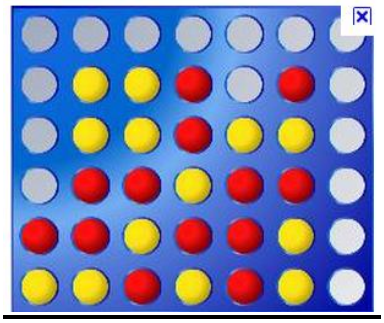
CSCI561 – Introduction to Artificial Intelligence
Instructor: Dr. K. Narayanaswamy

Assignment 3 - Connect N
Due: 10/17/2011 11:59:59pm
Electronic Submission (on Blackboard)

1. Problem Definition



The 'Connect N' game is played by two people (Player1 and Player2) using two different colored game pieces (Red and Yellow) respectively on a vertical m*m board with an aim of connecting n game piece together. The Player1 with red color game piece will always start the game; initially the board will be empty and then they play alternatively taking turns.

We consider the grid to be rows and columns, and the location of each game piece is uniquely specified by its coordinates given as a pair <x, y> where the piece is on row x and column y. The origin, i.e., coordinate <0,> is at the lower left corner of the grid . So, row numbers increase as you go upward from the origin, and column numbers increase as you go to the right from the origin.

When a player to takes his turn, he indicates the column in which the game piece is to be placed, if the column contains no game pieces, then the game piece is placed in the bottom most row and if the player selects a column containing other piece/s, it gets inserted in the largest row not containing a piece in the specified column. They take turns while playing this game.

Each player tries to connect N game pieces together in a row or column or diagonally.

You might first try to play the game:
http://www.2flashgames.com/f/f-581.htm

Input Format:-

Command line syntax for the game is given below

game {minimax|alpha-beta} [–m boardSize] [–n peiceToConnect] [–d depthOfSearch] [-c] outputFileName

where game is the name of the executable.

Either ‘minimax’ or ‘alpha-beta’ algorithm will be used to run your program, default is minimax.

Square bracketed items are optional. You must follow the UNIX convention that command line options can come in any order. (Note: a command line option is a command line argument that begins with a - character in a command line syntax specification.)

The first argument is the algorithm to run flag (‘min-max’ for Min-Max algorithm and ‘alpha-beta’ for Alpha-Beta Pruning Algorithm) and the last argument is the name of the output file which you are supposed to write with the game information as mentioned in the output section of the assignment.

If –m is specified, then ‘boardSize’ should be considered as size of the vertical square board, default value of the boardSize is ‘10’.

If –n is specified, then a player should try to connect ‘pieceToConnect’ number of connections to be made in order for a player to win, default value of the ‘pieceToConnect’ is ‘5’.

If –d is specified, then ‘depthOfSearch’ would be the depth of search on min-max or alpha-beta tree, default value of the ‘depthOfSearch’ is ‘5’.

If –c is specified, then it is the computer v/s computer game with no human intervention else it is human v/s computer (please let the computer be the first player to start the game).

Interface for Human to input the column for his/her move would be (Only of Computer v/s Human Game):

Please select the column for Move ‘X’: Y

where ‘X’ is the number of the move he/she is going to make and ‘Y’ is the selected column number and ‘Y’ can take values from 0 <= Y < m (0 is the left most column)

Output Format:-

Name of the winner should be displayed on the console as specified below,
Congratulations Player {1|2}, you have won the game.

Followed by the entire board with Red Pieced Marked with ‘R’ (character) and Yellow with ‘Y’ with a line between rows and a tab between column entries, an example grid is shown below.

R Y R Y

Y R R Y

R Y R Y

There will be an output file common to both the players, specifying the player move number along with co-ordinates of the game piece placed as a result of the move.

Total Number of Moves: X
Player 1 Move 1: (0, y1) – Number of nodes examined: 'a'
Player 2 Move 1: (X1, Y1) – Number of nodes examined: 'b'
Player 1 Move 2: (x2, y2) – Number of nodes examined: 'c'
Player 2 Move 2: (X2, Y2) – Number of nodes examined: 'c'
.
.
.
Player X Move'N': (xN,yN) – Number of Nodes examined: 'z'

Here x is the row number and y is the column number and Number of nodes examined is the number of nodes evaluated to make a move.

Please consider bottom most row as the 0^{th} row and left most column as the 0^{th} column for this assignment.

2. Algorithm Requirement

Design a '**rational**' agent that plays N Connect well. You must implement both '**Minimax**' and '**Alpha-Beta**' Pruning Algorithms to achieve this.

Algorithm Specific Details: -

Utility function: $+\infty$ if player has connected n game pieces, 0 if board is full, $-\infty$ if opponent has connected n game pieces. We will be using the utility function at depth "d" to make the computations manageable since full descent minimax/alpha–beta pruning will be infeasible. So, you will need to devise an appropriate utility function that can estimate the value of nodes without fully descending to the leaf nodes of the tree like the standard algorithms.

