|  |
| --- |
| Assignment2 |
| Connect-4 |
| Report |

|  |
| --- |
| Abdelrahman Amr Salah 8016  Ahmed Mahmoud Aly 8062  Youssef Mohamed Attia 8164 |

Overview:

our project is multi-module project consists of Solver, Game-window ,Test, tree rep. , Analyse modules as they imports solver module which contain needed functions and classes

Data Structures Used:

1. **2D NumPy Array (np.array):**
   * The board is represented as a 2D array, where each cell can hold a value representing an empty space, the player's piece, or the AI's piece.
   * Example: board.current\_state is a 2D NumPy array representing the current board configuration.
2. **List:**
   * valid\_columns (in get\_neighbors): A list that holds the indices of columns where pieces can be dropped.
   * columns (in get\_cols): A list of columns neighboring the current column, used for probabilistic moves in the ExpectiMiniMax algorithm.
   * probability\_distribution (in get\_cols): A list representing the probabilities associated with choosing each neighboring column.
   * cols (in many methods): Holds potential column indices to place the next piece.
3. ** Tuples:**
   * Return values: Multiple methods return tuples, where the first element is a column index, and the second is the evaluation score (e.g., col, value).

A screen shot of a computer program

Description automatically generatedSolver code Breakdown:

1. **\_\_init\_\_ Method:**
   1. Initializes the solver with a maximum search depth (depth), the AI's piece identifier (Ai\_piece), and the player's piece identifier (player\_piece).
   2. self.max\_depth: Defines how deep the AI will search for optimal moves.
   3. self.ai\_piece and self.player\_piece: Used to distinguish between the AI's and the human player's pieces on the board.
2. **solve Method:**
   1. this method selects the appropriate solving algorithm (Minimax, Alpha-Beta Pruning, or ExpectiMiniMax) and returns the best column (col) and the associated value (value) for that move.

A computer screen shot of a program code

Description automatically generated

1. **count\_fours Method:**
   1. Counts the number of "four-in-a-row" sequences on the board for a given piece.
   2. This method scans the board horizontally, vertically, and diagonally for four connected pieces that match the given piece.

A computer screen shot of a program

Description automatically generatedA computer screen shot of a program

Description automatically generated

1. **MiniMax Method:**
   1. Implements the basic Minimax algorithm. It recursively explores the game tree, alternating between maximizing (AI) and minimizing (player) at each depth level.
   2. If the maximum depth is reached or there are no available places, the algorithm evaluates the board's state.
   3. Maximizer's Turn (AI): Looks for the move that maximizes the score.
   4. Minimizer's Turn (Player): Looks for the move that minimizes the score.

A computer screen shot of a program code

Description automatically generatedA screen shot of a computer program

Description automatically generated

1. **MiniMax\_alpha\_beta\_pruning Method:**
   1. Similar to Minimax but with alpha-beta pruning, which cuts off branches in the search tree that are not likely to affect the final decision. This speeds up the process by reducing the number of nodes evaluated.
   2. Alpha: Represents the best score that the maximizer can guarantee.
   3. Beta: Represents the best score that the minimizer can guarantee.
   4. Pruning: If at any point alpha is greater than or equal to beta, further exploration of that branch is stopped (pruned).

A computer screen shot of a program code

Description automatically generatedA computer screen shot of a program

Description automatically generated

1. **ExpectiMiniMax Method:**
   1. An extension of the Minimax algorithm incorporating probabilistic outcomes, which is useful in games with elements of chance.
   2. The AI considers not only the immediate moves but also the potential responses of the player and their probabilities.
   3. The method calculates an "expected value" instead of a fixed best or worst outcome

A screen shot of a computer program

Description automatically generated

A computer screen shot of a program

Description automatically generated

1. **evaluate\_board Method:**
   1. Evaluates the board state to provide a score that reflects the AI's advantage.
   2. If the board is full (board.available\_places == 0), it counts the "four-in-a-row" sequences for both players and returns the score accordingly.
   3. If not full, it calculates the score based on the AI's piece position on the board.
2. **get\_neighbors Method:**
   1. Returns a list of valid columns where a piece can be placed, shuffled to add randomness.
3. **get\_cols Method:**
   1. Provides neighboring columns and their associated probabilities, used in the ExpectiMiniMax method to model uncertainty.

Sample runs:

Comparison: