CSCI–4150 Introduction to Artificial Intelligence, Fall 2004
Assignment 4 (160 points), out Thursday September 30, various due dates

Introduction

This assignment is on game playing, mostly centered around implementing the alpha-beta minimax search and creating an evaluation function for the game of Connect 4, but there are also several written problems.

The web-tester for Connect 4 will allow you to upload an evaluation function and play games against other students' (and the teaching staff's) evaluation functions. We will run an (offline) round-robin tournament with your final evaluation functions; part of your score for Problem 4 will be based on how your entry performs in this tournament.

Notes on this assignment

- There are three deadlines for this assignment:
  - Thursday October 7:
    - Problem 1: written alpha-beta pruning problem
  - Thursday October 14:
    - Problem 2: alpha-beta MINIMAX proof
    - Problem 3: alpha-beta MINIMAX implementation
    - Problem 4-a: a working evaluation function for Connect 4
  - Thursday October 28:
    - Problem 4-b: final evaluation function
    - Problem 4-c: writeup of your evaluation function
    - Problem 5: EXPECTIMAX sampling
- There will be two support code files for this assignment:
  - `connect4.scm` — support source code for Connect 4
  - `a4code.com` — compiled support code for Connect 4
- There will be two web-testers for this assignment:
  - The web-tester for Problem 3 will be a regular web tester, as for previous assignments, with the standard electronic submission policy.
  - The web-tester for Problem 4 will be set up to run Connect 4 matches between students. You may upload a new evaluation function as many times as you want to this web-tester. For problem 4-a, you must submit a minimally competent evaluation function by the deadline; late submissions will be subject to the regular late policy. At the deadline for problem 4-b, we will take your most recently uploaded evaluation function as your final evaluation function. This is a hard deadline, i.e., no late submission!

    There will be a CPU time limit for each player in these games; see the Assignment 4 information web page for details.
- It is not appropriate for a student to give his or her evaluation function to another student for purposes of playing a game between their evaluation functions. You will be able to play games against other students' evaluation functions through the web tester.

A. Alpha-beta pruning

Alpha-beta pruning allows a MINIMAX search to prune certain branches of the game tree, resulting in faster evaluation of the tree.

> **Problem 1** (24 points) A separate handout/worksheet will have a written alpha-beta pruning problem. You will indicate which nodes are evaluated, give the value of the game tree, and indicate an "optimal" path from the root node to a leaf node of the tree.

> **Problem 2** (18 points) Do exercise 6.5 from the text. [proof of correctness for alpha-beta pruning]

B. Connect 4

The Connect 4 game is played on a "board" 6 rows high and 7 columns wide. Players alternately drop a piece from the top of any column, and it will fall to the lowest open row. At most 6 pieces can be in any column. Traditionally, one player is red and the other black, but for purposes of a text representation, we will use the letters "X" and "O" instead. Player "X" will always play first. The object is to get four pieces in a row, either horizontally, vertically, or diagonally.

The support code contains procedures that implement the Connect 4 game — printing, accessing, and manipulating boards, getting the children of a board state, running a game, and so on. Your tasks will be to implement the alpha-beta pruning algorithm and to write an evaluation function for the Connect 4 game.

You will need to read through much of the `connect4.scm` file. There is detailed documentation on much of the support code, some of which is repeated here. There is a section in this file on how to get started on this problem. One of the first things it suggests is to try playing against a random player:

```
(load "a4code")
(load "connect4")
(play-c4 random-player human-player)
```

B.1 Player procedures

In order to separate the evaluation function from the alpha-beta MINIMAX search and to independently control the search depth, you will write a procedure that returns a "player procedure." A player procedure is called with a board state and information about which player is up; it must run alpha-beta MINIMAX with the given evaluation function and depth cutoff and then returns a move.

> **Problem 3** (30 points) Write the procedure:
>
> ```
> (create-c4-player eval-fn depth-cutoff)
> ```
>
> where:
>
> - `(eval-fn board current-player max-player)` returns a number corresponding to the quality of the game state for MAX. The argument `board` is a Connect 4 board state, and `current-player` and `max-player` are both symbols (either `X` or `O`) to indicate which pieces a player is playing.
>
>   Although this may seem redundant, it avoids confusion and may actually be necessary for more sophisticated evaluation functions. To be concrete, if MAX is playing `'X`, evaluating a leaf node is done by the call: `(eval-fn state 'X 'X)` When MIN plays `'O'` and evaluates a leaf node, the call is: `(eval-fn state 'O 'X)`.
> - `depth-cutoff` is the depth at which the game tree will be cut off; leaf nodes of the tree will be at this depth (unless the game ends naturally at a shallower depth)

This problem is where you will implement alpha-beta MINIMAX search that will be embedded within a "player procedure." A player procedure must be of the form:

```
(player-fn board player-symbol)
```

where `board` is the current Connect 4 board state and `player` is the piece to be played, either (the symbol) `X` or `O`. This procedure must return a valid move, i.e. an integer between 1 and 7 inclusive.

