CONNECT4 GAME

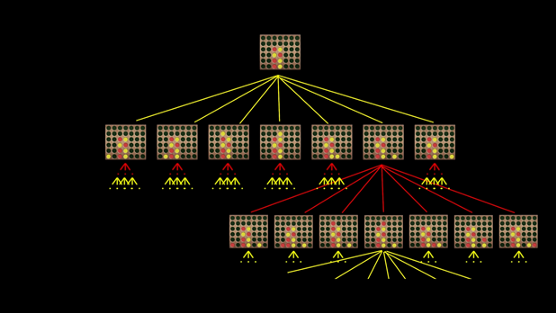
# GAME DESCRIPTION

This is a centuries-old game even played by Captain James Cook with his officers on his long voyages. Milton Bradley (now owned by Hasbro) published a version of this game called **“Connect Four”** in 1974. It is also called **“Four-in-a-Row”** and **“Plot Four”**. This game is played by two players on an upright board which has six rows and seven columns of empty holes. Each player has equal number of pieces (21) initially to drop one at a time from the top of the board. They will take turn to play and whoever makes a straight line either vertically, horizontally, or diagonally wins.

# Decision tree in Connect Four

When the game begins, the first player gets to choose one column among seven to place the colored disc. There are 7 columns in total so there are 7 branches of a decision tree each time. After the first player makes a move, the second player could then choose one column out of seven continuing from the first player’s choice of the decision tree.

Notice that the decision tree continues with some special cases. First, if both players choose the same column for 6 times in total, that column is no longer available for either player. It means that their branches of choices are reduced by one. Second, when both players make all choices (42 in this case) and there is still no 4 discs in a row, the game ends as a draw and decision tree stops. Finally, if any player makes 4 in a row, decision tree stops and the game ends.



# INIT FUNCTION:

This function is used to save all the variables we use in Connect4.

**Time & Space Complexity:**

The time complexity of this function is O (1) and the space complexity is also O (1).

# CREATE BOARD FUNCTION:

In this function we use built in function of numpy module that is np.zeros () and this function returns a new array of given shape and type, where the element's value as 0.

**Time & Space Complexity:**

# DROP PIECE FUNCTION:

This function drops the piece on the place where user want.

**Time & Space Complexity:**

The time complexity of this function is O (1) and the space complexity is also O (1).

# IS\_VALID\_LOCATION FUNCTION:

This function checks whether the piece drop on the empty place or the place is already filled. It returns True when the piece drop on the empty place and returns False when the place is already filled.

**Time & Space Complexity:**

The time complexity of this function is O (1) and space complexity is also O (1).

# GET\_NEXT\_OPEN\_ROW FUNCTION:

This function takes board and column number as arguments and traverses through every row of the argument column to check empty row available in the column and then returns that row number.

**Time & Space Complexity:**

The time complexity of this function is O (r) and the space complexity is O (1).

# WINNING MOVE FUNCTION:

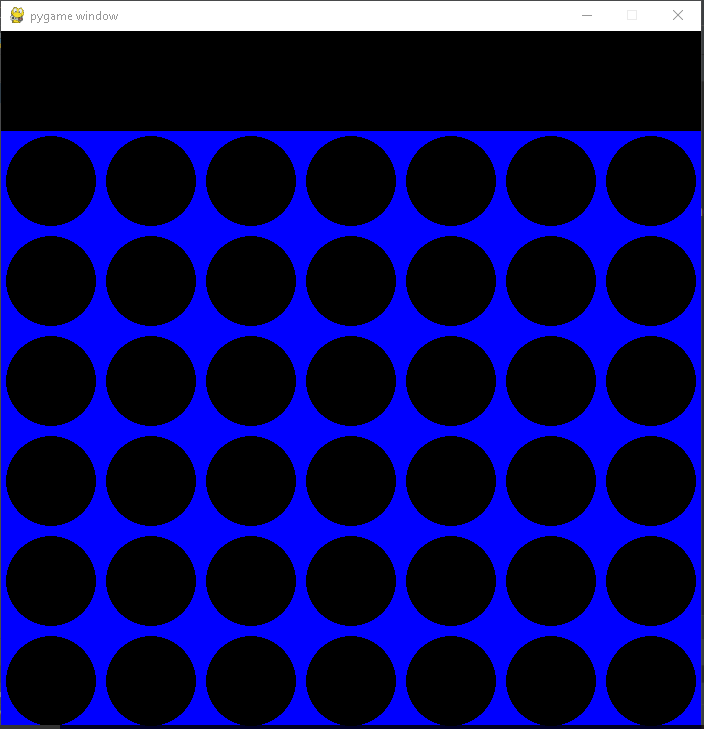
This function checks the winning condition that is when a player takes 4 consecutive circles in vertical, horizontal, positively sloped diagonal or negatively sloped diagonal direction.

