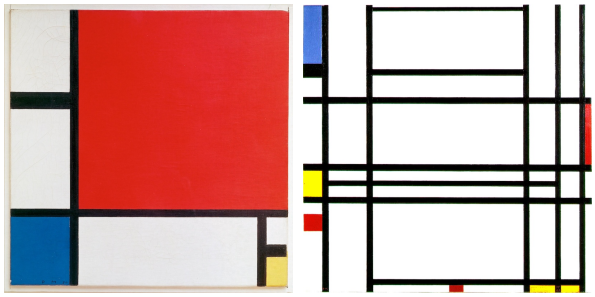
Computational Problem-Solving CSCI-603
Mondrian Recursion Lab 3

08/28/2024



1 Introduction

Continuing with the idea from problem-solving of subdividing squares using recursion, you will use the Python's turtle package to create images similar to the famous paintings of the artist *Piet Mondrian*.

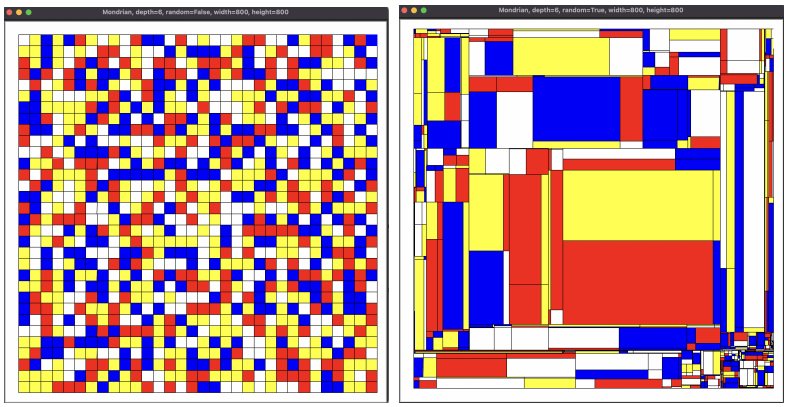
Piet Mondrian (1872-1944) was a Dutch modern abstract artist. Some of his most famous paintings, on display in the *Guggenheim Museum* in New York City, combine the simplest of geometric elements - straight lines, right angles, and primary colors. It's cleanliness and purity embodied his belief in a utopian and harmonious universe.

2 Implementation (75%)

You will **individually develop** a program called `mondrian.py` that will generate a drawing comprises with rectangles of random colors simulating a Mondrian painting.

The program will generate the drawing following one of these two approaches:

- Approach 1: The drawing is generated by subdividing the canvas in uniform sized rectangles of random colors. The image on the left in the next page is a sample run of this approach with depth 6.
- Approach 2: The drawing is generated by subdividing the canvas in randomly sized rectangles of random colors. The image on the right in the next page is a sample run of this approach with depth 6.



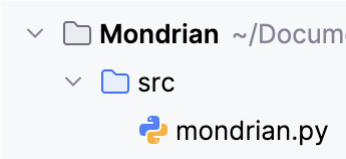
2.1 Starter code

Get the starter code from here:

`https://www.cs.rit.edu/~csci603/Labs/03-Mondrian/code.zip`

Extract the zip file, and then copy the `src` directory into the root of your project folder in PyCharm. To make PyCharm aware the `src` directory has source code, you should right click on the `src` directory and select **Mark directory as... Sources root**. The `src` directory should be a blue color.

Your project structure should look as follows:



Verify that you can see the starter code, `mondrian.py`, in the `src` sub-directory. Run the program by right clicking in the source code window. It should run but do nothing.

2.2 `init` function

Take a moment and look at the `init` function which is already written for you. It sets up the world coordinate system as $(llx = -20, lly = -20, urx = 820, ury = 820)$. The window, as well as the area you should draw your rectangles, is set up as $(llx = 0, lly = 0, urx = 800, ury = 800)$. These use the global constants `WIDTH` and `HEIGHT`. If the screen is too large you can make these smaller.

2.3 Program Operation

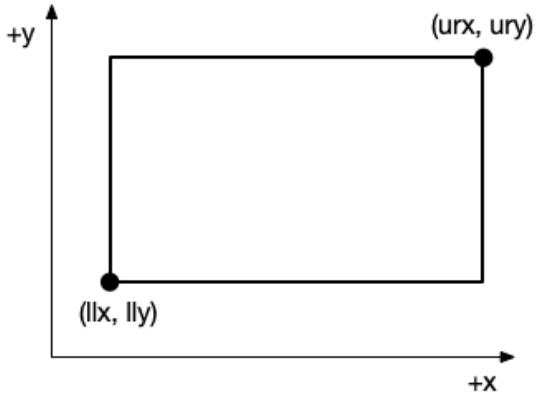
The program will prompt the user to provide some information via standard input. You do not have to deal with erroneous input. It is assumed that all inputs are legal, and are all formatted correctly.

When run, the program should execute as follows:

1. Prompt for the depth of recursion. A valid depth is between 1-8 inclusive. If a value outside of this range is entered, the program should continually re-prompt the user until a valid one is entered.
2. Prompt for whether the user desires randomly subdivided rectangles (approach 2), `'y'`, or whether the rectangles will be identical in size (approach 1).
3. Recursively generate the Mondrian system with rectangles that fill choosing a random color from the `COLORS` list.
4. Display the total surface area of each unique colored rectangle, followed by the overall total surface area. **This computation must be done recursively without the use of globals!**
5. Wait for the user to close the window and end the program.