In 'Connect N' game, a player needs a winning line (either row or column or diagonally) of n game pieces to win a game.

We can use an evaluation function such as 'n-1'-out-'n' winning pieces in a line for this assignment.

We are describing the same in the context of a connect 4 (connect 4 game pieces in a line).

- Count total 3-out-of-4 "unblocked" winning lines for a player
- Compare the same to that of the opponent
- Now Utility of a move would be equal to = f(p,G) – f(opponent(p),G)

where f(a,G) = number of 3-out-of-4 winning lines for player a on board G

You can now extend this heuristic to 'n-out-of-N' Heuristic where N is the number of pieces to connect and n is the number of pieces in a line (either in a row or column or diagonally). Again we are describing the same in the context of connect 4,
For example:

- To extend '3-out-of-4' heuristic to '*n*-out-of-4' where $n \leq 3$

- We need to award weighted points based on the value of *n (number of game pieces in a line)*
- Score(p,G) = $100(n_3) + 10(n_2) + 1(n_1)$
  n_i is the number of *i*-out-of-4 winning lines for player p on game board G, here a weight of 100,10 and 1 is assigned to 3,2,1 game piece/s in line respectively.

You must follow the evaluation function described above to compute the utility of the move.

You need to implement two variants of the game:

1) **Computer(Player 1) v/s Computer(Player 2)**
2) **Computer v/s Human** (Computer will act as Player 1 and will make the first move)

*If –c is specified, then it will be a Computer v/s Computer game with no human intervention, else it will be a Human v/s Computer (please let the computer be the first player to start the game).

Please assume that the 'Player 1' will always start the game.

Please refer to the class notes or recommended course book for the details of these algorithms.

You must design a rational agent as part of this assignment, though as a human you can add some irrationality to the moves while taking your turn but a computer must make a rational move.

Please note that we will running a depth limited version of both the algorithms, we will be specifying the depth of the search through the command line option to your program.

You might first try your hands on the game first: -
http://www.2flashgames.com/f/f-581.htm

3. Coding Requirement

Your program must be written in C++ or Java.

You may use the Standard Libraries (e.g. STL for C++ or Java's standard container libraries), you may not use any 3rd party objects/algorithms without prior permission from the course TA or Professor (don't go download code that runs search algorithms).

No collaboration will be allowed on this project. These must reflect just your own work, with no outside help (except for questions asked of the instructor and TA). This explicitly includes no use of any code from elsewhere (including the Internet) without explicit permission from the course Instructor or TA.

Your program must compile and run on Aludra (see: http://www.usc.edu/its/web/getting started/ppages.html if you are unfamiliar with Aludra).

4. Submission Guidelines

a) General Coding Guidelines

For the C++ programmers:
Make sure that your source code can be compiled and executed on aludra.usc.edu. Please provide a README file with instructions to run your programs and a Makefile to compile where possible.

For the JAVA programmers:
All JAVA classes should be in package CS561A3.<LastName> and the source files should be in src/CS561A3/<LastName>. Please provide a README file with instructions to run your programs.

For Submission:
Place all the required files in a single directoryZip all the files in your submission into a single file called: - Firstname_LastName_A3.zip

Only Zip format will be accepted.

Please do not use any other processes or tools.

b) Execution Details

game {min-max|alpha-beta} [–m boardSize] [–n peiceToConnect] [–d depthOfSearch] [-c] outputFile

or

java –cp CLASSPATH CS561A3.<LastName>. *game {min-max|alpha-beta} [–m boardSize] [–n peiceToConnect] [–d depthOfSearch] [-c] outputFile*

where 'outputFile' is the text file which will enclose the solution as described above.

c) README.txt

This file must contain the following information
– Your name (First and Last)
– Your USC Student number
– Your USC email address
– Implementation Details (**Steps of Algorithm**)
– Any additional information you want us to know when grading your work.

5. Grading Guidelines

• Incorrect File names and extensions (case sensitive): - [- 5 points]

• Your implementation will be checked with 5 different test cases: -

Correct listing of **all the moves** for each player and correct listing of the winner and resulting grid: - 20 * 5 [100 points]

• Missing README file: - [-10 points]
(even if lots of comments in code)

• Your code must be robust. If your program crashes for any reason or fails to terminate in reasonable time on any test case, then it fails that test. If your program terminates gracefully, more partial credit might be awarded.

Please make sure ALL source files are submitted in the zip file you submit. Errors in submission will be assessed –25 points.

A program that does not compile as submitted will be given 0 points. None of your other submissions will be graded if the program does not compile.

Only your FINAL submission will be graded.

For policies on late submissions, please see the Syllabus from the course home page. These policies will be enforced with no exceptions.