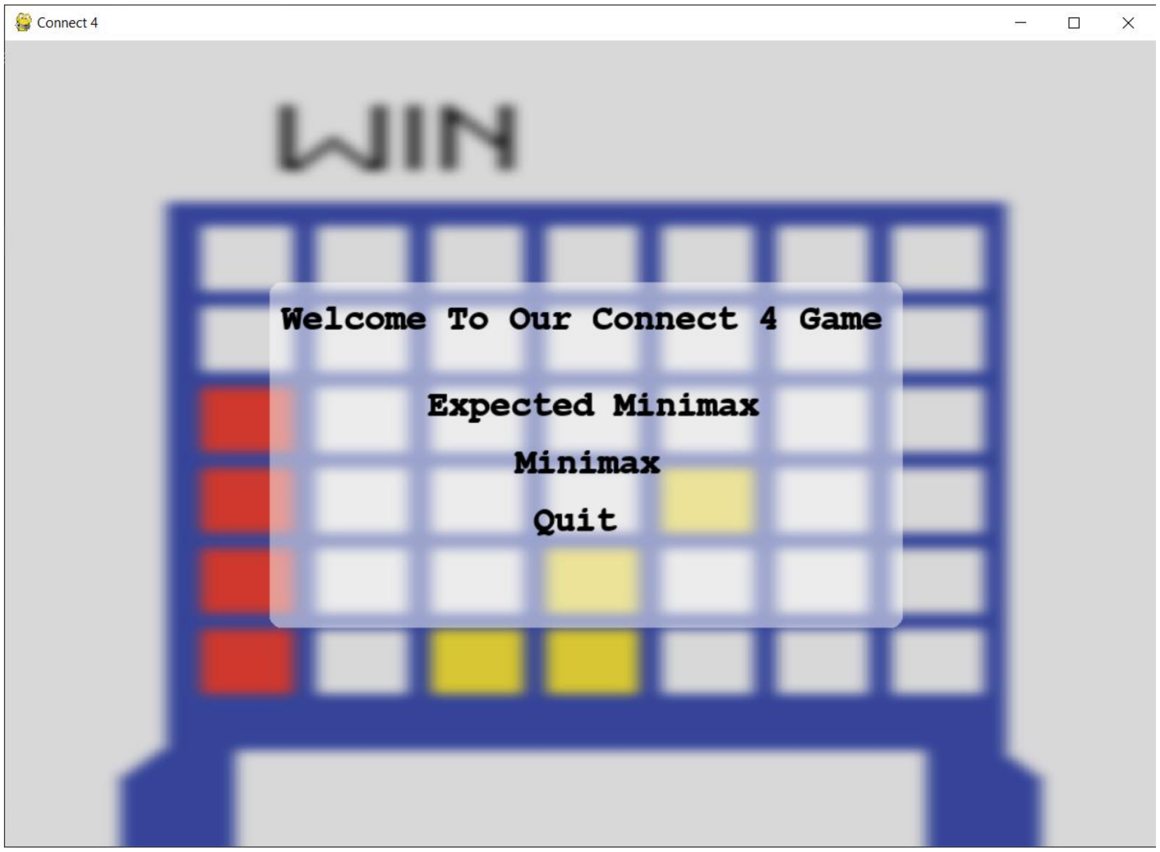
K Value	Time Taken (seconds)	Nodes Expanded
3	0.04941530454726446	399
4	0.347070478257678	2800

- Heuristic Pruning:

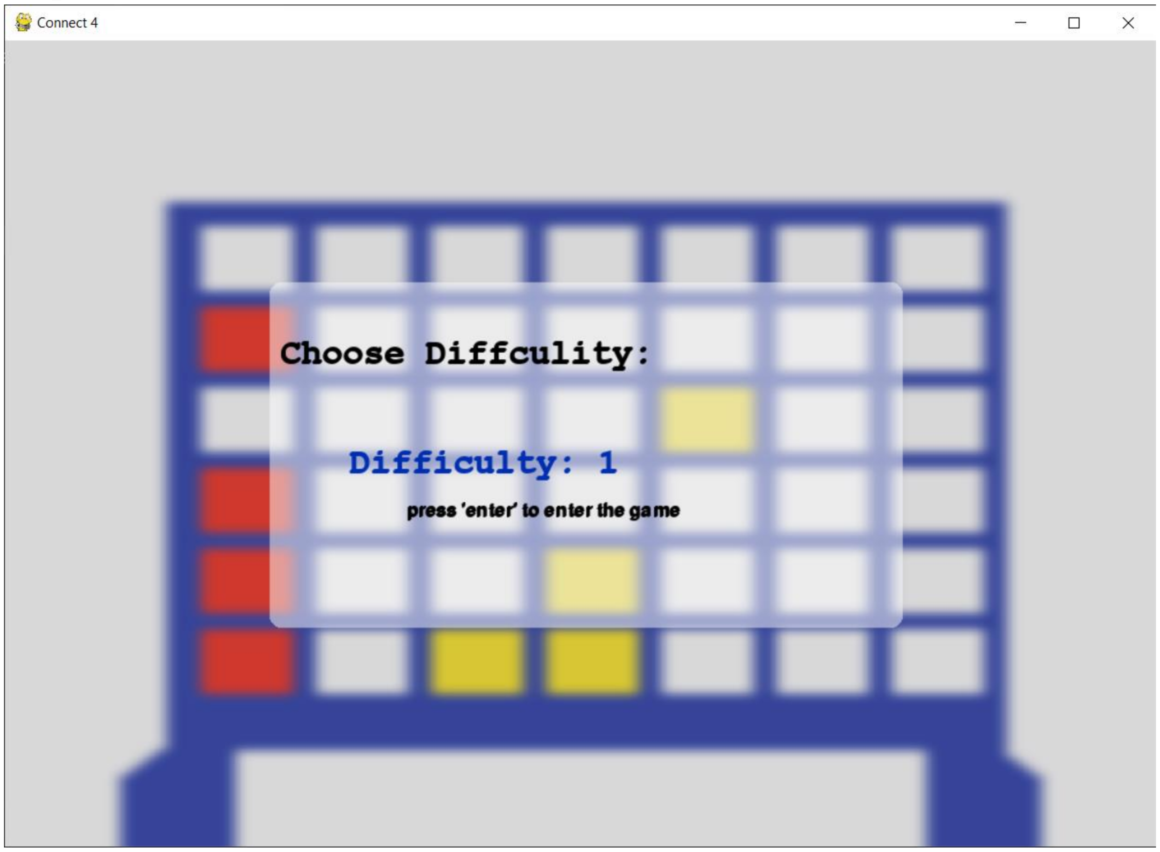
K Value	Time Taken (seconds)	Nodes Expanded
3	0.051667576744442896	171
4	0.1568800381251744	1422

Sampe runs:

Home page:



Expected minimax:

