CPSC 449 Assignment 1

Due: Monday June 5, 2017 at 4:00pm

Individual Work:

All assignments in this course are to be completed individually. Students are advised to read the guidelines for avoiding plagiarism located on the course website. Students are also advised that electronic tools may be used to detect plagiarism.

Late Penalty:

Late assignments will not be accepted.

Posting Submissions for Public Viewing:

This assignment includes a creative and artistic element. As a result, we are hoping to receive numerous interesting submissions that will be worthy of showing off. We plan to post the images that are created on the course webpage so that others can view them. Your image will be posted anonymously, unless you choose to include your name as part of the image that you create. Please do not put your student number on your image. If you are not willing to have your image included on the website then please send an email to ben.stephenson@ucalgary.ca clearly stating such when you submit your assignment.

Background:

Piet Mondrian (March 7, 1872 – February 1, 1944) was a Dutch painter who created numerous famous paintings in the early half of the previous century that consisted of a white background, prominent black horizontal and vertical lines, and regions colored with red, yellow and blue. Three examples are shown below:

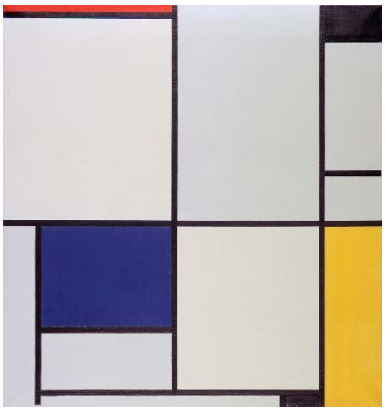
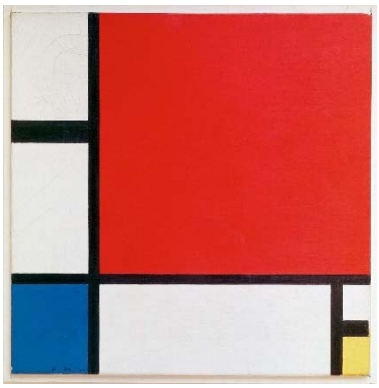
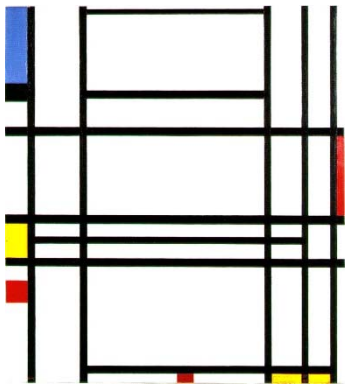


Tableau I, 1921



Composition II in Red, Blue, and Yellow, 1930



Composition No. 10, 1939-1942

Assignment Task:

Your task is to write a Haskell program that uses recursion to generate pseudo-random art in a Mondrian style. Your program's output will be an HTML document that contains rectangle primitives (perhaps among others) within an SVG tag. The following general strategy can be used to generate art in a Mondrian style:

If the region is wider than half the initial canvas size and the region is taller than half the initial canvas height:

    Use recursion to split the region into 4 smaller regions (a vertical split and a horizontal split) with the split location chosen randomly

Else if the region is wider than half the initial canvas size:

    Use recursion to split the region into 2 smaller regions using a vertical line with the split location chosen randomly

Else if the region is taller than half the initial canvas size:

    Use recursion to split the region into 2 smaller regions using a horizontal line with the split location chosen randomly

Else if the region is big enough to split both horizontally and vertically, and both a horizontal and vertical split are randomly selected:

    Use recursion to split the region into 4 smaller regions (a vertical split and a horizontal split) with the split location chosen randomly

Else if the region is big enough to split horizontally, and a horizontal split is selected:

    Use recursion to split the region into 2 smaller regions using a vertical line with the split location chosen randomly

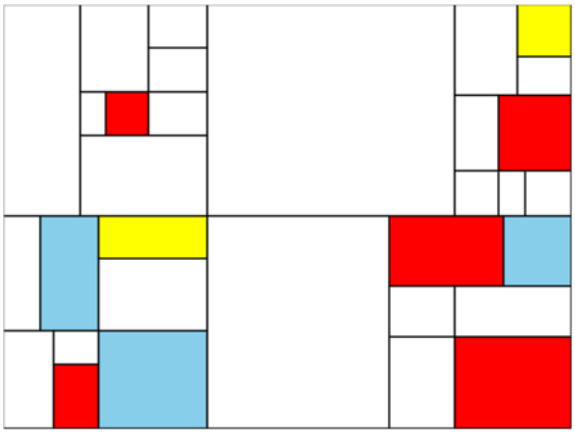
Else if the region is big enough to split vertically, a vertical split is selected:

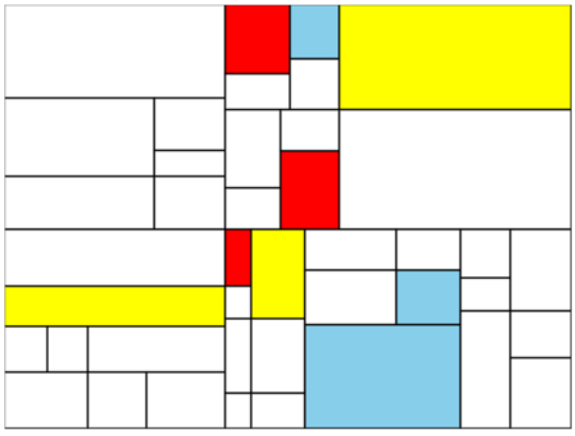
    Use recursion to split the region into 2 smaller regions using a horizontal line with the split location chosen randomly

