Lab 9

Programs

1. Code breadth-first search, as described in class and here. First, use the Unix cp[1] command to get the support file from the directory.

   `/www/htdocs/CIS/280/tharvey/09S/labs`

   The support file is named lab_scm.zo

   Place the file in a directory named **compiled** inside your lab09 directory. This is a special name for the **require** function.

   Now, inside your scheme file you can say

   ```
   (require "lab.scm")
   ```

   and require will look in the compiled directory and find the binary support file even though it has a different name.

   Use the functions in the support file (get-surrounding-hexes, is-valid-hex?, make-hex, and on-edge?)[2] to write breadth-first search. If you cannot figure out from the names or your project descriptions what these functions do, then try using them in the interpreter and see what they do. The parameter expl is the next hex to "explore". Exploring a hex means 1) seeing if it is a solution (i.e. it is on the edge of the board); if not, 2) adding its children to the back of the list of nodes to explore. Children always get added to the END of the list because we want to check every member of level n before we start to check members of level n+1.[3]

   `(define (breadth-first expl node-list num-tested) ...`

   As discussed in class, the project requires being able to determine how "good" a particular hex is as a next move. One measure of goodness might (or might not) be how many steps it will take to get from the hex to the edge.

   Breadth-first search looks at an entire level of a tree before examining any members of the next level. This means that it will always find the closest edge before it finds one further away, a very nice property. We can view a hex as the root of a tree, with each legal neighbor being a branch. Thus a center hex of an empty board has six branches.

   ```
   > (breadth-first (make-hex 5 5) () 0)
   edge found at num-tested 1555 ((0 . 10))
   > (breadth-first (make-hex 5 10) () 0)
   edge found at num-tested 0 ((5 . 10))
   ```

[1] Because it is good practice and because cut and psate will mess up the binary.

[2] Do you need all of them?

[3] If you place the child elsewhere in the list, it is not breadth-first, it is another kind of search. Use breadth-first for this lab.

(a) Code breadth-first very simply, as described in above. Your results should be identical to those shown. Note that these results don't look very good. Why? Can you determine the complexity of this solution?

(b) Fix breadth-first so that your numbers look more reasonable (check them with Peng). If you add or modify functions, be sure you maintain data abstraction. How expensive is your solution? Think about possible ways to minimize that expense when you use similar code in your game (but don't optimize until you have a working game!).

2. AS&S 3.1

3. AS&S 3.2

4. AS&S 3.3

Due Sunday night as usual.