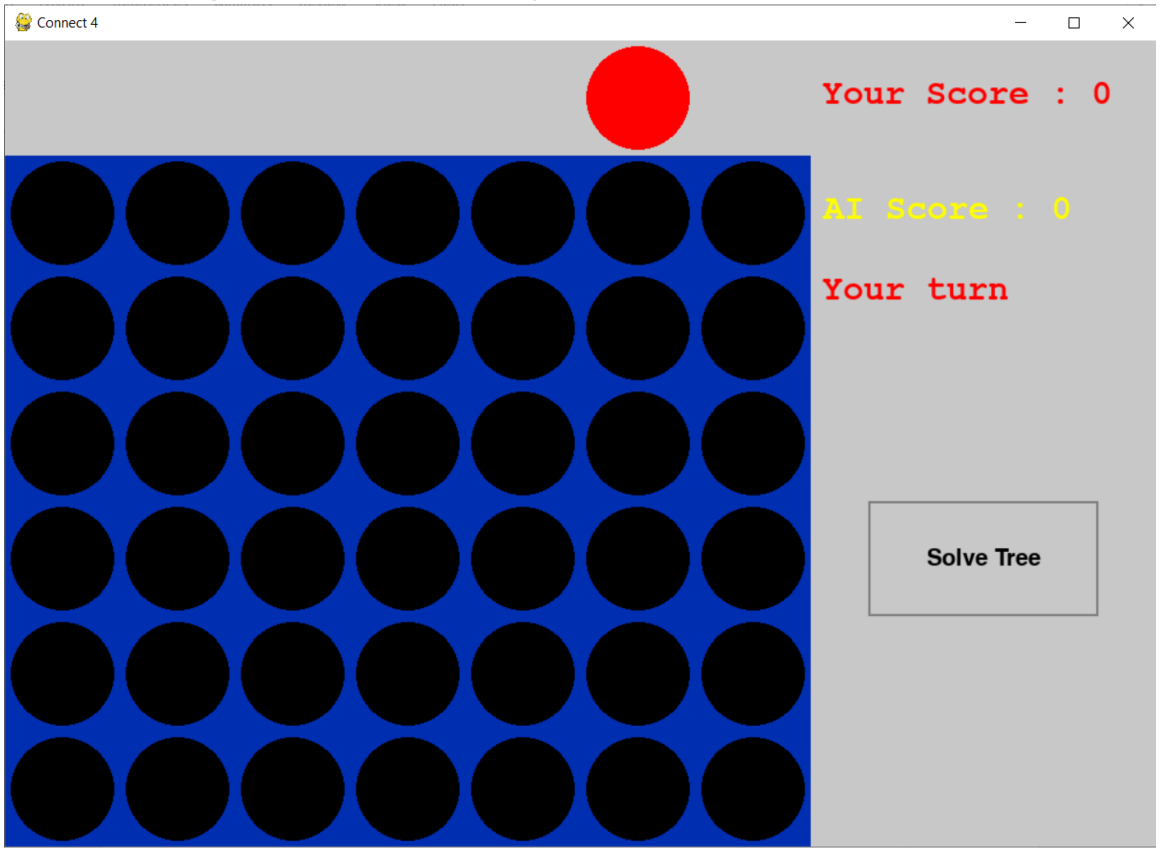


Choose Diffculity:

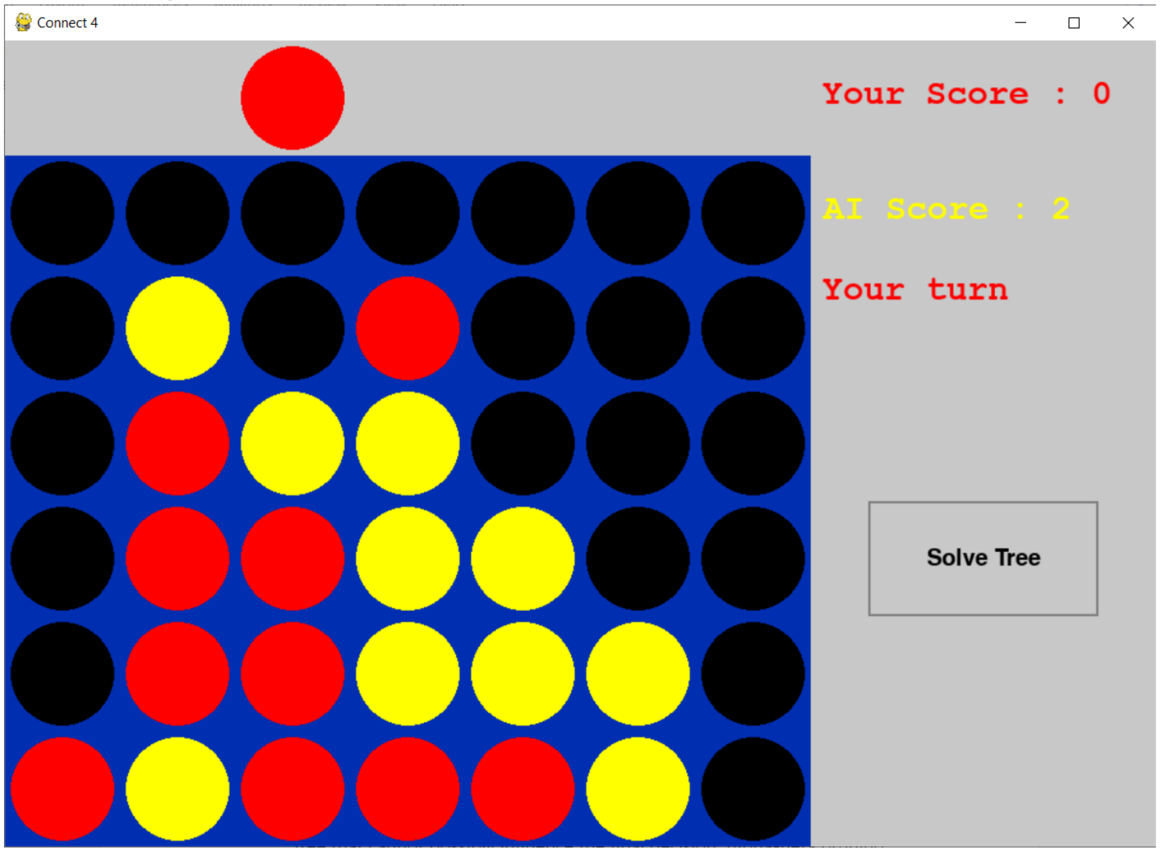
Difficulty: 1

press 'enter' to enter the game

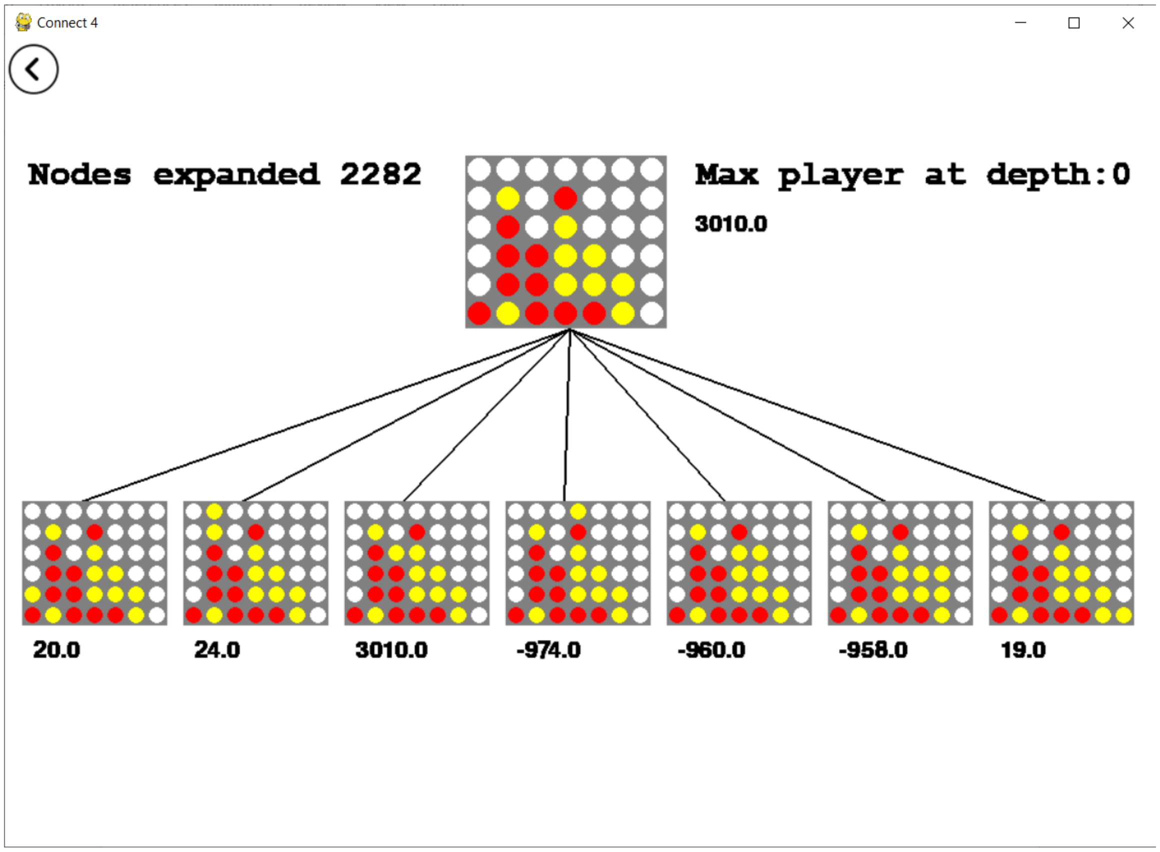
Start of the game with difficulty 4



from the game :



The tree :



