CITS3001 Algorithms, Agents and Artificial Intelligence

Labsheet 6: game-playing

This lab is worth 4% of your final mark. It is an individual effort.

It should be submitted through cssubmit, by 5pm, Friday September 27

Use Minimax (and alpha-beta pruning, as appropriate) to construct a computer player for the game Mancala: http://en.wikipedia.org/wiki/Kalah.

The game is played on a ring board with fourteen *houses*, as shown in Fig. 1. Play starts with 3 *seeds* in each of the twelve small houses (or the game can be played with 4, 5, or 6 seeds each instead). Each player controls the six small houses on their side of the board, plus the *store* on their right-hand side.

The players alternate turns: in each turn Player *P sows* seeds as follows.

1. *P* picks up all of the seeds from one of their own small houses *h*.
2. *P* sows these seeds one per house counting anti-clockwise from *h*, but excluding the opponent's store if they get that far.
3. If the last seed sown is in their own store, *P* gets to sow again.
4. If the last seed sown is in one of their own empty houses *x*, and the house *y* opposite *x* is not empty, *P* moves the seeds from *x* and *y* into their store.

Fig. 1 shows an example turn with two "sowings" and a capture.

The game ends when either player has no seeds in any of their own small houses: their opponent then moves all remaining seeds to their store. The winner is the player with more seeds in their store.

You will need to think about an evaluation function for Mancala – counting stored seeds is obvious and easy, but what about positional issues too?

The game mechanics of Mancala are pretty simple. Implement them yourself if you want, or on the CITS3001 Resources page there is a Python program that plays an entire game of Mancala, choosing random moves for both players and displaying the evolving board in an SVG file that you can view in a web browser.

Implement the interface `MancalaAgent` provided at
http://teaching.csse.uwa.edu.au/units/CITS3001/lectures/labsheets/MancalaAgent.java

You will also find a class `Mancala`, that will run a game between two provided agents, and a class `RandomAgent` in the same directory.
http://teaching.csse.uwa.edu.au/units/CITS3001/lectures/labsheets/Mancala.java

http://teaching.csse.uwa.edu.au/units/CITS3001/lectures/labsheets/RandomAgent.java

Complete an agent and call it `MancalaImp.java` and submit your code to cssubmit. Your code will be checked, compiled and then your agent will compete in a randomised Mancala tournament. You will get 3/4 if your agent can consistently beat a random agent, and 4/4 if it sometimes beats a model solution.

Fig. 1. An example turn in Mancala. (Figure reproduced from Wikipedia.)