2.4 Uniform subdivision

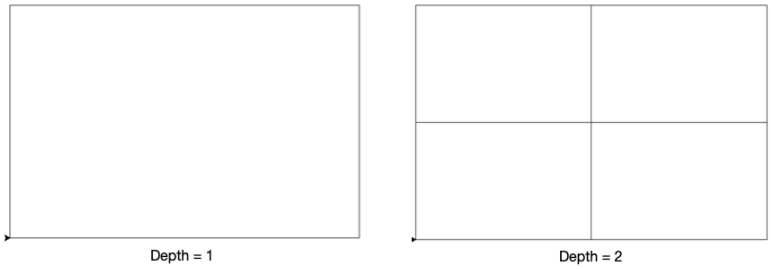
Consider the following coordinate system and function call.

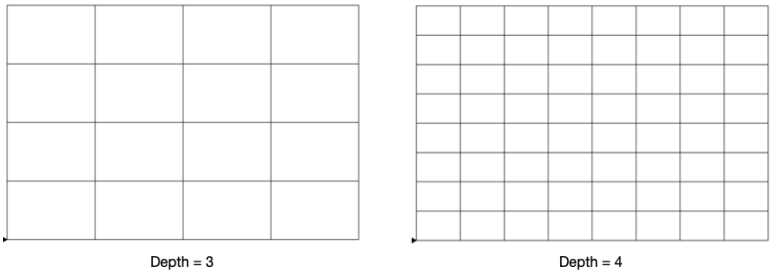



```
draw_rectangle(llx, lly, urx, ury)
```

Write a Python function, `draw_rectangle`, that takes the lower and upper coordinates as integer arguments and draws a rectangle using turtle. The *preconditions* and *postconditions* are the same - the turtle pen is at (llx, lly), is down and is facing east.

Using that function, write now a recursive function that draws rectangles that are *uniformly subdivided*. In other words, as the depth increases, the size of the rectangles decrease uniformly by one quarter. Initially you should call this recursive function with the following values: $(llx = 0, lly = 0, urx = WIDTH, ury = HEIGHT)$.

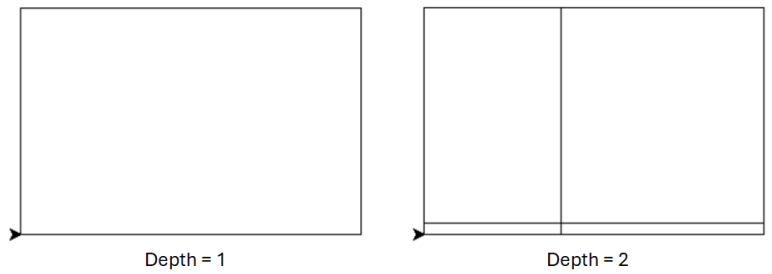


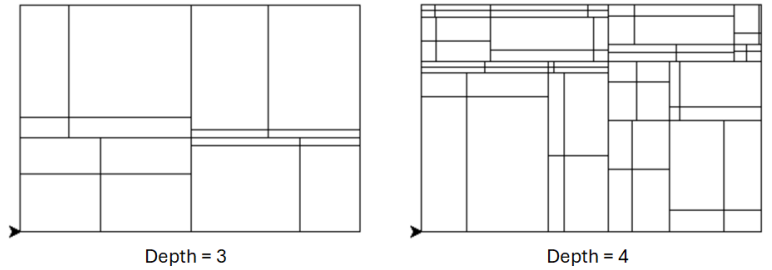


Make sure you consider the pre/post conditions carefully. It is imperative that the recursive function you write begin and end at the starting location, (llx, lly), with the turtle down and facing east. This will require you carefully retracing over the rectangle/s drawn to return to the start. **You are not allowed to teleport the turtle.**

2.5 Random subdivision

For this approach, you still need to subdivide in four rectangles, but now, the dimension of those rectangles are randomly selected at each depth of recursion. Consider the images below following this approach, notice that inside each region, all four rectangles shared one corner.





2.6 Sample Run

2.6.1 Uniform rectangles

To generate the upper left image (for example):

```
Enter depth? 6
Random subdivisions? n
Rectangle Surface Areas:
        blue: 158750
        red: 166250
        white: 162500
        yellow: 152500
Total Surface Area: 640000
```

2.6.2 Random rectangles

To generate the upper right image (for example):

```
Enter depth? 6
Random subdivisions? y
Rectangle Surface Areas:
        blue: 193124
        red: 102126
        white: 171696
        yellow: 173054
Total Surface Area: 640000
```

2.7 Implementation Details

This section will have suggestions and hints to help you implement things.

2.7.1 Filled Rectangles

In order to fill an image with the current color, use the turtle functions `begin_fill()` and `end_fill()`.

```
turtle.begin_fill()
# draw shape
turtle.end_fill()
```

2.7.2 Randomization

To select a random element from a list use:

```
color = random.choice(COLORS)    # COLORS is a list of strings
```

To select a random number between a range:

```
val = random.randint(10, 20)    # random number between 10-20 inclusive
```

3 Grading

The assignment grade is based on these factors:

- Problem-Solving: 20%
- Functionality: 60%
  - Proper user input (including order) and re-prompting: 5%
  - Uniform rectangles: 20%
  - Random rectangles: 20%
  - Surface area computation: 15%
- Design: 10% - Your implementation uses recursions and functions to promote code reuse.
- Code Style and Documentation: 10%

4 Submission

Go to your project's `src` folder and zip it up. Rename the zip file to `lab3.zip`. Upload this zip to the MyCourses Assignment dropbox by the due date.

- To zip on Windows, right click on the `src` folder and select `Send to -> Compressed (zipped) folder`.
- To zip on MacOS, right click on the `src` folder and select `Compress "src"`.