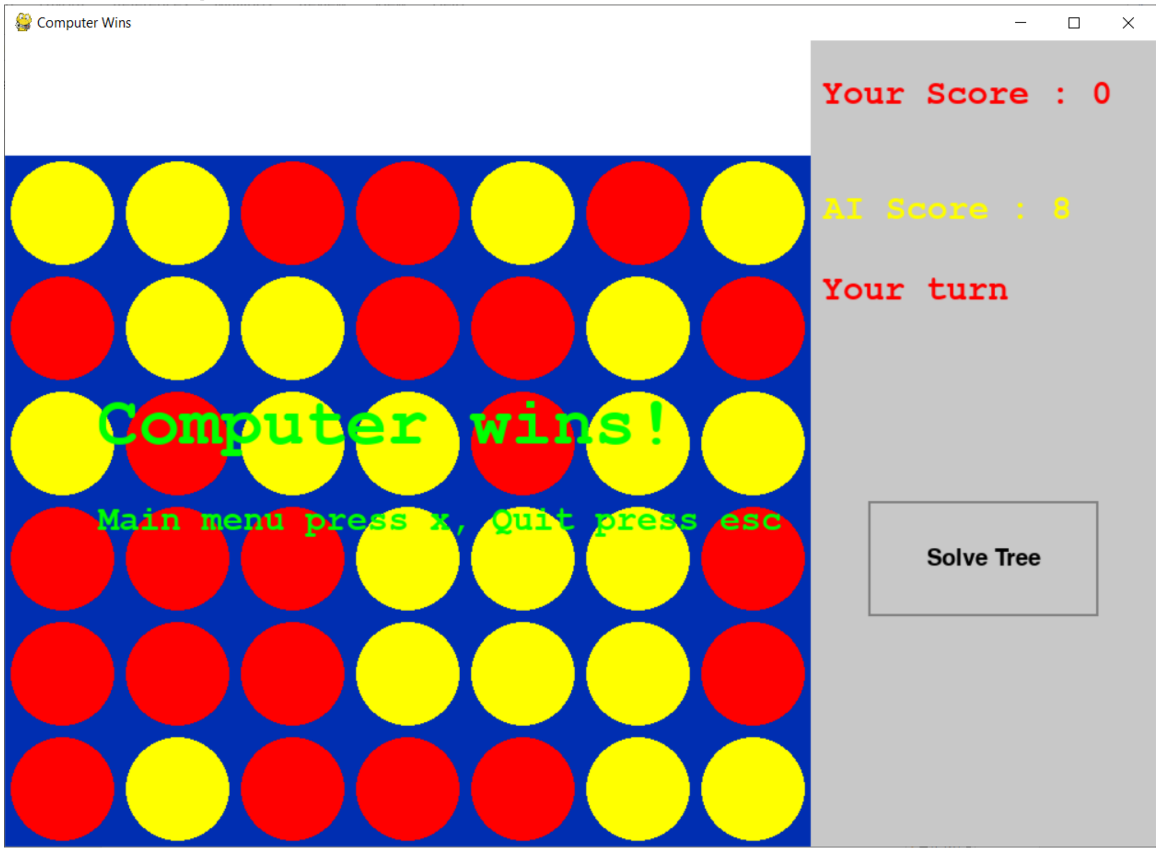
Nodes expanded 2282

Max player at depth:0

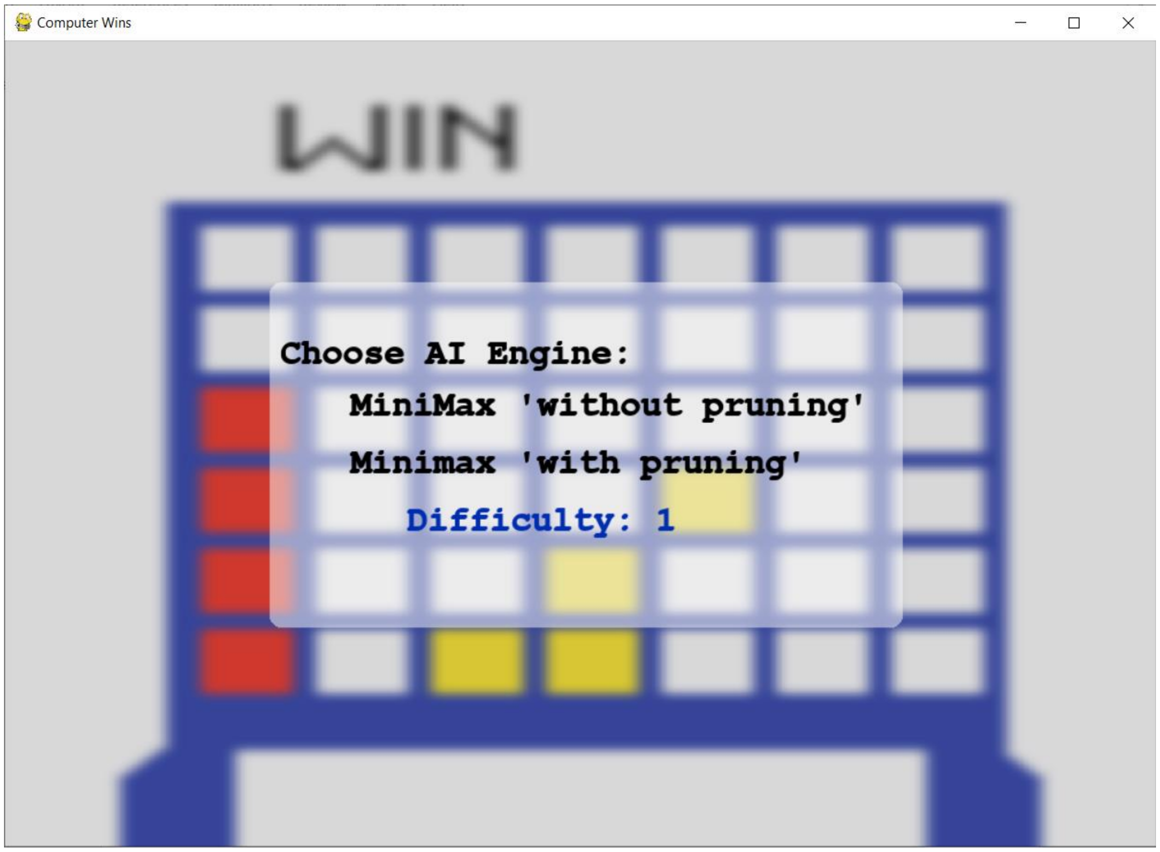
3010.0

20.0 24.0 3010.0 -974.0 -960.0 -958.0 19.0

End of the game :

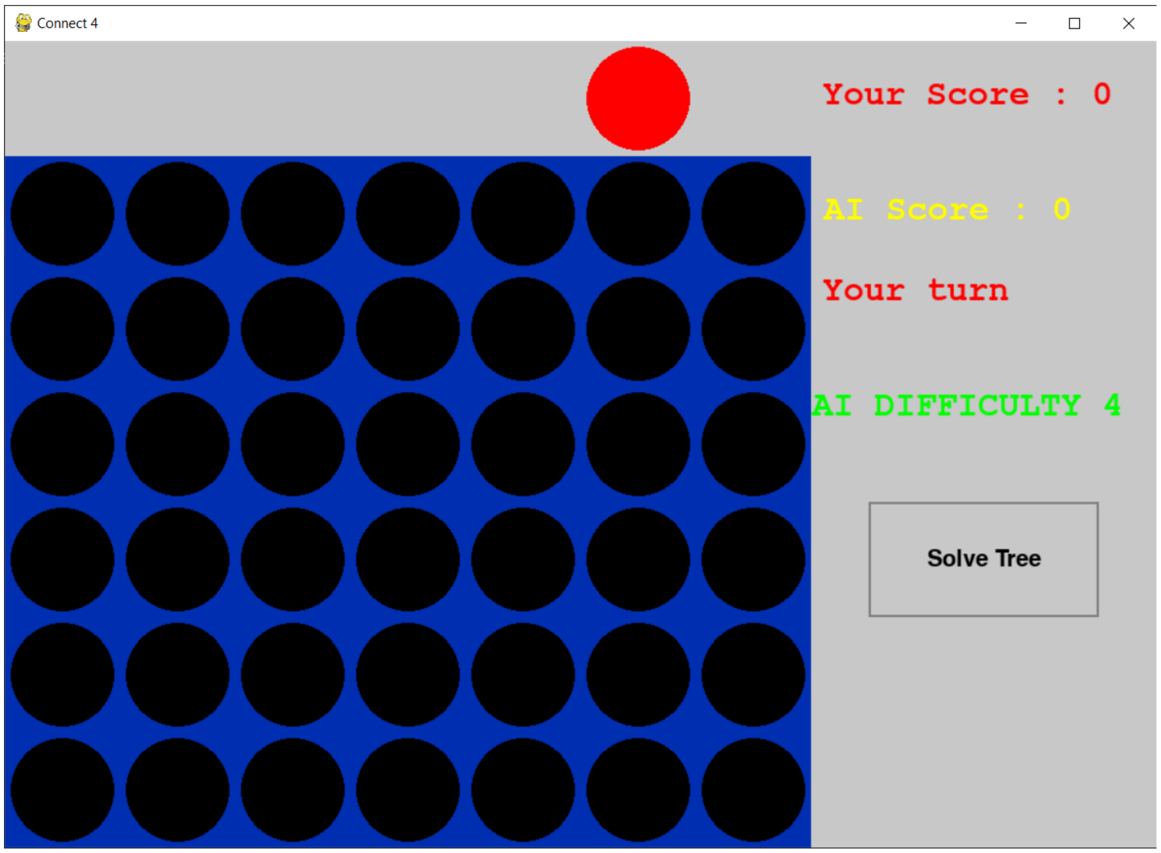


Minimax :

