CS 2400 – Fall 2016
Assignment 2
Minimax for Connect-4[1]
Due: Monday, 26 September

Note: You may work in groups of two if you want to.

Connect-4

You are probably all familiar with the game of `Connect-4`. Two players take turns dropping disks down the columns of a 2-dimensional grid, trying to get four of their disks in a row (vertically, horizontally, or diagonally). There are many `Connect-4` sites online; for example:

http://www.geocities.com/ResearchTriangle/System/3517/C4/C4Conv.html

In fact, the code you will be using for this assignment comes from this site. **Note that the code can be downloaded, but if you download it and look at it, that would be cheating, because it contains some code segments that I have removed from the program and am asking you to write. And that would mean a trip to the J-Board.**

The Code

The assignment distribution, posted on BlackBoard, contains this handout and a `C4` directory that contains all the code necessary to run the `Connect-4` applet. There are:

- generic game-playing classes,
- `Connect-4` game-playing classes,
- various necessary images, and
- an `html` file to start the game.

You will need to familiarize yourself with the code, but the code is reasonably straightforward and you do not need to look through every line of code to understand and modify it. Focus on the `C4*.java` files, especially `C4Board.java`, and `MinimaxPlayer.java`. In the original program, the latter file contains the code that the computer uses to calculate its next move. This class now contains a skeleton within which you will write code that implements minimax search with and without α-β-pruning. Note that the minimax search depth can be set (in fact, the "level" indicated in the applet is exactly that) and

[1]Based on an assignment developed by Roland Bol at Uppsala Universitet.

the total number of moves the program checked to calculate its move (and the time taken to do that) appears in another window.

The code has some quirks. For example, the method `getPossibleMoves` (in `C4Board.java`) always returns *all* the columns as possible moves, which seems to contradict the method name. Illegal moves (when a column is full) are caught in the `move` method (also in `C4Board.java`), which returns `true` if an attempted move is successful; `false` otherwise. Take note of the fact that the code actually makes a move in order to explore it (`board.move(moves[i])` in the code below), so you must make sure the code to undo it (`board.undoLastMove()`) gets executed at the appropriate place.

```
if(board.move(moves[i]))    // move was legal (column was not full)
   {
       moveCount++;  // global variable

       // ************************************
       //              CODE NEEDED
       //             (mostly here)
       // ************************************

       board.undoLastMove();   // undo exploratory move

   }  //end if move made
```

Also, there are some quirky situations in which the computer plays better at a lower level, which should not be the case since the level indicates the depth of the minimax tree. It is the result of the evaluation function it's using and you can ignore it.

After you have added the required code, compile the game (using `javac`) on the command line of a terminal window:

```
=> javac *.java
```

where `=>` is the prompt. You will get the following warnings (which you can ignore):

```
Note: C4Applet.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
Note: Some input files use unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
```

Run the game at the command line with the command:

```
     appletviewer c4.html
```

The Assignment

In the `MinimaxPlayer.java` file, add clearly-documented code that implements the minimax algorithm. There are three places (each in a different method) flagged by "`CODE NEEDED`" where you need to add code. Note that you may need to make other changes/additions to these methods (e.g. parameters, local variables). I have just indicated where most of the required additional code will go.

Test your code by playing 10 games against the program at each of the 8 levels. The program reports the number of moves explored for each actual computer move and the time, in milliseconds, required to explore those moves. Report the total number of moves explored and time required for each game at each level. Calculate and report the average number of moves explored and time required at each level. Of course, the results will be greatly affected by your Connect-4 prowess. We will ignore this since we're just trying to get an idea of how much α-β-pruning helps.

Add α-β-pruning to your minimax code. Please write your code so that I can easily turn α-β-pruning on and off when testing your code. Conduct the same set of tests and collect the same data that you did for minimax without α-β-pruning.

Compare the performance of minimax with and without α-β-pruning at each level in a brief, well-organized report illustrated with tables *and* graphs. Your report should include:

1. a title,
2. a description of Connect-4,
3. a description of the basic minimax appraoch and how α-β-pruning pruning is incorporated to improve the efficiency,
4. your data,
5. a discussion of your results, and
6. answers to the following questions:
   (a) Explain the heuristic function (look in `C4Stats.java` and `C4Row.java`). What quantities does it compute and how does it weight them? I will leave it to you to figurte out what `(i << j)` means (searching the web or consulting a book is fine).
   (b) Moves are always explored from left to right. There is no reason why they couldn't be explored in some other order. What ordering do you think might be better here? Could the ordering matter in minimax without α-β-pruning? with α-β-pruning? In each case, explain what difference it could make, or why it couldn't make a difference.
   (c) This version of `Connect-4` has 8 columns, while most have 7 columns. This program can be changed to a 7-column version by changing the constant `NUMBER_OF_COLUMNS` in `C4Board.java` to 7 and by changing the unnamed constant 84 on line 97 of that file to another value. What should that value be

and why? Wouldn't it be nice if it were a named constant? Note that the answer to this last question is yes. Note also that the graphics in the 7-column version will still show 8 columns, but that it will not be possible to drop a disk in the last column.

Grading

Your assignment will be graded approximately as follows:

50%: your code (correctness, clarity, etc.), and

50%: your report (completeness, clarity, use of tables and graphs, answers to questions).

Submitting Your Work

When you have completed the assignment, you should:

1. Upload a folder to Blackboard that contains all the files I need to run your program, including a `README` file with clear instructions about how to run your program with and without α-β-pruning. The folder should also contain an electronic copy of your report. You can upload it in the `Assignments` section on BlackBoard.
2. Please turn in a hard copy of your report as well.