Connect Four Artificial Intelligence (AI)

Preamble

This assignment introduces you to a core Artificial Intelligence (AI) concept that utilizes trees. In this assignment, you will implement a tree that can determine the best possible move given a game board. Please, read the whole handout before starting as there are some new concepts you have not seen before.

Learning Objectives

- Achieve proficiency with using trees and recursion
- Learn about the minimax algorithm
- Develop an awareness for how computationally infeasible it may be to enumerate all possible paths of a game

Collaboration policy and academic integrity

You may do this assignment with one other person. Both members of the group should get on CMS and do what is required to form a group well before the assignment due date. Both must do something to form the group: one proposes, the other accepts.

People in a group must *work together*. It is against the rules for one person to do some programming on this assignment without the other person sitting nearby and helping. Take turns "driving", using the keyboard and mouse.

With the exception of your CMS-registered group partner, you may not look at anyone else's code, in any form, or show your code to anyone else (except the course staff), in any form. This includes having a public repository for your code, like github.

Getting help

If you don't know where to start, if you don't understand testing, if you are lost, etc., please *see someone immediately* - an instructor, a TA, a consultant. Do not wait. A little in-person help can do wonders. See the course homepage for contact information and the course calendar for updates on office hours

Connect Four: The Game

The purpose of A4 is to create an AI program that can masterfully play Connect Four. Connect Four is a two-player game in which the two players take turns dropping colored discs from the top into a 7-column, 6-row vertically suspended grid. The pieces fall straight down, occupying the next available space within the column. The object of the game is to connect four of one's own discs (of the same color) next to each other vertically, horizontally, or diagonally before your opponent does.
(Source: http://en.wikipedia.org/wiki/Connect_Four)

Your job

There are two parts to this assignment. First, you have to familiarize yourself with the classes we give you. That includes writing two methods so that the game can be played.

Then, two humans can play, the computer can play against itself using a dumb method that plays randomly, or a human can play against the dumb method.

The second part consists of writing methods that implement an AI (artificial intelligence) technique called the minimax algorithm.

We start by explaining what you have to do for the first part. The AI part follows, and you would be wise to wait until you have completed the first part to even look at the second part.

Part 1: Getting the game to play —writing two methods

1. Download the A4 zip file from either the CMS or the course website, start a new project named A4 in Eclipse, and import the zip file into it.
2. Select class GUI in the Package Explorer and Run the program (Menu item Run -> Run). A GUI should appear on your monitor, and you will run into an UnsupportedOperationException. Comment out the line that throws this exception to explore the GUI. Now, red and yellow disks alternatingly drop into the columns. Red is one player; Yellow, the other. You see the basics of the game, with the dumb method playing against itself. But you will have to do some programming to keep the discs in the columns. Stop the game by clicking the red stop thingy at the top of the GUI (left for Mac, right for Windows).
3. Look at method GUI.main, and change the first two statements to assign Human objects to p1 and p2. Now, when you play the game, the GUI waits for each of the two human players to click on one of the words “Column 0”, …, “Column 6”. When you do, a disk will run down the column. Do this a few times.
4. Spend some time reading the classes. First read class Board, looking at —and understanding— its two static variables, enum Player (with its function opponent) and the board itself. Study the specs of all the methods in class Board —and look at their implementations too.

   Look at classes Move and State –don’t try to understand function toStringHelper at this point.
5. Write function Board.makeMove. To test it, use two Human players and play the game; you can see whether your method is right because, when you click on a column, a disk fills the lowest empty position in that column. See what happens when you click on a column that is already full; if you implement throwing an exception properly, a message should appear at the bottom of the GUI in this case –clearly, the caller caught the exception. Play the game with someone; see what happens when someone wins.

   Now let the computer play against itself using two Dummy Players –or you play against a Dummy.
6. Write function Board.getPossibleMoves. It is best to test it using method main in class GAME, so that it plays without a GUI. Read the comments in method Game.main. The comments contain information on how to test this method and others.

