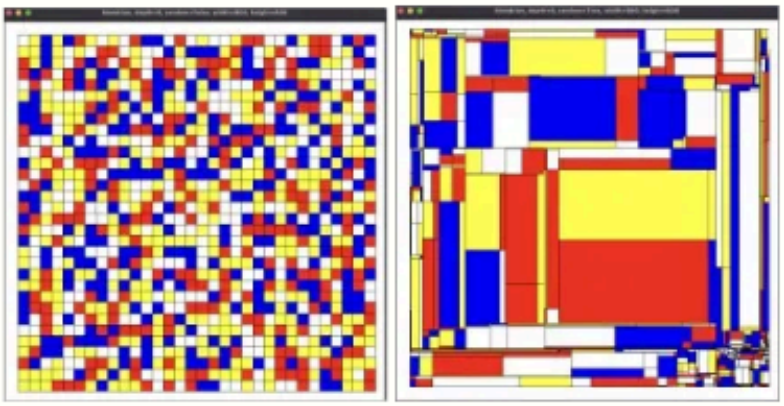
Computer Science AP/X CSCI-140/242
Mondrian Recursion Lab 2

08/28/2024



1 Implementation (75%)

You will **individually develop** a program called `mondrian.py` that implements the Mondrian recursion problem from problem solving and the in-lab activity.

1.1 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, `'y'`, or not (uniform subdivision from the in-lab activity).
3. Recursively generate the Mondrian system with rectangles that fill randomly based on 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.

1.2 Sample Run

1.2.1 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
```

1.2.2 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
```

Because the rectangles are randomly colored, the only thing to check is that the total surface area of all rectangles adds up to the dimensions of the drawing area, e.g. $(WIDTH * HEIGHT)$.

1.3 Implementation Details

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

1.3.1 Filled Rectangles

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

```
turtle.fillcolor('blue')   # blue shape
turtle.begin_fill()
# draw shape
turtle.end_fill()
```

1.3.2 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, inclusive use:

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

If you want to debug your program and get random values that are the same, you can seed it with a fixed number so that calls to `randint()` will always generate with the same values and order:

```
val = random.seed(10)
```

1.3.3 Faster Animation

When you are confident with your program, you can speed up the animation by turning the tracer off. Replace the call to `turtle.speed(0)` in `init()` with:

```
turtle.tracer(0, 0)
```

In addition, you should add a call to the end of your `main` function, just before calling `turtle.mainloop()`:

```
    turtle.update()
```

2 Grading

The assignment grade is based on these factors:

- Problem Solving: 15%
- In-Lab Activity: 10%
- Functionality: 55%
  - 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%

3 Submission

Go to your project's `src` folder and zip it up. Rename the zip file to `lab2.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"`.