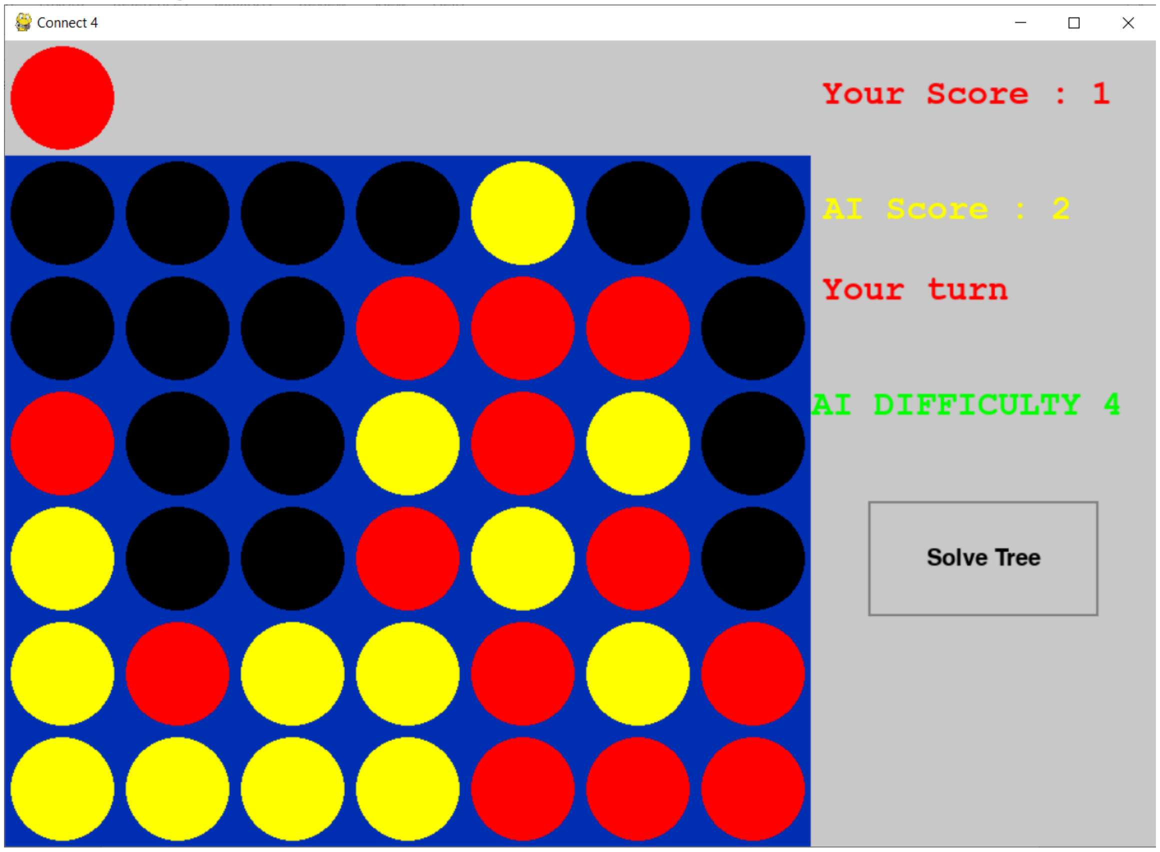

Computer Wins

Choose AI Engine:
MiniMax 'without pruning'
Minimax 'with pruning'
Difficulty: 1

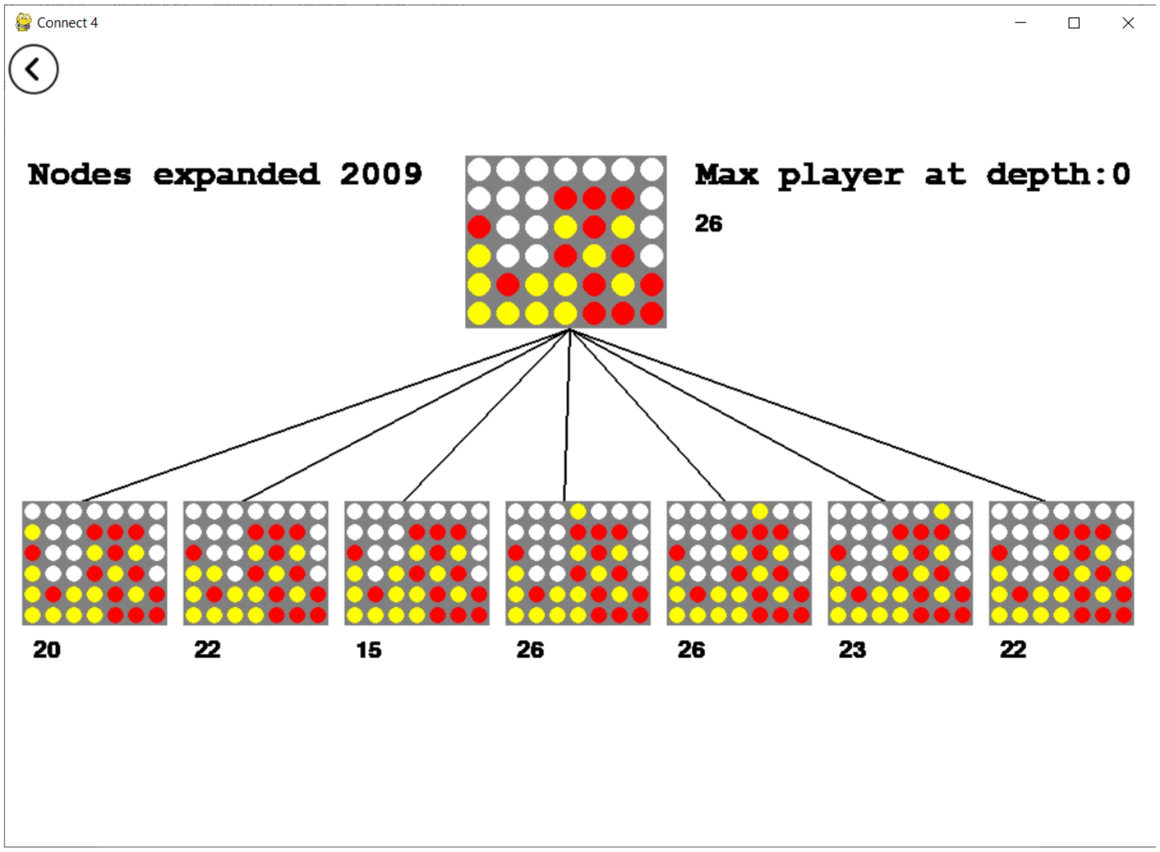
start of the game without pruning with difficulty 4:


