CS 152 Computer Programming Fundamentals
Lab 6: Connect Four*

Brooke Chenoweth

Fall 2019

1 Game Rules

Players: 2, red and black (or some other colors). In our program, the computer will play black and the human will play red.

Object: To be the first to connect four of one's own discs of the same color next to each other vertically, horizontally, or diagonally.

Play: Players take turns dropping colored discs from the top into a seven-column, six-row vertically suspended grid. The pieces fall straight down, occupying the next available space within the column.

2 Program Description

I have provided you with three files of code for this assignment.

- `ConnectFour.java` contains the constants and methods for the connect four game. There are nine methods that you must complete to be able to play the game. The `main` method in this class will set up the game board and start the GUI.
- `ConnectFourGUI.java` manages the GUI (Graphical User Interface) for the connect four game. It will use the methods in `ConnectFour.java` to determine legal moves, allow the computer player to move, detect a winner, etc. You don't need to understand the code in this class, but feel free to take a look. The only method you need to use is `showGUI`, which is already being called in `ConnectFour`'s `main` method in the code I gave you.
- `ConnectFourTest.java` contains code to test the `ConnectFour.java` methods. If you run this class, it will report how many tests you passed. You may find this especially useful when you have yet to complete enough methods to be able to play the game.

*This lab is based on the project at `http://ljing.org/games/connect-4/`

2.1 Constants

You should use the constants defined at the beginning of the `ConnectFour` class instead of hard-coding certain integer and color values.

The integer constants `ROWS` and `COLUMNS` define the size of the game board. Do not change them, since it may break the GUI and/or test code. You may assume that any game board array used in the methods for this assignment will have ROWS rows and COLUMNS columns.

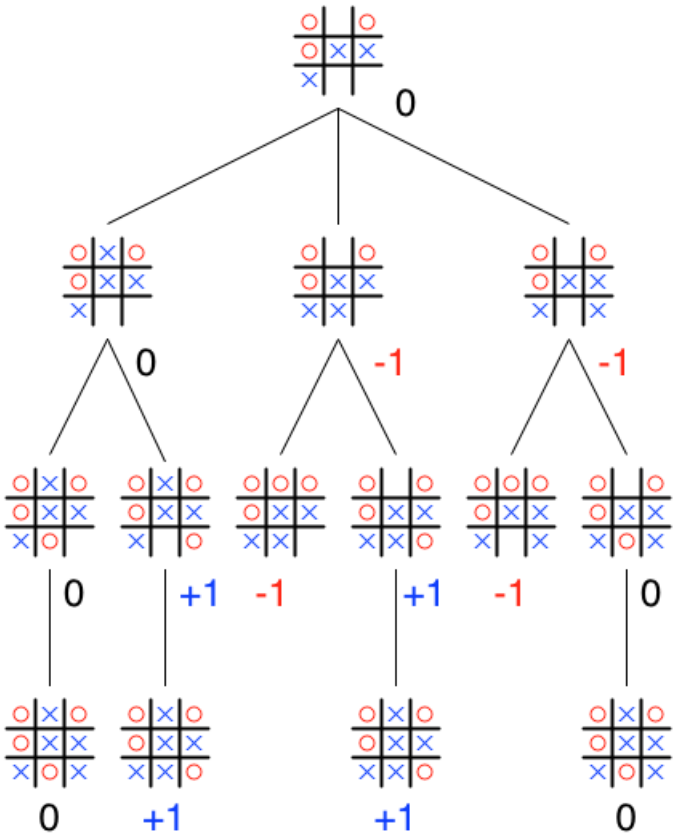
The constants `HUMAN`, `COMPUTER`, and `NONE` represent characters for human player's pieces, computer players pieces, and empty spaces, respectively.

2.2 Board Representation

The board is represented as a two-dimensional array of `char`s, with each entry representing either the a human player's piece, a computer player's piece, or an empty location. The first row in the array (that is, row 0) is at the bottom of the game board. Take care not to accidentally drop pieces upside down!

2.3 Computer Player: Minimax Search Algorithm

How does the computer choose its moves? To answer this, consider the game as a tree. To simplify the explanation, here is part of the game tree for Tic-Tac-Toe:



Each node in the tree represents a state of the game. At the bottom of the tree, the number associated with a node indicates who, if anyone, has won: +1 for X, -1 for O, and 0 for a tie. Higher up, the number indicates who would win if both players played perfectly. This can be determined in terms of the nodes directly below. For example, in the center node in the second row, O is to move and has two choices. Since O is trying to minimize the score, she chooses the move to the left, winning the game. We can therefore say that the node in question has a value of -1. By similar reasoning, X is trying to maximize the score at the top of the tree, so will choose the move with value 0.

We don't actually build this tree as a data structure in memory. Instead, it illustrates a recursive computation:

- If the game is over, determine the value of this game state directly.
- Otherwise, recursively determine the value of each state reachable in one move; the value of this state is the max (if X is to move) or min (if O is to move) of these states.

This is the *minimax* algorithm. In a very simple game like Tic-Tac-Toe, a computer can quickly explore the entire game tree this way. For deep games like Chess and Go, this would take more time than the expected lifespan of the universe. Connect 4 is somewhere in between.

Since we can't read all the way to the bottom of the tree, we stop after a certain number of levels; if the game isn't over at that point, we count it as a tie. Our program won't play perfectly, but it's a challenge against beginning opponents. (If you want a smarter opponent and are willing to wait longer for the computer's turns, you can pass a larger depth argument when starting the GUI.)

The minimax search algorithm is implemented in three methods. For this assignment, you are given `minScoreForHuman`, which determines the value of the game state when the human (who is trying to minimize the score) is to move. You must write the corresponding method `maxScoreForComputer`. These two methods are *mutually recursive*: they call each other. The third method, `bestMoveForComputer`, is similar to `maxScoreForComputer` but returns the move to make to maximize the score, rather than the resulting value.

3 Turning in your assignment

Submit your `ConnectFour.java` file to the UNM Learn. Do not attach `.class` files or any other files.