CISC280 Spring 2009 Project, part A, 9 of 15 percent

Testing Code due Friday Midnight April 24

20 pts

Working tested code due Friday Midnight May 1

Testing 20 pts, working code 60 pts

Your project will to be develop a computer program for playing the game "Hex", invented by Piet Hein (and possibly John Nash) in 1942. The game is very simple to play and very difficult to master. I don't expect that any of us will write a version that can beat a human, but you are welcome to try. We can have lots of fun with simple strategies that only try to outwit each other.

The Testing

Testing represents forty percent of this portion of the project.

Every function must have a test (see the comments in test.scm). For every function, determine boundary conditions and test them. Do not error-check incoming parameters in real functions[1], only in test functions. In other words, if a game function says it takes a hex, do not try to pass it anything else in your testing.

The initial testing grade will be based on completeness and correctness. The second testing grade will be based on those and also on the runs of your automatic testing of all code. Your test code may also be run on my code to be sure you have followed the structure correctly. Similarly, your code may be run against my tests.

The Game

The game board consists of an $11 * 11$[2] parallelogram of hexes. Players alternate putting Xs and Os inside hexes, with X going first. Player X tries to connect the top and bottom of the board with an unbroken line of Xs; similarly player O tries to connect the sides of the board. Either player may occupy corners, which count as either top, bottom, or sides. Once a hex has been played, it may not be changed.

For the purposes of uniformity, let us assume that the board is oriented with a side flat to the bottom of the screen, and the top right corner pointing to the right, while the bottom left corner points left.

Two coordinates will identify any hex. Picture an x-axis along the bottom numbered $0 - 10$ and a similar y-axis going up the left side.

[1] Why not?

[2] We'll stick with eleven, but the game is defined for any n.

The Project

I will put an FAQ online; check there for answers before you send me a question.

I have provided a Scheme representation of the game using data abstraction. There are hex, board, and AI layers. I have specified areas where the layers may interact. Don't have layers interact unnecessarily (use my definition, not yours).

The only language restrictions are that your whole program must be functional and recursive in nature, i.e. *no assignment or loops.*

A player will tell the program which square they wish to play by passing in a list containing an x-coord and a y-coord (see the "read" primitive on page 383 of AS&S). Your code must keep track of who has played where, and will not allow illegal moves (e.g. (2 13) or (-1 7), or on top of an existing move).

I will provide instructions for a very limited strategy involving clusters, which you will eventually augment.

- What strategies are practical to implement? Which are not?
- What about more than one strategy simultaneously?
- How will you compare the relative goodness of two possible moves? We discussed this at length in clas - be sure you can think critically about the kinds of issues we discussed.

You do not need to be able to answer these questions fully at this point. But these questions can help you design your code in a way that will save you time and effort later. Think abstract, think modular, think components!

Your code for this part will be graded largely on your style, i.e. data abstraction, procedural decomposition, clear naming, reasonable efficiency, etc. Your code must be accompanied by brief but clear (to us, not to you) comments explaining each function.

Tips:

- Have one simple idea per procedure. Many short procedures in a hierarchical design are much more likely to be flexible than large procedures. My longest procedure for the game is 20 lines long with comments, but most of my procedures are one to four lines.
- The sequence toolkit can save you *piles* of work.
- Go over your design with teammates before you start to code. Many heads write better code, which is why the software industry uses strategies like pair-programming.

Not yet, but...

Eventually we'll be able to play against each other. Think about the efficiency of your code; we may have to implement time limits which could penalize inefficient code. All code will have to demonstrate some ability to strategize.

Keep in mind the principles of data and procedural abstraction, test, test, test, and have fun!

If you choose to use graphics, use the world.ss package.