Connect 4
Your Score : 0
AI Score : 0
Your turn
AI DIFFICULTY 4
Solve Tree

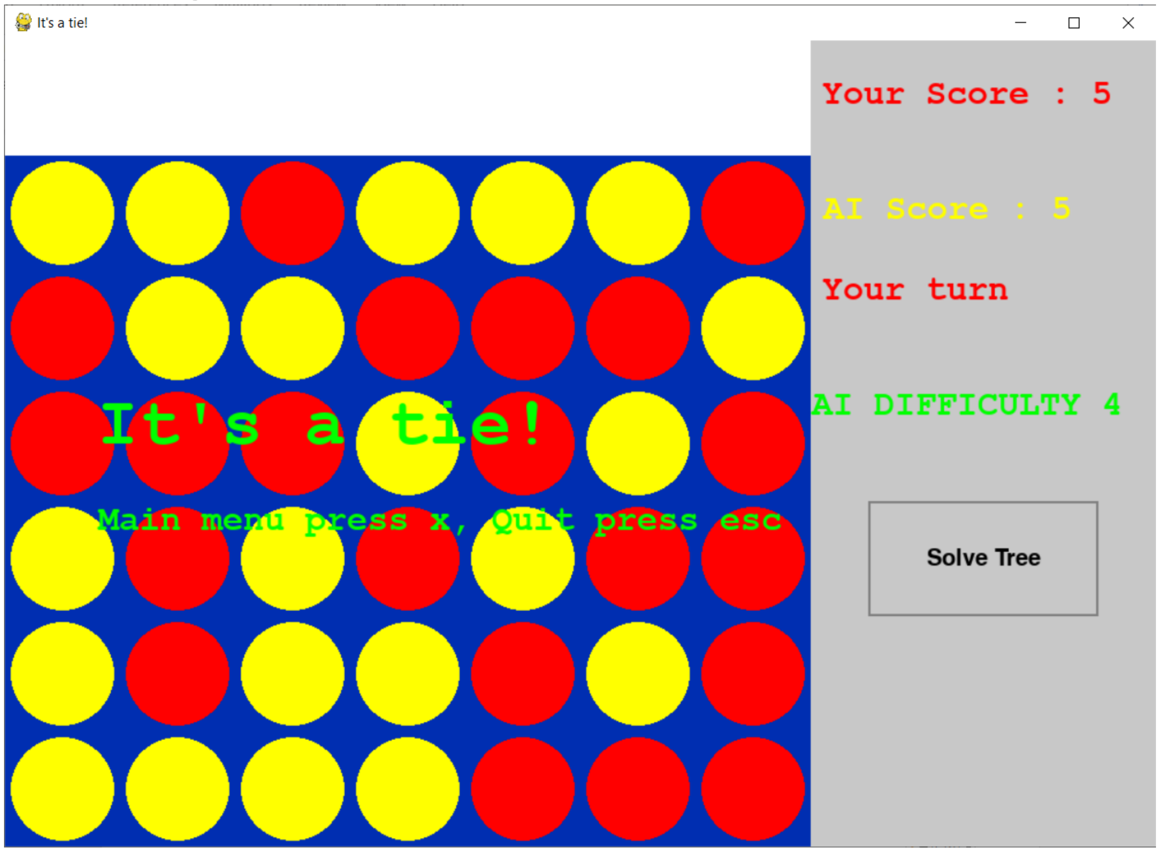
From the game :



The tree:


Connect 4
Nodes expanded 2009
Max player at depth:0
26
20
22
15
26
26
23
22

end of the game :


It's a tie!

Your Score : 5

AI Score : 5

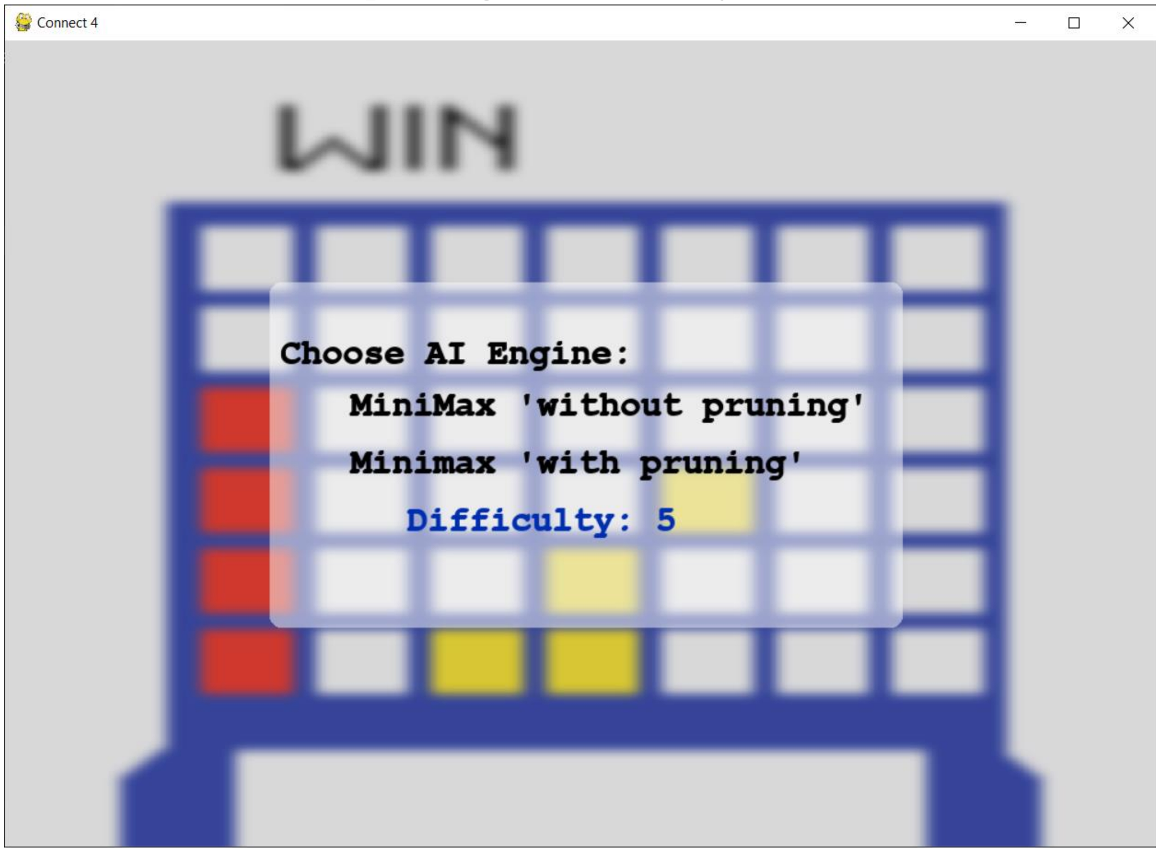
Your turn

AI DIFFICULTY 4

It's a tie!