# DRAW\_BOARD FUNCTION:

This function uses built in functions of pygame module like pygame.draw.rect () to draw a rectangle on the interface on the screen and pygame.draw.circle () to draw circles.

**Time & Space Complexity:**

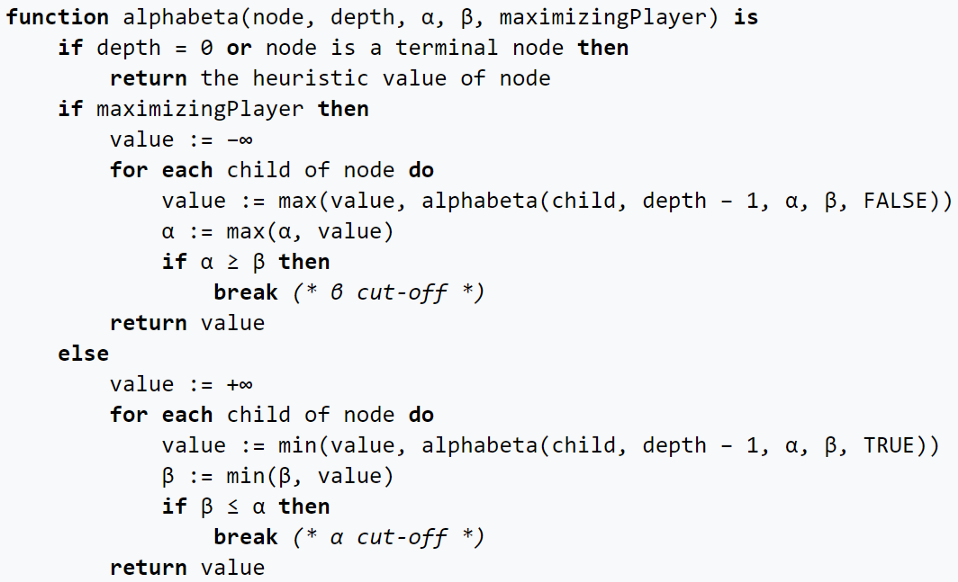
The time complexity of this function is O(rc) and the space complexity is O(1).



# MINIMAX ALGORITHM

Minimax algorithm is a recursive algorithm which is used in decision-making and game theory especially in AI game. It provides optimal moves for the player assuming that the opponent is also playing optimally. Considering two opponents: Max and Min playing. Max will try to maximize the value while Min will choose whatever value that is the minimum. The algorithm performs a depth-first search (DFS) which means it will explore the complete game tree as deep as possible, all the way down to the leaf nodes.

Below is a python snippet of Minimax algorithm with Alpha-beta pruning strategy to improve the computational speed and to save memory. The figure below is a pseudocode for the alpha-beta minima algorithm.



In our case maximizing player is AI and minimizing player is human player.

Minimax algorithm takes board, the depth of the tree to pick the best score and column, alpha beta values which are negative infinity and positive infinity in our case and the maximizing player which is AI player

Terminal node is when no more pieces are left to drop in the board or when the AI player or human player wins the game.

Alpha chooses the maximum value among the child nodes of parent ‘maximizing player’ and Beta chooses minimum value among the child nodes of parent ‘minimizing player’, in this way we don’t have to visit each and every child node of maximizing or minimizing player.

The minimax algorithm returns the best score of the best column to drop the piece.

**Time & Space Complexity:**

**Time complexity** of minimax is **O(b^m**) and the **space complexity** is **O(bm)**, where b is the number of possible moves to drop a piece in the board, which is 7 in our case and m is the maximum depth of the tree and in our case depth of the tree is 5.

# TEST CASE RUNS

# FUTURE EXTENSIONS

## ADD USER REGISTRATION KEEP RECORD OF AI BEATER

One of the future extension includes registration window which takes user name and password and save it in database. Also keep track of the user who beats the AI in connect four game.

## USER CHOICE TO REPLAY THE GAME

A user choice to replay or quit the game will be included.

## MAKE BOARD WITH VARIABLE ROW AND COLUMN

A window which takes user input to create the board with user input row and column.

## DIFFICULTY LEVEL OF AI

Include three options for the user to choose the difficulty level of AI (easy, medium and hard).