Part 2: Doing the AI stuff

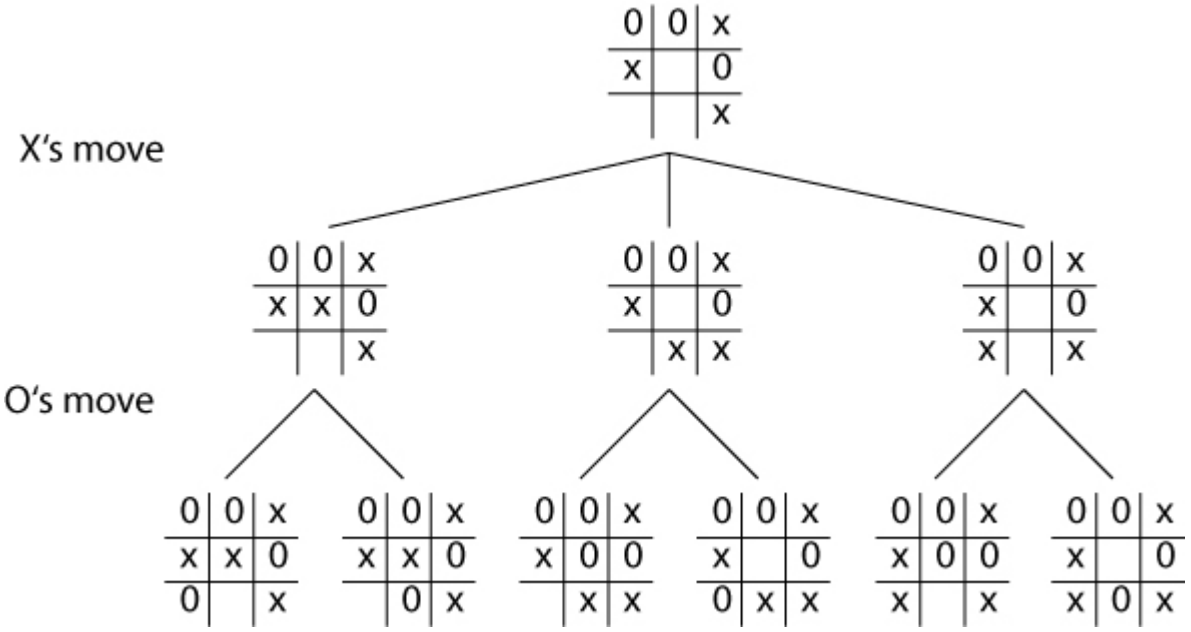
We now talk about implementing an AI (artificial intelligence technique and discuss implementing it in class AI.

Minimax Algorithm

Suppose your program wants to make a move in Connect 4. Before it makes the move, it will look

ahead at the possible sequences of moves it and the opponent could make and determine which first move is best, based on all those sequences of moves. The key data structure behind this is a tree in which the nodes are possible game states. Each state consists of the current board and which player's turn it is. From this state, the AI can generate all possible future states that can result after that player takes his turn.

We illustrate below with a tic tac toe game, Player X has three possible places to play, so there are three states children of the root state. From each of those states, the states are then generated for Player O's potential moves. This can continue until a state is reached that ends the game or results in a tie game. In the case of Connect Four, this happens when the board consists of one player having four discs connected.



However, it is computationally infeasible to generate all of these states because the state space grows exponentially with the number of turns your tree examines. Therefore, we have included a depth parameter, constraining the depth of the tree.

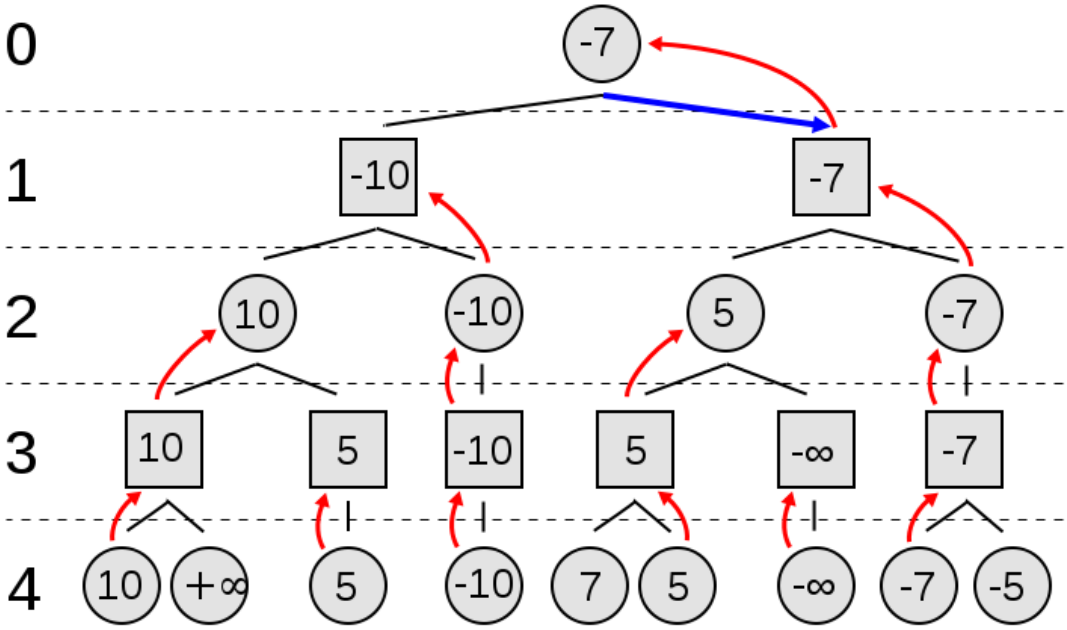
After the game state tree has been constructed, values need to be associated with each state in order to know which move is the best to make. The leaves of the tree are evaluated first: Each leaf calls an evaluation function to evaluate the state's board. The evaluation function gives a score to the state. The higher the score, the better the board is for you. If the state has four discs connected in your favor, the score is positive infinity (or the maximum integer value.) If it is a losing board, the score is negative infinity (or the minimum integer value.)

