CSE 150 - Project 2

Due 11:59PM, 6 November 2012

In this project you will develop agents to play checkers. You will create a simple minimax agent, an alpha-beta pruning minimax agent, and a custom agent of your own design that should outperform your minimax agents.

1 Logistics

You can work alone or in groups of at most 3. This project requires programming in Java, since we have created an interface to allow competition between different groups. If you haven't worked in Java, we encourage you to use the "project teams and study groups" post on Piazza to find teammates who are comfortable with Java.

We will not accept late submissions.

2 Provided code

We have provided code to deal with representing the checkers game and board states. Documentation of the provided code is given in `doc/index.html`.

Board states are represented by the `Checkers.Board` class. The `board` field of this class represents the board state as an 8x8 integer array, where

- 0 denotes an empty space
- 1 denotes a red checker
- 2 denotes a red king
- -1 denotes a black checker
- -2 denotes a black king

The `Checkers.Game` class contains code for determining successor states of a given board position. Successor states are found by the method `public static Vector<Board> expand(Board)`. This method also keeps track of the depth and number of nodes expanded during each round of search. It returns `null` when the depth and/or number of expanded nodes exceeds the allowed limits (which are specified in the `Game` constructor). You MUST use the provided interface; trying to get around this interface will result in heavy penalties!

`CheckersPlayer.Solver` is an interface that your checkers-playing agents will implement. It contains a single function, `int selectMove(Board)` which returns an index into the Vector of successor states returned by `public static Vector<Board> expand(Board)`. This index represents the action that your

agent chooses in that state. We have provided a simple agent in RandomPlayer/RandomSolver.java, which simply selects a move at random.

Also included is a folder called "TeamName," which you will rename to a group-specific name. In this folder are MinimaxSolver.java, AlphaBetaSolver.java and CustomSolver.java, which contain starter code for the agents you will implement. These files are all in the package "TeamName," which you must also rename to your group-specific name. **All additional files you create must be placed in this folder, and all additional classes you implement must be in the (renamed) "TeamName" package**.

Player.java contains code that plays a game between two agents. You will want to modify this to test your agents. **This is the only one of our provided files outside of the "TeamName" folder that you should modify, and you only need to modify it as indicated by the comments marked "TODO"**.

3 Requirements

- You must rename the TeamName folder to a name specific to your team. Either come up with a creative name, or use the names of the students in the group. **You must also change the package name to match the folder name. All other files you add must be in the (renamed) TeamName folder and package (i.e., additional source files should begin with "`package WhateverNameYouCameUpWith;`")**.

- Implement the `MinimaxSolver` in TeamName/MinimaxSolver.java. This must use simple minimax search with an evaluation function. A simple initial evaluation function would be to sum the values in the `Board.board` 8x8 integer array. This will return a higher value if red has more checkers than black for a given board position.

- Implement the `AlphaBetaSolver` in TeamName/AlphaBetaSolver.java. This must use minimax search with alpha-beta pruning.

- Implement the `CustomSolver` in TeamName/CustomSolver.java. This agent will be used to compete with other students.

  We are giving you some freedom in your custom solver, but it must be an improvement over a minimax alpha-beta agent with the board sum evaluation function. You will want to consider improved evaluation functions for your custom solver, and may want to look at other search algorithms, such as beam search.

4 Testing

You should modify Player.java to pit your solvers against each other. Make sure to test your agents both as player one (red) and player two (black).

Remember that the provided code will keep track of the search depth and number of nodes expanded and prevent searching beyond the limits. To test different limits, change the values passed into the `Game` constructor in Player.java.

5 Writeup

You must submit a writeup for this project in PDF format. You should include the following:

- The students in your group, their emails, and how each one contributed to the project
- What you named your custom solver and anything else we should know about running your code
- The approach you used in your custom solver, and other approaches you tried/considered
- Evaluate qualitatively how your agents play. Did you notice any situations where they make seemingly irrational decisions? What could you do/what did you do to improve the performance in these situations?
- Include a graph that shows the average execution time per move as a function of max depth for each of your three methods.
- What is the maximum depth your minimax agent can search for a reasonable amount of time (say one or two seconds per move). How much deeper can your alpha-beta agent search in the same amount of time?
- Let both of your minimax agents play the RandomSolver for 100 games. Repeat this test with several different depth limits. Report the number of wins, losses, and ties.
- Let your CustomSolver agent play against your alpha-beta agent, once as red and once as black. Repeat the test with several different depth limits/ limits on the number of expanded nodes (called "look limit" in the code). Analyze the results.
- Let your custom agent play itself, but with different depth/look limits. Repeat this test with several different combinations of limits. What values worked well for your agent?

Your writeup should be in NIPS format (`http://nips.cc/PaperInformation/StyleFiles`). We will grade based on the quality of the writeup, including structure and clarity of explanations.

6 Submission

Please zip your source files and PDF writeup together. You only need to submit the source files in your (renamed) "TeamName" directory. Name your zip file `student1_student2_p2.zip`, where student1 and student2 are student IDs or names of your group mates.

Email this file to baduncan@eng.ucsd.edu with the following subject line: CSE 150 P2 Submission. Do not copy the professor, and be sure the subject line is correct or I may not see your email. If you worked in a group of several students, please only send one submission per group.

7 Evaluation

Your project will be graded based on whether you implemented all the requirements, the quality of your report, and how well your Solver agents perform. Your minimax agents should beat the random solver most of the time, and your custom solver should beat your minimax agents most of the time. We will also give points for how well your custom agent performs against other students' custom agents.

For the class competition, the custom solver will be run in "look-limited" mode (limit on number of expanded nodes). We will not reveal the exact limit that will be used, but it will be at least 500 nodes.

Note that we will be using an automated tester, so it is important that you use the interface provided. Bypassing the interface in order to expand more nodes will be heavily penalized.