Assessed strategy-game exercise

November 25, 2015

1 Domineering

In the lectures you have learned how to play tic-tac-toe using game trees. We used the minimax algorithm to play optimally, and alpha-beta pruning to make this significantly more efficient. In this exercise, you will implement another strategy game, called *domineering*, using the same techniques, but going a little bit beyond.

1. Two players take turns placing dominoes on a board.
   (So that the dominoes don't overlap, of course.)
2. The first player, H, places them horizontally.
3. The other player, V, places them vertically.
4. The first player unable to move loses.

We will mark your solution for both correctness and efficiency:

1. Concentrate on correctness first.
2. Only once you achieved this, move on to consider efficiency for higher marks.

2 Important difference with tic-tac-toe

There is no draw. So the possible outcomes are -1 and 1. However, as we discuss below, other numbers can be used for euristic purposes, when the trees are too big to be fully computed and analysed in a feasible amount of time and space.

3 Organization of the exercise

There are two parts:

1. Deliberately open-ended, reported here.
2. More constrained, to be reported in a subsequent handout. In this second part, you will use the first part to produce required deliverables that will be marked.

4 Part 1

1. You will need to play on an $m \times n$ board for "small" given m, n.
2. You will need to play optimally, either as a first player or as a second player. You can define two functions for that, like we did in tic-tac-toe.
3. It is alright to ignore efficiency to begin with. However, alpha-beta pruning is just as easy as for tic-tac-toe, and makes a big difference.
4. But eventually you will need to consider efficiency, both regarding:
   (a) *The representation of the board.* The new thing you may wish to consider is special data structures, such as hash tables or binary search trees, or whatever you find to make a significant speed-up.
   We will have a lecture on "off-the-shelf data structures in Haskell".
   (b) *The search of the tree.* Perhaps the tree is too big to be fully (computed and) searched. So maybe you will need some euristics to stop the search prematurely.
   We will have a lecture on "stopping a tree search prematurely based on euristics".

Guidelines.

1. You will need to decide how you are going to represent the board.
2. Many things are going to be very similar, if not literally the same, to our tic-tac-toe program.
3. An obvious difference, other than the representation of the board, is the "play" function. What else?
4. You will need to print boards, in whatever format you choose, to experiment with your program, like we did with tic-tac-toe. At this stage, choose whatever format you find convenient for your own experiments.
5. Once you are satisfied with correctness, and you wish to improve your program so that bigger boards can be played in feasible time and space, you can consider the following euristics, which sacrifices optimality:

   > At the point we abandon a search, by refusing to make a too deep recursive call before we reach a leaf, we can approximate the outcome of a game tree by the difference of the number of available horizontal and vertical plays.

   I will address this kind of consideration in a lecture.

5 Part 2

To be given in a separate handout, a few days after you have started with Part 1.