CS1112 Spring 2015 Project 6 Part B **due Wednesday 5/6 at 11pm**

(Part A appears in a separate document. Both parts have the same submission deadline.)

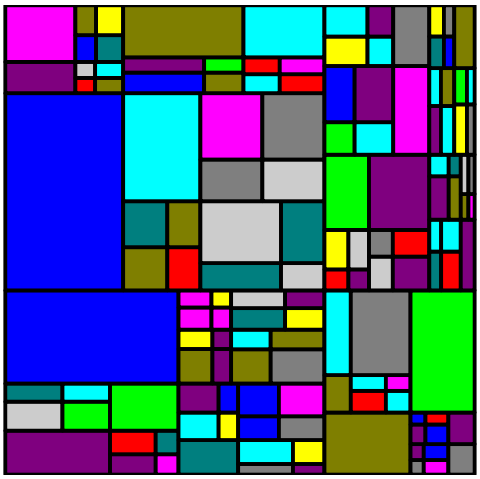
You must work either on your own or with one partner. If you work with a partner you must first register as a group in CMS and then submit your work as a group. *Adhere to the Code of Academic Integrity.* For a group, "you" below refers to "your group." You may discuss background issues and general strategies 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 exercise your knowledge of object-oriented programming (Part A) and recursion (Part B).

2 Random Mondrian

This problem is an expanded version of *Insight* Exercise P14.1.7. In this problem you will write a recursive function `RandomMondrian(a,b,L,W,level)` that produces Mondrian-like designs of the form



The general appearance of the partitioning will be determined by how you address a number of design issues that relate to the display of the individual tiles and the subdivision process whereby they are obtained. **Before you begin, read** *Insight* **§14.1**! It'll help you solve this problem as the implementation of `RandomMondrian(a,b,L,W,level)` is structured similarly.

```
function RandomMondrian(a,b,L,W,level)
% Display an L-by-W random Mondrian with lower left corner (a,b) using
% RECURSION.
% Level is the level of partitioning, 0<level<7.
% Assume that hold is on and that the figure window is appropriately set.
```

You can assume that your function will be called by a script *similar* to this:

```
close all
figure
set(gcf,'color','k','position',[50 50 800 600])  % Fixed
hold on
axis equal off manual
RandomMondrian(0,0,1,1,5)  % Different arguments may be used
shg
```

We will definitely use a black background with the figure window size given in the above example script, but different arguments may be used in the call to `RandomMondrian`.

You are to incorporate certain features in `RandomMondrian` that make the final partitioning both unpredictable and aesthetically pleasing:

- The decision to subdivide the input rectangle must depend in part upon a simulated coin toss. This induces variability in tile size.
- If the input rectangle is to be displayed, then it should be bordered in black and randomly colored. The border width and the set of allowable tile colors are for you to decide. No right or wrong—just make the tiles look nice.
- If the input rectangle is to be subdivided, then the "partition point" is to be randomly located. The partition point determines the four sub-rectangles and you must take steps to guard against the production of "super skinny" tiles that are aesthetically unappealing.

Guidelines associated with these design aspects follow.

2.1 The Subdivision Coin Toss and Related Issues

In `RandomMondrian`, the decision to subdivide the input rectangle is not based on `level` alone as is the case in `MeshTriangle` in *Insight* §14.1. `RandomMondrian` is to stop recursion if `level` is zero *or* if the simulated toss of an unfair coin "comes up heads." It is up to you to choose the probability for heads, which we refer to as `pHalt`. By playing with the value of `pHalt` you will discover that its value has an influence upon the range of tile sizes in the final partition. Tile size generally decreases with each level of subdivision. Thus, larger tiles can result by allowing the subdivision process to be "called off" before the recursion level reaches 0. Set `pHalt` to a value that you think produces a nice mix of tile sizes.

We mention here that MATLAB has a default recursive call limit of 500. This limit may be changed (increased) by the user, but it is there to remind us that *recursion is memory intensive!* Since a (recursive) function call does not end before another recursive call is made, memory usage increases fast with the number of recursive calls made. In `RandomMondrian` we assume 0 <`level`< 7 because of aesthetics, but in your experimentation you can try higher values of `level` to see how much *time* it takes to produce a Mondrian (knowing that it wouldn't look good). The growth of run-time and memory usage is *exponential* to the recursion level! Our use of "early termination" based on `pHalt` reduces the time and memory usage somewhat, but the load is still significant.

2.2 The Display of an Individual Tile

Tiles are to be colored by randomly selecting from a set of "allowable" colors. Proceed by assembling the rgb values of your chosen colors row-by-row in an array `MyColors`. For example, if in the partition you only want tiles that are either red, medium gray, mauve, green, or blue, then set

```
MyColor = [ 1.0 0.0 0.0;...
            0.5 0.5 0.5;...
            1.0 0.0 1.0;...
            0.0 1.0 0.0;...
            0.0 0.0 1.0]
```

`MyColor` is said to be a "color map." The act of displaying the input rectangle with a random allowable color becomes the act of choosing a random row in the color map and then passing it together with the vertex information to built-in function `fill`—for this part of this project, learn to use `fill` directly and do not use the functions `DrawRect` and `DrawRectNoBorder` provided for Part A. For your information, a colormap of 11 colors was used to produce the figure on page 1. You can produce nice designs with our "basic colors" `100`, `010`, `001`, `110`, `101`, `011`, and `111` and simple scalings thereof. It is up to you to set up a color map of your own choosing, but it should have at least five colors. Do not include black because your final design will be displayed against a black background and we want to clearly see the outer edge.

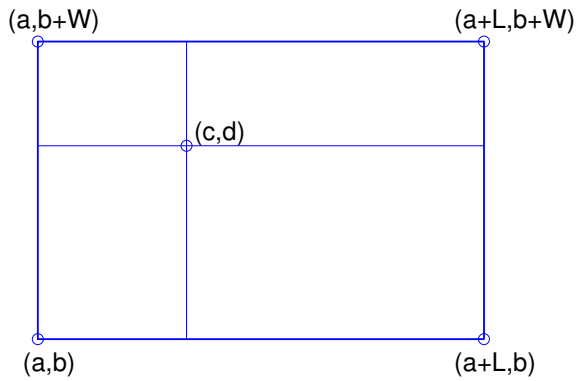
An easy way to get a black border of width `bWidth`, chosen by you, is to use the `Linewidth` property of `plot`:

```
fill([a a+L a+L a a],[b b b+W b+W b], [0 0 1])
plot([a a+L a+L a a],[b b b+W b+W b],'k','Linewidth',bWidth)
```

The above fragment plots a blue `L`-by-`W` rectangle with its lower left corner at (`a`,`b`) that is bordered by black lines of width `bWidth`. For your information, MATLAB's default line width for a plot is 0.5.

2.3 Rectangle Subdivision

The partitioning of the input rectangle into four smaller rectangles requires the selection of a "partitioning point" (c, d):



This should be done randomly subject to the constraint that the point is not too near the edge of the input rectangle. This reduces the chance of creating super-skinny tiles in the final partitioning. It is up to you to figure out how to parameterize this and how to set the value of the parameter(s).

Final Note

The act of identifying design parameters and choosing their value is central to computational science and engineering. In this problem, the design parameters are `pHalt`, `MyColors`, and `bWidth`, together with the design parameters associated with your method for choosing the partition point. *Highlight these values by placing the assignments to these variables at the beginning of your* `RandomMondrian` *implementation.* It is imperative that you include appropriate comments.

Finally, you are free to implement "helper" functions if it is handy. Include them as subfunctions to `RandomMondrian` so that you only submit the one file to CMS.