Connect 4 - OVERVIEW

The rules are simple: Try to build a row of four checkers while keeping your opponent from doing the same. Sounds easy, but it’s not! The vertical strategy creates a unique challenge: you must think in a whole new way to block your Opponent’s moves!

CONTENTS

1) grid

2) 21 black checkers

3) 2 end supports

4) 21 red checkers

OBJECTIVE

Be the first player to get four of your colored checkers in a row— horizontally, vertically or diagonally.

HOW TO SET UP

1. Assemble the game .
2. Make sure the sliding lever is positioned so that the bars pre- vent the checkers from falling through the grid.
3. Place the game between you and your opponent.
4. Take all the checkers of one color. Your opponent takes all the other color checkers.

HOW TO PLAY

1. Decide who plays first. Players will alternate turns after playing a checker. NOTE: The player starting the first game will play second in the next game.

2. On your turn, drop one of your checkers down ANY of the slots in the top of the grid.

3. Play alternates until one player gets FOUR checkers of his or her color in a row. The four in a row can be horizontal, vertical or diagonal.

HOW TO WIN

If you’re the first player to get four of your checkers in a row, you win the game!

TO START ANOTHER GAME

First, clear the board of checkers by simply sliding the lever at the base of the grid to one side. The checkers will drop out, and you are ready to start the next game. Be sure to slide the lever back to its original position.

Connect 4 – Code Design

The game consists of a board , two players and the discs.

My implementation consists of three python codes namely board.py , game.py and player.py

Board.py

This script generates the board of the connect 4 game.

The board consists of 6 rows and 7 columns making a total of 42 empty slots.

This is also responsible for determining if player 1 is the winner or player 2 is the winner or if it ends in a draw.

It puts a disc in an appropriate place and checks to see if it results in a winning move or not.

It also generates a list of valid children to fill up the next place.

It also has a hash function which generates values based on the position of the discs.

Function isTerminal : Determines if the game is over or not.

Function makeMove : Places a piece in an empty slot.

Function children: Creates a valid children.

Function isFull : Determines if the board is full or not (42 slots are filled). This is called each time a new piece needs to be placed to check if the board is full or not.

Game.py

This is used to simulate a game and choose the winner.

It sets up the board and starts the game based on the choice of out algorithm.

Function simulateLocalGame : This function creates an instance of the game (i.e.) it initializes the board and starts the game. It also declares the winner or if the game has ended in a draw.

Player.py

This is used to generate the player's move.

The algorithms are:

1) **Minimax algorithm:**

This algorithm is used since it is an adversarial search algorithm. It tries to maximize its chances of winning the game at the same time decreasing the opponents chances of winning the game. This can be used as any one of the players method of playing or both the players method of playing.

Function miniMax:

Checks if the game is still going on or if it has finished.

Implements the minimax algorithm

Creates a list of valid children to populate the board.

Returns the best score for the player.

2) **Alpha-beta Pruning**

This algorithm is a variation of the minimax algorithm and also it is an adversarial search algorithm. This is computationally fast than the minimax algorithm since it “prunes” some of the tree’s children and removes unwanted computations.

It also has a heuristic function which returns the heuristic values. Good positions for 0 pieces are positive and good positions for 1 pieces are negative.

Function alphabeta:

This function implements the alpha-beta pruning algorithm.

Creates a list of valid children to populate the board.

Returns the best score for the player

The player can either use the Alpha-beta pruning algorithm or the minimax algorithm for playing the game.

3) **Cutting off search**

   Cut off search at a certain depth and compute the value of an evaluation function for a state instead of its minimax value
   The evaluation function may be thought of as the probability of winning from a given state or the expected value of that state.

   Function cutoff:
   This function implements the cutting off search algorithm.

Returns the score at a certain depth.
if CUTOFF-TEST(state, depth) then return EVAL(state)

When the game reaches a certain depth it returns the heuristics for the depth and calculates

Major Data Structures:

The data structures which are being used include arrays , lists.
Board is an array of 42 empty slots.
numMoves is an array which holds the number of moves it takes to reach the goal state.

PseudoCode:

For row strike:

```
Dim desiredColor As ???
Dim thisRow As Integer = ???
Dim totalAdjacent As Integer = 0
For column As Integer = 0 To NumberOfColumns - 1
If GetColor(thisRow, column) = desiredColor Then
totalAdjacent += 1
Else
totalAdjacent = 0
End If
If totalAdjacent = 4 Then
' Winner!
End If
Next
' Not a winner.
```

For column strike:

```
Dim desiredColor As ???
Dim thisColumn As Integer = ???
Dim totalAdjacent As Integer = 0
For row As Integer = 0 To NumberOfRows - 1
If GetColor(thisColumn, row) = desiredColor Then
totalAdjacent += 1
Else
totalAdjacent = 0
End If
If totalAdjacent = 4 Then
```

```
' Winner!
End If
Next
' Not a winner.
```

For Diagonal Strike:

```
Dim desiredColor As ???
Dim startRow As Integer = ???
Dim startColumn As Integer = ???
Dim totalAdjacent As Integer = 0
Dim thisRow As Integer = startRow
Dim thisColumn As Integer = startColumn
While thisRow < TotalRows AndAlso startColumn < TotalColumns
If GetColor(thisRow, column) = desiredColor Then
totalAdjacent += 1
Else
totalAdjacent = 0
End If
If totalAdjacent = 4 Then
' Winner!
End If
thisRow += 1
thisColumn -= 1
While thisRow < TotalRows AndAlso startColumn < TotalColumns
If GetColor(thisRow, column) = desiredColor Then
totalAdjacent += 1
Else
totalAdjacent = 0
End If
If totalAdjacent = 4 Then
' Winner!
End If
thisRow -= 1
thisColumn += 1
End While
' Not a winner!
```