Project 3
CS4013/5013: Artificial Intelligence
Due 11:59PM March 1, 2011

Introduction

Your task for this assignment is to create an intelligent Mancala player using minimax search with alpha-beta pruning. Mancala is an ancient game, first thought to be developed in Africa. There are a number of variants of the game available. For this assignment, the rules are as follows.

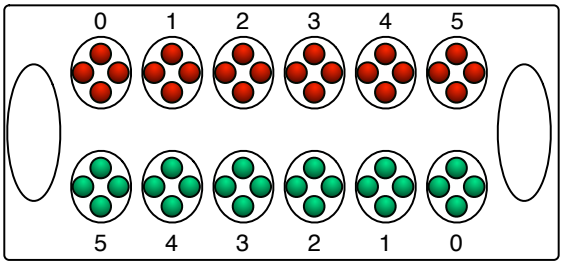

Figure 1: Mancala board at the start of the game

Mancala is played on a board with 12 regular pits and two special pits. There are two players, each of whom starts out with 24 stones evenly distributed over the 6 pits on their side of the board (4 per pit). This is shown in Figure 1. Each player has a special pit on the right side of the board (from their point of view) called the mancala. A player can choose to pick up all of the stones from any of their own six pits. The player then distributes these stones one at a time moving counter-clockwise around the board. A stone is dropped into the player's own mancala when passing it but stones cannot be dropped into the opponent's mancala. The game ends when one player has no stones. The stones remaining on the opponent's side are deposited into the opponent's mancala. The winner is the player with the most stones. It is not always an advantage to run out of stones first!

Two special rules apply to moving stones. The first is that if a player places the last stone in the mancala, that player receives an extra turn. For example, the first player can always get an extra turn by choosing pit 3 as her first move. I leave it to you to decide if this is a good strategy! Figure 2 shows this graphically.

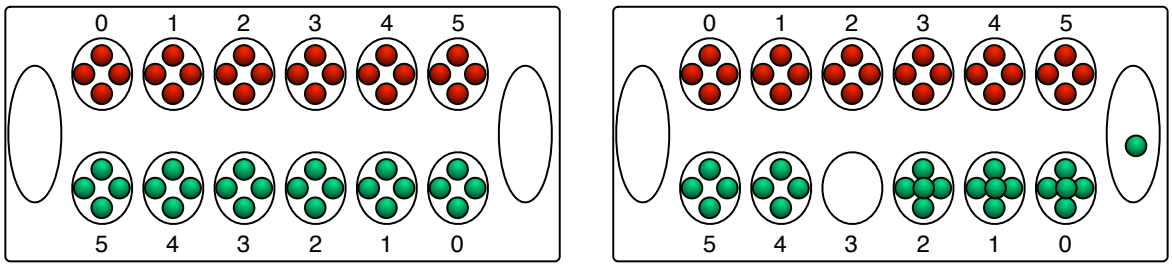

Figure 2: Illustration of the ability to get an extra turn. Note that the bottom player chose pit 3 and thus will receive an extra turn as the final stone was deposited into that player's mancala.

The second special rule is that if a player places the final stone in an empty pit on that player's side, that player steals all the opponent's stones from the opposite pit. Figure 3 shows an example of this rule.

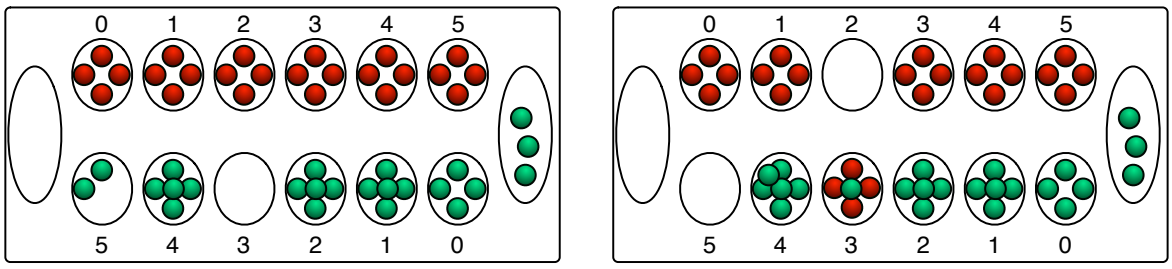

Figure 3: Illustration of stealing stones. The bottom player chose pit 5 with two stones in it. Her final stone is deposited into her pit 3, which is empty and thus she gets to steal the opponent's stones from pit 2.