Else:

    Fill the current region (randomly, either white or colored, and if colored, with a random determination of red, blue or yellow)

A couple of images generated using this general algorithm (with my own specific constant values for deciding the frequency with which regions are split, the locations of the splits, the colors and when a region is big enough to split) are shown below:

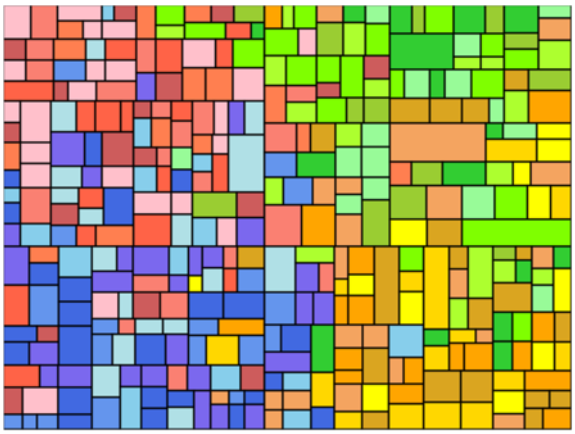


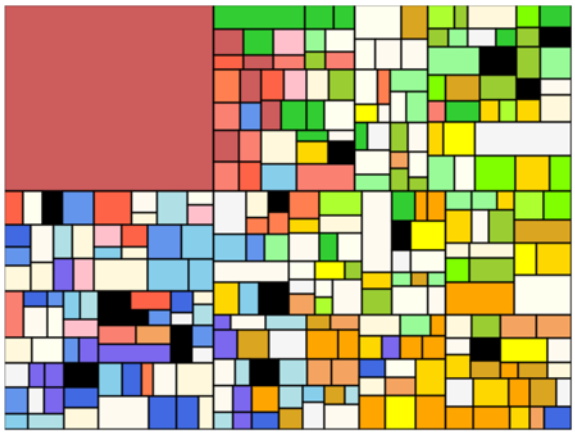


You are encouraged to adjust / expand / adapt this algorithm to generate art with your own specific style provided that is still at least vaguely Mondrian in style (meaning that it largely consists of horizontal and vertical lines and colored regions, at least the majority of which are rectangular). Ideas for customizing your work that you might want to consider include:

- Using lines of variable width
- Using a broader color palette than red, yellow and blue for the filled regions
- Changing the distribution of random numbers used when selecting the sizes of regions, colors (or anything else random). For example:
    - Instead of using numbers that are an even random distribution, you could add two random numbers together and divide by two to form a distribution that favours numbers in the middle of the random space.
    - Instead of allowing all possible values, reduce the space to numbers that are evenly divisible by 10 (or 20 or some other number) so that the random lines have more regular spacing to them.
- Using a patterned fill for some regions instead of only using solid fills
- Occasionally splitting a region into something other than rectangles
- Occasionally split a region into 3 smaller regions instead of 2 or 4

For inspiration, here are a couple of other images that I generated with different variations of my solution. In both of these images, the color used to fill the region is influenced (but not fully determined) by its location:





Implementation Details:

Randomness is an essential part of this assignment, but a function that returns a random number is contrary to the spirit of a pure functional language because a pure functional language requires each function to always return the same value when it is provided with the same parameters. While Haskell provides mechanisms for producing random numbers, they require concepts that we won't be exploring in detail in this course. To work around this limitation, I have provided some code for you to use on the course website. While you are free to modify this code if you find it helpful to do so, it will work as written (other than requiring you to replace the body of the mondrian function).

As you implement the mondrian function (and additional helper functions), you will likely find yourself needing to write mathematical expressions that mix integers and floating point numbers. To convert from an integer to a floating point number, use fromIntegral – it takes an integer as a parameter and returns the equivalent floating point number. To convert from a floating point number to an integer, use round – it takes a floating point number as a parameter and returns the closest integer.

Grading:

A base grade will be determined for your assignment based on its overall level of functionality. Then that grade may be adjusted if your implementation fails to use functional programming constructs in a reasonable manner.

D	Uses Haskell to generate an output file that can be rendered in the browser and demonstrates the ability to draw something non-trivial
C	Uses recursion, but not randomness, to generate something that has a Mondrian style
B	Uses recursion and randomness to generate something that has a Mondrian style, but uses an approach that is somewhat simpler than what was outlined in the assignment description
A-	Implements the approach outlined earlier in the assignment description to generate something that has a Mondrian style
A	Implements modest extensions / improvements to the approach described in the assignment handout to generate something that has a Mondrian style
A+	Substantially extends / improves the approach described in the assignment handout to generate something that is artistically impressive and has a Mondrian style

Note that Mondrian style will be interpreted relatively loosely. You should feel free to experiment and explore, both with the features of Haskell as a programming language, and with your artistic style.

Submission Instructions:

Submit your .hs file to the Assignment 1 drop box in D2L. Since the results from your program are random, you may also want to submit the html file for your favourite image (or two) that your program has generated because we will (probably) not see the same image(s) when we test your program.