Programming Project #1: Othello

CSE 473, Spring 2005

Due Monday, May 2

In this project you will implement and document an Othello-playing game agent. We will give you starter code so that you dont have to worry about the non-AI overhead. At the end of the assignment we will have an Othello tournament in which your agents play against each other, with a prize for the winning team.

The Game

This version of Othello is a tiny bit different from the Othello you already know. The four corner squares have been removed from the board. All of the rules, the initial configuration, and the conditions for winning remain the same. You can see the rules for Othello at:

`http://www.rainfall.com/othello/rules/othellorules.asp`

and if you look around on the web you can find sites that allow you to play the original version of the game (also called Reversi) against computers or other people.

Your Task

Your job is to implement an Othello-playing agent that plays as well as possible under various time constraints. Your program needs to be able to accept arguments specifying whether it is playing as black or as white, and also the amount of thinking time the program gets for the game.

You will need to write a function that takes a current board state and the amount of thinking time remaining, and makes a single move. This function will then return the state of the board after the move is made.

The way in which you implement this function is up to you. However, there are some things you need to do along the way:

1. **Implement a straightforward minimax search agent.** Choose an arbitrary (but legal) game configuration with 4 moves (2 for each player) remaining. How long does it take for minimax to do an exhaustive search of the remaining space? What if there are 8 moves remaining? Or 16?
2. **Implement alpha-beta pruning in your minimax search agent.** For the same near-endgame configurations you used in 1), how do the search times compare now that pruning is enabled?

Of course, during a real game with limited time it will not be possible for either your minimax agent or your alpha-beta agent to do an exhaustive search, so you will have to use some of the other techniques we discussed in class. Most notably, you will have to:

3. **Develop and implement a set of heuristics that allow you to assign/evaluate the utility of a midgame board.** What characteristics make a board configuration good? How can you formalize these characteristics into efficiently computable heuristics?
4. **Develop and implement a timing strategy.** Since we know that each player takes exactly 28 moves per game in our modified Othello, the simplest timing strategy is to divide the available time by 28 and use a fixed amount of time per move. Of course, there are some moves in a game that are more important than others and perhaps youd like to spend more thought time on those.

Once these four basics are completed, there is a world of opportunity for further improvement (and extra credit!). Some ideas include:

- Fancify your search depth choices. You can try adding a quiescence-based heuristic to decide how deeply to search each branch of the game tree, for example.
- Use different sets of heuristics for the beginning, middle and end of the game.
- Create a library of opening moves, or endgame positions.
- . . . etc.

What To Deliver

After the assignment is turned in we will have a tournament in which we play your Othello agents against each other. So, we will need a copy of your codeplease make sure you follow the naming conventions given in starter code so that everyones programs can be called in the same manner. Instructions on how to turn in your code will come out closer to the deadline.

In addition to your code, we will need a writeup explaining what it is that youve done. The writeup should include the following:

- A discussion of minimax as it pertains to our modified Othello. You can use the questions in the project description section as a starting point.
- A discussion of alpha beta pruning in the context of our modified Othello. Again, you can use the given questions as a springboard. How much advantage does pruning gain us in this context?
- A detailed description of the heuristics you develop for board evaluation. What characteristics did you consider important, and why? Is there anything that you considered using but in the end did not? Why?
- A description of your timing strategy. Why did you choose this strategy?
- A description of anything beyond-the-basics that you did.
- A discussion of what, if anything, you might do differently if you had to start over, or what you think would have been helpful to implement if youd had more time.

What We Provide

We provide Java code to represent the current state of an Othello game, as well as some GUI code to play the game itself. The javadoc info for these classes is located at:

`http://www.cs.washington.edu/473/homework/othello/docs/`

What you'll first need to do is implement the `Player` interface. The main thing needed in this class is to implement the `makeMove` method, which takes in an object of type `OthelloState`. The `OthelloState` object records the state of the game, with its most important methods being:

1. `at`: this will tell you what piece (if any) is at any position on the board.
2. `makeMove`: this makes the specified move.
3. `forfeitMove`: if there is no move to be made you must forfeit your move.

If an invalid move is made at any point during the game, the game will be ended, so you should make sure your computer-player makes no errors. Each "piece" is represented using an `OthelloPiece` object (either `Black` or `White`).

Once you're done making your player (or if you want to test it any point) you should flesh out your main function. To play the game all you need to do is make a new object of type `OthelloGame` with 0, 1, or 2 computer controlled players. Then, just call the `play` method on this object.

If you have any questions about the underlying classes be sure to email Matt.