Extra-credit opportunities

To keep grades within a fair range, all extra credit will be capped at 10 points. This means that no project can receive a grade higher than 110, no matter if they win the ladder,

find great exploits, and are very creative.

Wanted dead or alive: bugs or exploits in the simulator

We are reasonably certain that you cannot exploit the simulator (say by directly affecting your points scored). However, any project has bugs and we want to know about them! If you find a bug or an exploit, you can receive extra credit according to the following scale:

- **5 points:** If you find an exploit and report it to us you can receive 5 points extra credit upon verification of the report. Note, we know of two exploits for which you cannot get extra credit as they are already discovered. Since both are both extremely difficult to fix and extremely difficult to implement, I'm not listing them here.
- **10 points:** If you find an exploit and give us a fix for it, you can receive 10 points extra credit upon verification of both the exploit and the fix.
- **1-3 points:** General simulator bugs are much more likely than exploits. Finding a bug and reporting it can get you 1 point. Fixing the bug and giving us the fix (you can't check it in directly but you can give it to us in the bug report) can get you three points. Both bug and fix must be verified for any extra credit to be awarded.

Competition ladder

This project has a class-wide competition ladder. Because the project is short, the ladder starts Feb 22 (the day after Project 2) and will run each night until the project is due. The first place player will receive one extra point for each night that that player wins the ladder up to a maximum of 5 points. The second place player will receive 1/2 extra point for each night that the player is in second place. No player can receive more than 5 extra points from the ladder and points will be distributed down the ladder accordingly should the maximum be reached.

The ladder will be ranked by your average score in the games. Unlike the previous projects, you will be playing round-robin fashion against everyone else who has submitted to the ladder. To receive extra credit, you must rank above the Random agent.

Wanted (alive please): creative individuals

Creativity is highly encouraged! To make this real, there are up to 10 points of extra-credit available for creative solutions. If you choose to implement anything that you consider creative, please do the following:

- Document it in your writeup! I can't give extra credit unless I know you did something extra.
- Your search must still have minimax at the heart of it. If you are unsure if your search qualifies, come talk to me.
- Remember that by being creative I am referring to the algorithm and *not* to the ability to creatively download code. All project code must be written exclusively by you except for the sample players that we provide.

Implementation details - these have changed!

All of your source code must reside in your src/4x4 directory and be in your 4x4 package. You may name your files within this package anything that makes sense to you (remember that we are grading on coding style as well). To make the simulation know about your agent, you also need a `mancalainit.xml` file. There is a sample file in the examples.hello-mancala subdirectory. You will need to change this file to point to your new agent at name1234.MyAgent (or whatever you call your agent class). **Do not name it something other than mancalainit.xml.** Also, you will need to update the entry for displayName to be something other than the default. Include your name in it for ease of grading.

The agent class contains a startAction(), endAction(), and initialize() method by default. startAction() is called each time an agent is about to begin an action and it must return a valid action for the agent to execute. endAction() is called after all agents have ended their actions but before the simulator goes to the next timestep. This may be left empty if you have no need for cleaning up after an action. initialize() is called when an agent is created (but not when it comes back to life from being killed). The example heuristic agent shows you how to access the internal state of the agent and of the environment.

Using methods from the Java SDK is acceptable and encouraged but downloading or using code from any other sources is not allowed. See the syllabus for more details on what is considered academic misconduct. As discussed below, any additional files you create should be turned in along with your main agent class.

Competing heuristics

For this project, you may play against the following heuristics:

- **Random** makes completely random moves. Not very bright but you must beat it for extra-credit on the ladder!

- **GreedyClient** is our heuristic greedy player. It does the following. 1) If it can get an extra turn, it does. 2) If it can steal a stone, it does. 3) If it can get points, it does. 4) Last, play randomly. It isn't the best player but it is a lot better than Random!

Project 3 details

For this assignment, you need to do the following:

1. Update your aiprojects code from project 2. If you got subclipse working, you can update your code from within eclipse using Team → Update. If not, use the command line or tortoiseSvn to do "svn update". If you did not get the code checked out for project 0, follow the instructions to check out the code in that writeup. Note that because mancala was not in your original code release, you MUST update to even start this project! Also be sure to refresh eclipse (right click on the project and select refresh). This will update the listing of files.