Important note: your alpha-beta MINIMAX implementation must evaluate the children in the order given by the `c4-children` procedure, otherwise your code will not test properly. Also, your `create-c4-player` procedure must be able to work with any evaluation function, so you cannot hardcode any numerical values!

After you have written this procedure and your evaluation function, you can make the following call in order to play against it:

```
(play-c4 human-player (create-c4-player c4-eval 4))
```

The player procedure here uses the `c4-eval` function in its alpha-beta MINIMAX search up to a depth of 4.

B.1.1 Support code

There are a number of useful procedures in the `a4code.com` file:

- There is an example `create-c4-player` procedure given in the `assign4.scm` file for this problem. This procedure simply serves as a "front end" for the alpha-beta MINIMAX search that you will implement. See the file for details.
- `(other-player player-symbol)` — given an `'X` or an `'O`, it returns the other symbol
- There are a number of comparison procedures that work with numbers extended to include the symbols `pos-infinity` and `neg-infinity`:

```
(inf:max a b)     (inf:> a b)     (inf:>= a b)     (inf:= a b)
(inf:min a b)     (inf:< a b)     (inf:<= a b)
```

For example:

```
(inf:max 7 'neg-infinity)   ==> 7
(inf:< pos-infinity 200)    ==> #f
```

B.2 Evaluation functions

An evaluation function returns a number that indicates how good the board state is for MAX.

Problem 4 (72 points) Create an original evaluation function for Connect 4 and implement it in the Scheme procedure:

```
(c4-eval board current-player max-player)
```

The arguments and return value of this procedure are described in Problem 3 under the `eval-fn` argument. Part of your score on this problem will be based on the performance of your evaluation function.

a) Upload a working, minimally competent, evaluation function to the Connect 4 web-tester. "Minimally competent" means that your evaluation must beat a simple "reference player."

b) Play games against other evaluation functions and improve your evaluation function before the (hard) deadline for this part of the problem. We are somewhat likely to look at the code for your evaluation function, so please indent properly and comment it.

c) Writeup of your evaluation function. Final details and guidelines for the writeup will be specified later on the Assignment 4 information web page. I will probably ask you to describe (in prose) your evaluation function, run it against some "reference players," and do some analysis on the resulting games (citing specific examples of situations where your evaluation function casued the computer to make good or bad moves).

C. EXPECTIMAX

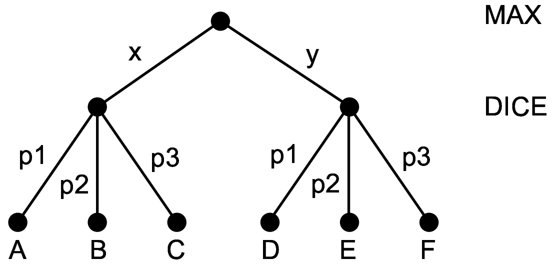
The EXPECTIMAX algorithm is an adaptation of the MINIMAX algorithm for games that involve an element of chance. The following problem is based on Exercise 6.5 from our text.

Problem 5 (16 points) Consider the following approach to analyzing games with an element of chance. Suppose the element of chance is the outcome of rolling two dice.

- Generate a number of dice roll sequences, where each sequence consists of many rolls of the dice, i.e., enough for the number of moves that you want to explore in the search
- For each dice roll sequence, you can use regular MINIMAX (with or without alpha-beta pruning) to analyze the game and determine what move MAX should make.
- Let each dice roll sequence vote on which move MAX should make, and choose the move with the most number of votes.

This approach essentially samples paths through the full game tree. Although this approach tends to work well in practice, it is not correct.

Consider the following game tree where MAX has a choice between two moves, x and y, and then there is a dice roll which has three possible outcomes (with probabilities p_1, p_2, and p_3), and then the state of the game is evaluated which results in the respective values of A, B, C, D, E, and F.



Answer the following questions:

a) Write a mathematical expression for the value of the MAX node under regular EXPECTIMAX.

b) Write a mathematical expression for the value of the MAX node using the above approach. Assume that the number of dice roll sequences (well, each sequence here consists of only one roll), is such that the number of rolls for each outcome is in exact proportion to the probability.

More specifically, suppose there are n_i rolls that produce outcome i. Then we are assuming that $\frac{n_i}{\sum_i n_i} = p_i$.

c) Find values for A, B, C, D, E, F, p_1, p_2, and p_3 for which the above algorithm makes a different decision than the regular EXPECTIMAX algorithm. Show the computations that demonstrate that your numbers produce the above behavior.