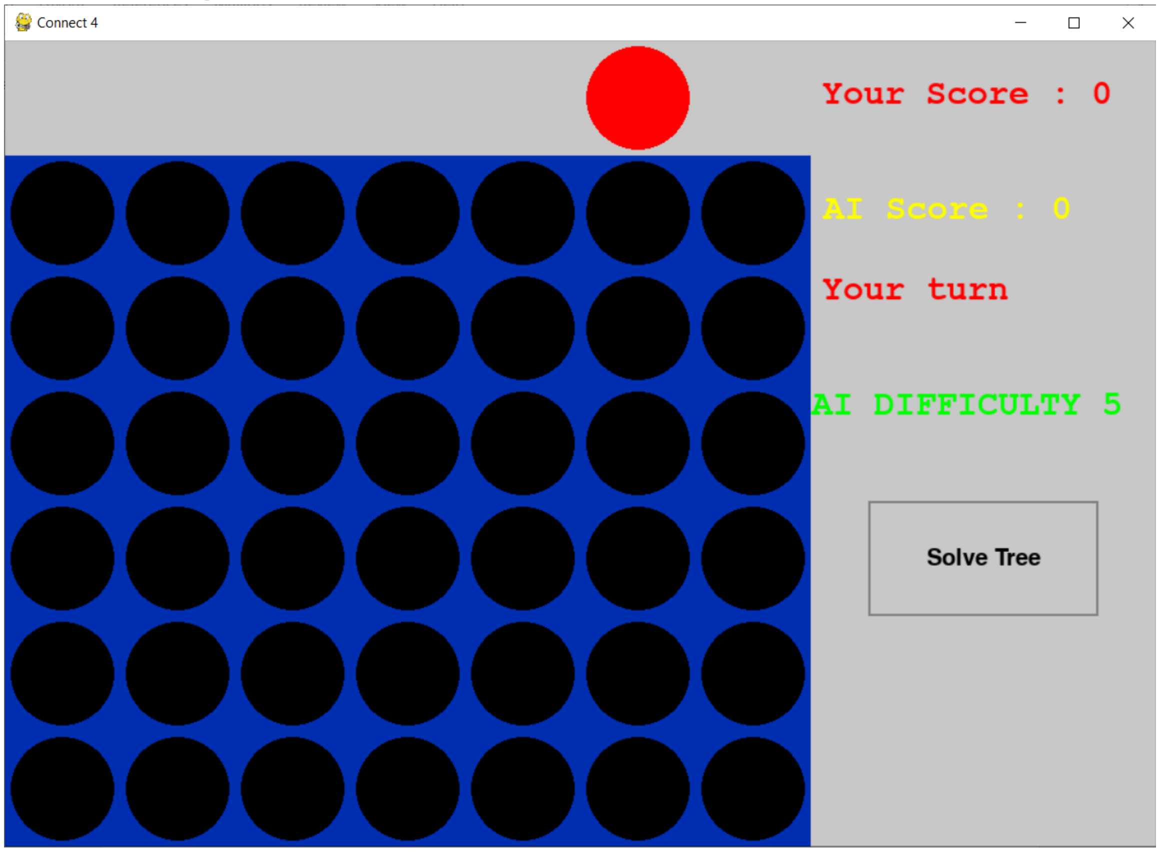
Main menu press x, Quit press esc

Game minimax with pruning and difficulty 5:

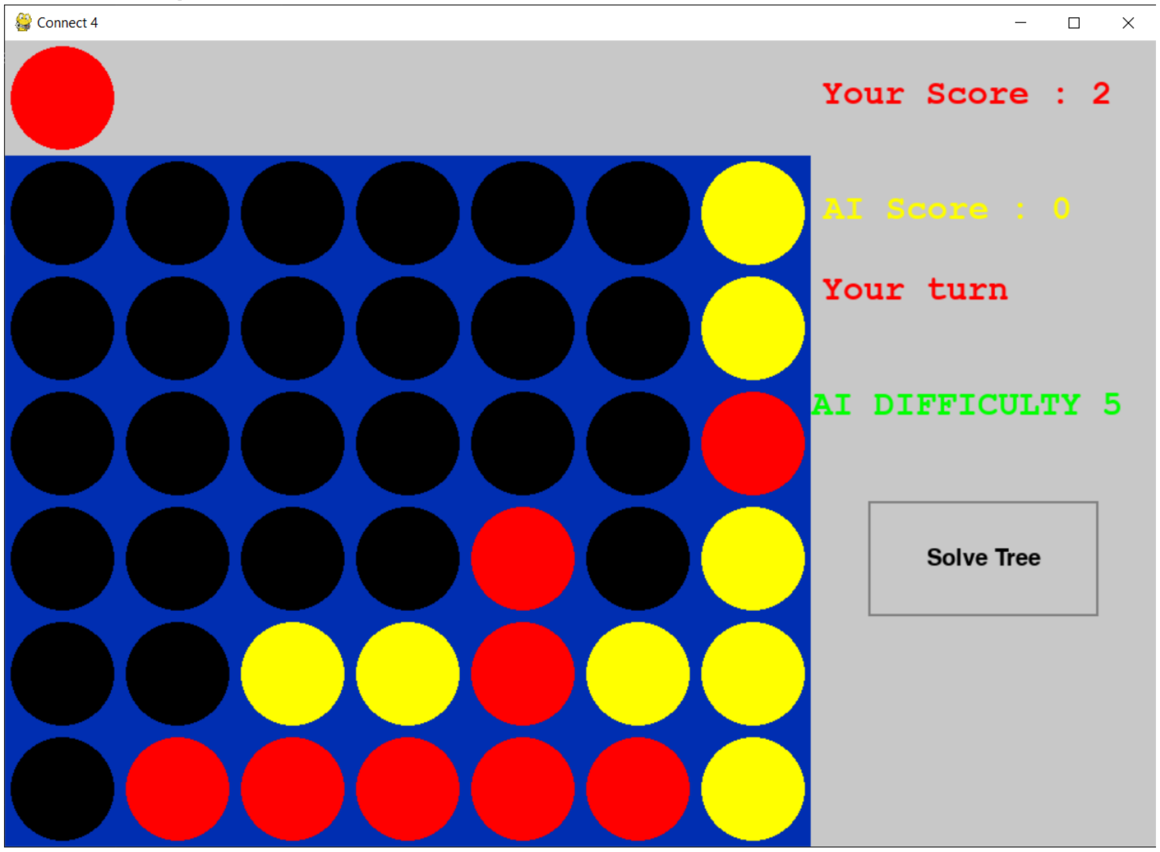


Choose AI Engine:
MiniMax 'without pruning'
Minimax 'with pruning'
Difficulty: 5

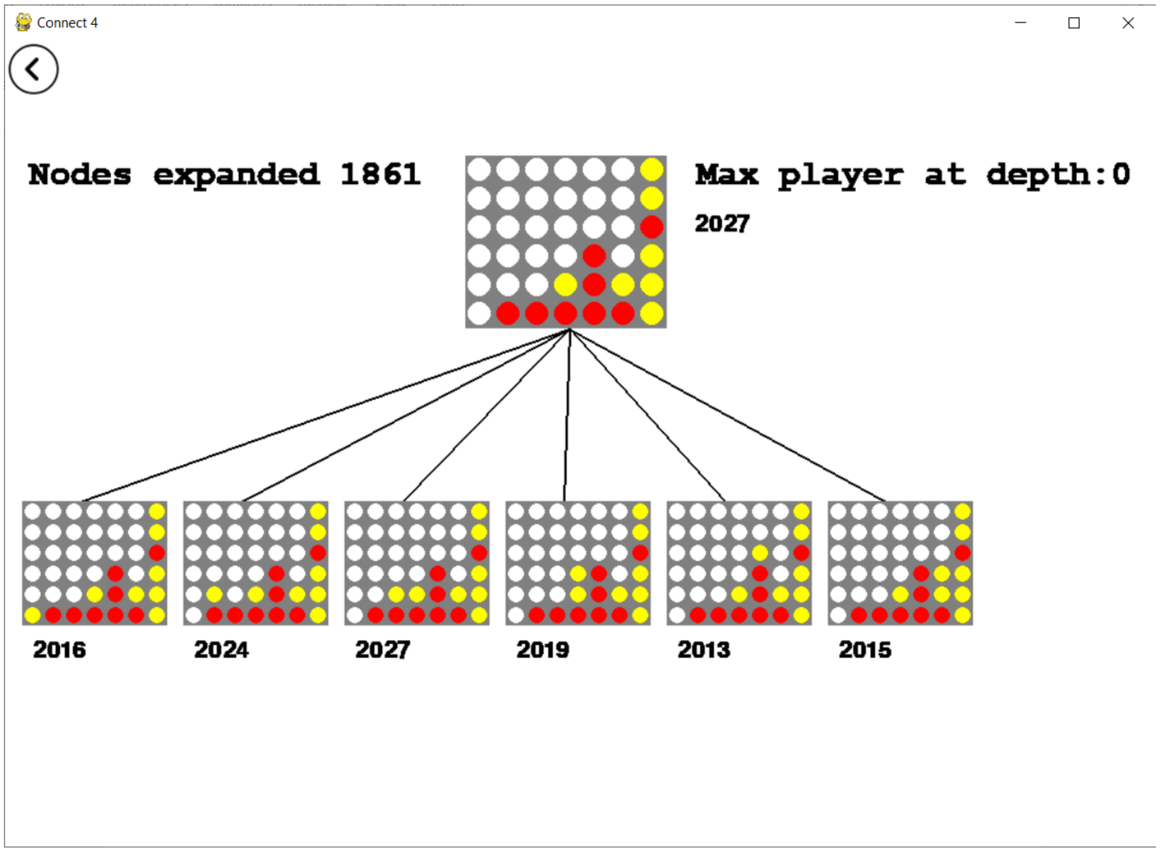
start of the game :



