Assignment 2: Mancala

Acknowledgments: Much of this assignment is copied from an assignment written by Professor Grant Braught.

This assignment consists of both code and written components. The criteria for grading the written components are described separately for each component below.

Question 1. (30 points) A MiniMax Mancala Player

Using the starter code provided, implement a player for the Othello game that uses the MiniMax algorithm (with cutoff at a fixed depth) to select its move. You must create a new class called MMMancalaPlayer player, which extends the MancalaPlayer class and implements the MiniMax interface.

Comments:

* None of the given code may be modified.
* Your player must have 2 constructors:
  + public MMMancalaPlayer (String name)
  + public MMMancalaPlayer (String name, int depthLimit)
* Both constructors must start with the statement: super(name);
* The one-argument constructor may initialize the depth limit as you see fit.
* The search must not generate nodes beyond the depth limit.
* Your player must correctly report all of the values described in the MiniMax interface.
* Your static evaluation function must be completely contained in the staticEvaluator method specified by the MiniMax interface.
* You may ignore the deadline parameter to getMove at this point.
* Use the following simple static evaluation function: the number of pieces owned by MAX minus the number of players owned by MIN.

In writing your MancalaPlayer and later your static evaluation function you may find useful the methods of the GameState class that are listed below. You will need to examine the source code for specific information on how to invoke these methods and exactly what they return.

* getValidMoves – get a list of all valid moves for the player whose turn it is.
* isLegalMove – returns true if the specified move is legal in the current state.
* makeMove – returns the state reached by applying a specified move to this state.
* getTurn – get the player whose turn it is.
* getSquare – find out which player owns a given square.
* getPlayerScore – get the score for a specified player.
* getFinalNetScore – get the final score.
* isGameOver – get the status of the game

Question 2. (25 points) An alpha-beta Mancala player

Create a copy of your Mancala from question 1 in a new class, called ABMancalaPlayer, and modify it so that it uses alpha-beta pruning during search.

Question 3. (20 points) Comparing minimax and alpha-beta

(a) Design and conduct an experiment to determine the following values:

* the number of nodes per second that your MiniMax player can generate
* the number of nodes per second that your Alpha-Beta player can generate
* the approximate average branching factor of the search tree in an Mancala game
* the approximate effective average branching factor for your implementation of alpha-beta search

Provide a written description (1-2 pages) of the experiment that you performed containing sufficient detail that one of your classmates could repeat your work. At the top of the report, clearly state your estimates of the four values listed above. The grade for this question will be based on the elegance and suitability of the experimental design, as well as the clarity, correctness, spelling and grammar of the description.

(b) Answer the following questions using your data from part (a). Justify your answers.

1. Approximately how long would your MiniMax and Alpha-Beta players take to explore the game tree to a depth of 12 plies?
2. If your MiniMax player can explore to a depth of D in a given time T, approximately what depth would your Alpha-Beta player be able to explore in the same time T?

Question 4: (20 points) Mancala Steel Cage Death Match Of Terror

(a) (0 points, but you can earn extra late days as prizes) Create an MancalaPlayer, including an improved static evaluation function, to participate in our Mancala Steel Cage Death Match. Your player must be implemented in a class called XXXMancalaPlayer, where XXX is the name you have chosen for your entry into the Death Match. The only limitation is that your player must use some variant of the MiniMax algorithm (perhaps with Alpha/Beta pruning, perhaps with other optimizations that you develop). In class we will run an Mancala tournament in which your MancalaPlayer implementations will play against each other. The players will have a maximum of 2 second per move. The following line of code will tell you how many milliseconds remain before the time for the current move expires:

long msLeft = deadline – System.currentTimeMillis();

Your XXXMancalaPlayer must be self-contained, as it will be copied and pasted into the example framework in order to run the tournament. Therefore, if you need additional helper classes, include them as nested classes within XXXMancalaPlayer.

You are encouraged to research Mancala-playing strategies for this part of the assignment, and it is permissible to copy any *ideas* your research reveals, provided they are appropriately cited in part (b) below. However, you may not copy any code whatsoever, from any source. All code submitted (except for the framework provided, of course) must be your own work.

Prizes for the Mancala Steel Cage Death Match will be awarded as follows: first place gets 2 extra late days, second place gets 1 extra late day.

(b) (20 points) Write a description of the strategy you implemented (0.5-1.5 pages). Remember to cite any sources on which your ideas were based. The grade for this question will be based on the originality and appropriateness of the ideas, as well as the clarity, correctness, spelling and grammar of the description.

Question 5: (5 points) Self-assessment document

As with the previous assignment, please create a self-assessment document for all the code you wrote. The main purpose of this document is to encourage you to test and assess your own work; the document will also help with grading as described below. The document should be no more than one page in length. For each of questions 1, 2, and 4a in this assignment, write up to three sentences explaining whether or not you completed the question correctly. If you were not able to complete the question, briefly describe what problems you observed in your code, indicate roughly which parts are working correctly and which parts are not working correctly, and if possible give a reason for the problems you observed. If you were able to complete the question correctly, give a brief description of how you tested for correctness. The instructor will use the self-assessment document to help with grading. Incorrect self-assessments will lose a few extra points compared to correct self-assessments.

Submission instructions

Please submit exactly 1 zip file to Moodle. The zip file must contain

* the three relevant .java files (MMMancalaPlayer.java, ABMancalaPlayer.java, XXXMancalaPlayer.java), and
* a single file in any reasonable format (e.g. PDF, OpenOffice, Microsoft Word) containing the written answers to questions 3, 4(b), and 5.