

CS440: Introduction to Artificial Intelligence
Fall 2008
Programming Assignment #1: “Tri-State Othello”
Due Thursday, Oct. 2nd, at noon.

Introduction

Othello is a popular board game from Pressman Toys (a variant on the older game of *Go*). For this assignment, you will write a program based on opponent search that plays a variant of *Othello*.

Othello is a two-player game. Taking turns, one player places white stones on a grid while the other player places black stones on the grid. Whichever player has the most stones on the grid at the end of the game wins. The game ends when every grid square has a stone on it.

The key to *Othello* is to outflank your opponent. When a stone is placed on a square, it “outflanks” one or more opponent pieces if they form a contiguous straight line (vertical, horizontal or diagonal, with no empty spaces) between two stone of the same color. When this happens, the outflanked pieces change from the opponent’s color to the player’s color. Note that outflanking is not recursive: one of the flanking pieces must be the newly played piece. Also, every move must outflank at least one of your opponent’s pieces. If a player cannot play a stone that outflanks any of the opponent’s stones, then that player’s turn is skipped. It is not legal to skip a turn, however, if a legal move exists. For the complete rules to Othello (with illustrations), see the official rules from Pressman Toys. They are accessible through the External Links page on the class web site.

Tri-state Othello is just like Othello, with the following exceptions: (1) the initial board size varies, and the board may not be square (although it is always a rectangular grid); and (2) some of the grid squares are marked ‘E’ (for *empty*). These squares may not be filled, yet they count as empty squares for flanking purposes. Each game will begin with on a different size board with a different pattern of squares marked ‘E’, and continues until all the squares not marked ‘E’ are full. This is designed to interfere with traditional *Othello* strategies, and thereby remove any advantage that previous experience with *Othello* might bring.

The Challenge

Your assignment is to write a program based on opponent search that selects the next move in a game of *Tri-state Othello*. Your program will be given the name of an input file that depicts the current state of the board, and indicates whether you are playing white or black. Your program should write a grid coordinate (x,y) to `std::cout` (in C++) or `standard.out` (in Java), or the word “skip” if your player is unable to move.

You may write your program in either C++ or Java (any other language must be explicitly approved by the instructor). The input file is an ASCII file. The first line begins with one of two characters -- ‘B’ or ‘W’ – indicating whether your program is playing black or white. This is followed by white space, the width of the grid, more whitespace, the height of the grid, and a newline. The rest of the file contains the current state of the board. There are *height* additional

rows in the file, each of which contains *width* characters. There are four characters that may appear in the grid, as follows: 'B' means the grid cell is already filled with a black stone; 'W' means the grid cell is filled with a white stone; 'E' means the grid cell is empty but cannot be filled; "-" (dash) means the grid cell is empty but can be filled. (Note: just because a grid cell contains a "-" does not mean that it is a legal move: all legal moves must outflank at least one of your opponent's stones.) A sample input file is being distributed with this assignment on the web site; go to the Assignments page.

Finally, there is a time limit. Your program is not allowed to take more than 2 minutes per turn. Since we do not know how loaded the machines will be when your assignment is graded, we strongly suggest that your program note the time when it is called, and check periodically during its search process to make sure that it produces an answer before the allotted time is up.

Grading

A minimum acceptable standard for a program to get a passing grade (i.e. a 'C') is for the program to always produce a legal move, given a legal input file. We will test your program for this standard by giving it a series of input files (board positions) and checking the legality of the output. Note the following: (1) any output after 2 minutes is illegal; (2) if no move outflanks at least one opponent stone, then 'skip' is the only legal output; (3) if at least one move exists that outflanks an opponent's stone, then "skip" is not a legal move.

To get a good grade (i.e. above 'C'), your program should perform well. Jason (the teaching assistant) has prepared a program that plays at a 'B' level. If your program plays about as well as it does, it will earn approximately a 'B'; if your program plays significantly better than it does, it will earn an 'A'. We will test this experimentally (although generally beginning in the middle of games, to save time).

Finally, we will also run a tournament. This is for fun, and is optional (tournaments can be unfair, if you face a first-round opponent who is really good). The tournament is not a basis for grading. You should feel free to opt out by sending an e-mail to Jason (no one else will know).

Submission

Your assignment should contain a README file that clearly explains to Jason (1) how to compile your program and (2) how to run it. If he has trouble compiling and/or running your assignment, it's your fault, not his (so long as he has read and follows the README file). To submit your assignment, make a single tar file containing all your source code (no executables, please) and the README file, and submit it through the class on-line Check-in program.

Late Policy

Assignments are due when they are due; late assignments are not accepted. There is a departmental exception for "unforeseeable circumstances". Examples include medical emergencies, family tragedies, fires, etc. If such an unforeseeable circumstance happens to you, please talk to the instructor. We will work with you on a case-by-case basis.