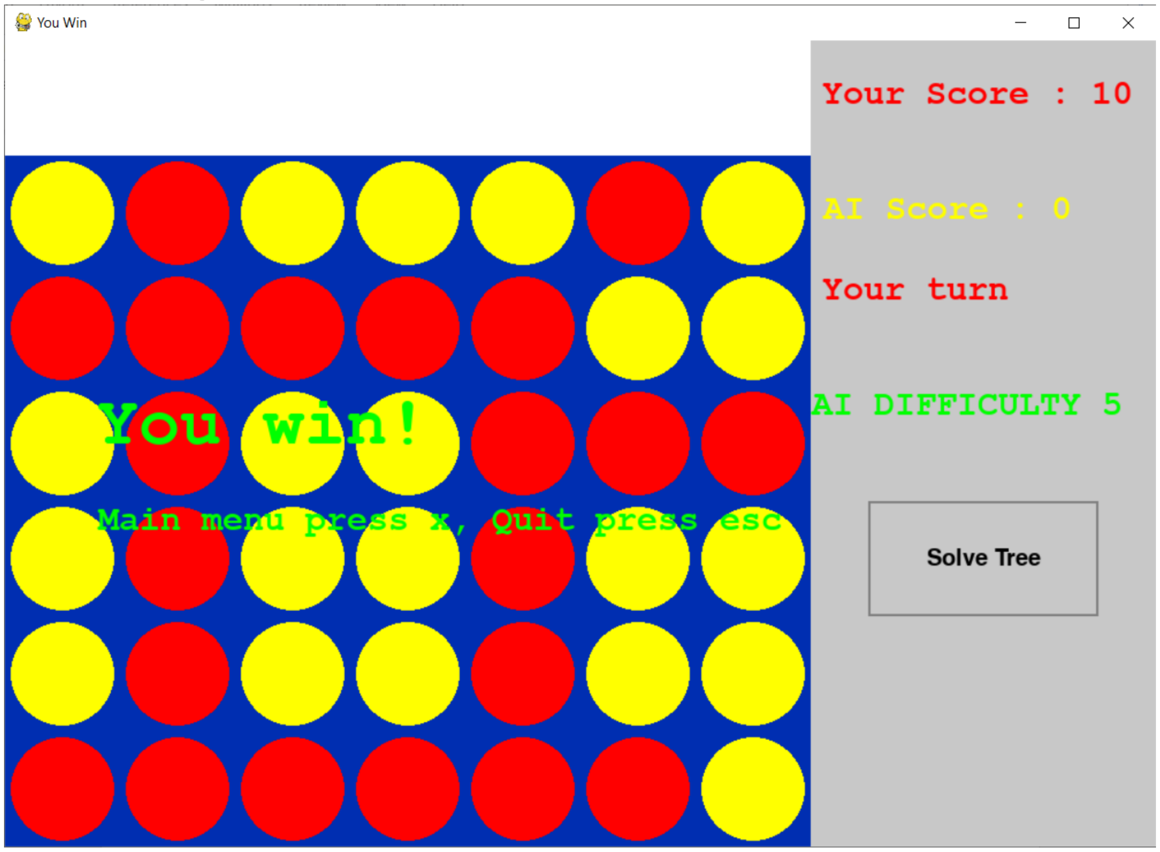
From the game :



The tree :


Connect 4
Nodes expanded 1861
Max player at depth:0
2027
2016
2024
2027
2019
2013
2015

End of the game :



1. **Minimax Algorithm**: This classic algorithm is used for decision-making in two-player games. It explores the game tree recursively, considering all possible future moves up to a certain depth. The agent aims to maximize its score while assuming that the opponent also plays optimally.
2. **Minimax Algorithm with Alpha-Beta Pruning**: This variant of the minimax algorithm enhances efficiency by pruning branches of the game tree that cannot possibly influence the final decision. Alpha-Beta pruning significantly reduces the number of nodes evaluated by the minimax algorithm.
3. **Expected Minimax Algorithm**: This algorithm extends the minimax approach by introducing probability distributions over possible moves. It calculates the expected value of each move considering the probabilities of different outcomes.

Data Structures Used

1. **Dictionary**: Dictionaries are extensively used to represent game states, game trees, and associated values. Each game state is mapped to a dictionary containing metadata and child states.
2. **Lists**: Lists are used to store valid locations for dropping pieces and for various computations within the algorithms.
3. **Numpy Arrays**: Numpy arrays are employed for efficient manipulation of game grids during score calculation.

heuristic function:

The heuristic function in the Connect Four AI evaluates game states to guide decision-making. It scores potential moves based on key factors:

Winning Conditions: Assigns a high score for configurations with four consecutive pieces, indicating an imminent victory.

Strong Winning Opportunities: Recognizes configurations with three pieces and one free slot as high-scoring moves.

Potential Winning Connections: Awards moderate scores for configurations with two pieces and two free slots nearby, suggesting potential winning connections.

Defensive Measures: Prioritizes blocking opponent's winning configurations by assigning high scores to defensive moves.

Counteracting Opponent's Moves: Penalizes configurations indicating the opponent's imminent victory to prioritize defensive actions.

Defense Against Two-In-A-Row Configurations: Penalizes configurations where the opponent has two consecutive pieces with two free slots nearby to prevent immediate defeat.

This heuristic function enables the AI to balance offensive and defensive strategies effectively, contributing to its competitive performance.