The last thing to do is evaluate the parent nodes in the tree. The game state tree creator (you) is trying to win the game. You want to maximize your score, while your enemy is trying to minimize your score. This knowledge will help determine the value for the parent states. If the current state is your color (i.e. it is your turn in that state), use the *maximum* value from the list of children as the current state's value. If the current state is not your color, use the *minimum* value from the list of children.

Now you see why it is called the Minimax algorithm.

For example in the illustration below, the circles represent your color and squares the opponent. The tree has already been constructed and the evaluation function was run at level 4. It is the opponent's turn at level 3, so the minimum value is always selected from the children. While at level 2, it is your turn, so

you always select the maximum value from level 2's children. Ultimately, the best possible known move bubbles up to the top where at level 0, the maximum value state is selected as the next move.



What to do for part 2

You are ready to start working with classes AI and State, using also Game or GUI to test your work.

Look at interface Solver. This is the primary means of allowing the game simulation to run separately from the players that make moves on the game while maintaining necessary communication between the two. This Solver abstraction is useful because it allows the game simulation not to care whether Humans, Dummys, AIs, or mixtures of them are playing the game.

Your ultimate goal with class AI is to implement method getMoves as specified in Solver so that, when class Game calls getMoves, it supplies your preferred Move. You may find it useful to look at class Game in order to see how the game actually runs by using Solvers.

Here are the methods you have to write. We suggest writing and testing them in this order:

- State.initializeChildren. Comments in Game.main explain how you can test it.
- AI.createGameTree. You can test it by creating a board, filing some columns, creating a new state, calling createGameTree, and printing the output to a file using s.writeToFile. Make sure you test it with several levels —0, 1, 2, 3 perhaps.
- AI.minimax. Testing it is much like the test for createGameTree except that you call minimax to fill in the values before printing.
- AI.getMoves.

One thing to keep in mind is the difference between generating a game tree and using that game tree to find the best move with algorithm minimax. Methods createGameTree and initializeChildren are related

only to generating a game tree. For this reason, don't call evaluateBoard in either of these functions.

If you wish to use any sorting method from the Java API to sort State objects by value, you must implement method State.compareTo. We did not use it in our sample solutions.

More on Testing

Here are a few ways to test your project:

1. GUI Interface
   a. As soon as you finish the methods in class Board, you should be able to run GUI.java and play against a Dummy AI. This Dummy AI will place discs in random columns. After you have built your AI, you can play against it or watch it play against itself!
2. Console Interface
   a. When you start developing, you will want to see how your game plays against itself. Run Game.java to see your AI play. Change the constant "SLEEP_INTERVAL" to lengthen the delay between plays, if you'd like.
3. Visualizing the Game Tree
   a. To print the game tree, use method State. writeToFile(). This method prints tree to file "output.txt" so that you can easily view the different values associated with each state.

What not to change

Write code only in classes Board, State, and AI –and Game for testing purposes. All the functions you must implement are marked with a comment that says TODO. Note the specifications for these (and all) methods because your grade will be based partially on how well your code implements the specification.

Submit your files on the CMS by the due date; the CMS tells you which classes to submit.