CS1112 Fall 2012 Project 6 Part C due Thursday 11/29 at 11pm

You must work either on your own or with one partner. If you work with a partner you must first register as a group in CMS and then submit your work as a group. *Adhere to the Code of Academic Integrity.* For a group, "you" below refers to "your group." You may discuss background issues and general strategies with others and seek help from the course staff, but the work that you submit must be your own. In particular, you may discuss general ideas with others but you may not work out the detailed solutions with others. It is not OK for you to see or hear another student's code and it is certainly not OK to copy code from another person or from published/Internet sources. If you feel that you cannot complete the assignment on you own, seek help from the course staff.

Objectives

Completing this project will solidify your understanding of structs and struct arrays (Part A), object-oriented programming (Part B), and recursion (Part C).

Parts A and B (questions 1 and 2) appear in separate documents.

1 H-Tree

In this problem you will write a recursive function `drawHTree` to draw an "H-tree." You will use recursion—no loops! The idea is that you start with one 'H', then add four smaller 'H's, one at each end of the previous 'H', then add four more 'H's at each end point, and again, and ...

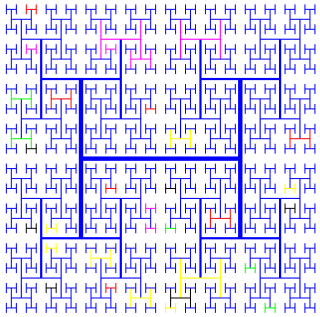

Figure 1: Level 5 H-tree

The H-tree is isn't just an interesting fractal that one can draw; it is the pattern used in laying out an interconnection network on microprocessor chips! Such a network has the feature that at any "level" the ends of the 'H' are the same distance from the center of the network. Consider a 2-level H-tree. The ends

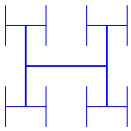

Figure 2: Level 2 H-tree

of the smaller 'H's are the same distance from the center of the network. In microprocessor chip design this is important because the delay in routing time signal from the center clock to each part (end point) of the network is then the same. The H-tree is also a space-efficient layout for cramming components onto a chip!

Function `drawHTree` has the following specifications:

```
function drawHTree(x,y,len,w,level)
% Draw recursively an H-tree centered at (x,y).
% Each of the three lines of the 'H' has length len and width w.
% level is a non-negative integer indicating the level of the recursion.
```

At each level of recursion, the length and width of each line of the 'H' are halved. Therefore the level of recursion corresponds to the number of different sizes of 'H's in the tree. The level 5 H-tree in Figure 1 shows five different sizes of 'H's; the level 2 H-tree in Figure 2 shows two different sizes of 'H's. From Figure 1, you see that most of the 'H's are drawn in blue. Write your function so that there is a *one in ten* probability that an 'H' is drawn in a color that is not blue. In our implementation, one of the following colors are chosen randomly when a "non-blue" color is needed: black, red, yellow, magenta, green. (*Hint*: we used the string `'krymg'` somehow.) You can choose your own colors, but your code should allow for at least four different colors to be used.

Start by reviewing `MeshTriangle` in §14.1 of *Insight*. Your function `drawHTree` should have a very similar organization. In fact, `drawHTree` is simpler because there's nothing to do when the minimum recursion level (0) is reached. Use the `plot` function to draw each line. For example,

```
plot([x1 x2], [y1 y2], c, 'Linewidth', w)
```

draws a line of width `w` between (`x1`,`y1`) and (`x2`,`y2`) in color `c`, where `c` is an rgb vector or a predefined color name such as `'r'`, `'b'`, ..., etc. (Note: Don't worry about `plot`'s minimum line width. As long as the calculated line width is positive, `plot` will produce a line even though at some point the lines will stop getting thinner visually.)

As usual, decompose the problem! Start by drawing the H-tree in just one color and with one line width. Test your function to make sure that it draws the structure correctly! Be sure to check the base case, i.e., `level`=0 should produce *no* 'H'. `level`=1 should produce one 'H', `level`=2 should produce a tree with two different sizes of 'H', and so forth. After you have tested your function to see that it draws the structure correctly, then deal with the line width and implement the random color functionality.

Submit your function `drawHTree`. You can assume that `hold` is on and axes are appropriately set. For example, we can call `drawHTree` with the following script:

```
% Show drawHTree
close all
figure
x= 0;
y= 0;
len= 10;
width= 5;  % corresponds to the "linewidth" property in function plot
level= 4;
axis equal off
hold on
drawHTree(x,y,len,width,level)
hold off
```

Submit your function file `drawHTree.m` in CMS.