2. Change the worldconfig.xml file in examples.hello-mancala to point to your agent in src/4x4. The detailed instructions for this are similar to the previous projects. Make sure to copy over a mancalainit.xml in the src/4x4 directory so your agent knows how to start. Be sure to update the displayName entry to something other than the default! You must include your name it in somehow for ease of grading but you can make it funny/witty beyond that. To run your agent, use ant to run the target hello-mancala.

3. Create a mancala player that makes moves using minimax search with $\alpha - \beta$ pruning. Because the search tree is large, you must also implement an evaluation function. Ensure that your player does not take more than 3 minutes total to search in a game. Build and test your code using the ant compilation system within eclipse or using ant on the command line if you are not using eclipse (we highly recommend eclipse or another IDE!).

    - Note: This project is the same for CS 4013/5013 students.

4. Test your player(s) on a sample ladder. This will ensure that the agent runs on the CSN linux machines and that you have the agent correctly configured. To do this, we have created a submission script that will take your agent as input and run it against the heuristic agents several times and output the information to your terminal window. WARNING: It will output a LOT of text or either be ready to scroll or put the output to a file. To submit to this agent, do the following:

```
/opt/ai4013/bin/submit CS4013 Project3TestLadder *.java mancalainit.xml
```

where you can simply list the java files instead of using *.java if you choose. Note that the xml file MUST be named mancalainit.xml!

5. Submit your project on codd.cs.ou.edu using the submit script as described below.

   (a) Log into codd.cs.ou.edu using the account that was created for you for this class. Your username is your 4x4 and your default password is cs#4x4. Remember to change your password! Use ssh to login to codd.

   (b) Make sure your working directory on codd.cs.ou.edu contains all the files you want to turn in. All files should live in your 4x4 package. The mancalainit.xml file is required to run your client! Also be sure to turn in your board template.

   ```
   /opt/ai4013/bin/submit CS4013 Project3 *.java mancalainit.xml
   ```

   (c) After the project deadline, the above command will not accept submissions. If you want to turn in your project late, use:

   ```
   /opt/ai4013/bin/submit CS4013 Project3Late *.java mancalainit.xml
   ```

Point distribution

- 40 points for correctly implementing minimax (even if you mess up pruning, that is described below). A correct player will handle the potential extra turns in the game and will play following the moves generated by minimax.
  - 35 points if there is only one minor mistake. Off by one errors count as minor mistakes.
  - 30 points if there are several minor mistakes.
  - 25 points if you have one major mistake. An example of a major mistake would be failing to account for the extra turns (if you do this, your whole min/max alternation will be wrong and you will play poorly. Hence it is a major error.).
  - 15 if there are several major mistakes.
  - 10 points for an agent that at least does something other than random movements.
- 20 points for designing and correctly implementing alpha-beta pruning.

  - 15 points for one minor mistake. An example would be incorrect bookkeeping on alpha or beta's values such that some additional nodes are searched.
  - 10 points for several minor mistakes or one major mistake. An example of a major mistake would be incorrectly pruning.

- 20 points for designing and correctly implementing an evaluation function that orders states correctly and is useful to your minimax agent.

  - 15 points for one minor mistake. An example would be correctly designing an evaluation function that orders most states correctly but has a coding or other error that can cause your agent to incorrectly rank a board state.
  - 10 points for several minor mistakes or one major mistake. An example of a major mistake would be designing a heuristic that that takes too long to evaluate or incorrectly ranks states (e.g. says a losing board is a winning one).

- 10 points: We will randomly choose from one of the following good coding practices to grade for these 10 points. Note that this will be included on every project. Are your files well commented? Are your variable names descriptive (or are they all i, j, and k)? Do you make good use of classes and methods or is the entire project in one big flat file? This will be graded as follows:

  - 10 points for well commented code, descriptive variables names or making good use of classes and methods
  - 5 points if you have partially commented code, semi-descriptive variable names, or partial use of classes and methods
  - 0 points if you have no comments in your code, variables are obscurely named, or all your code is in a single flat method.

- 10 points for your writeup. A full-credit writeup will describe your evaluation function and why you chose that function.

- As with the previous project, we will deduct 5 points from your total score if your password has not changed from the default (cs#4x4) password.