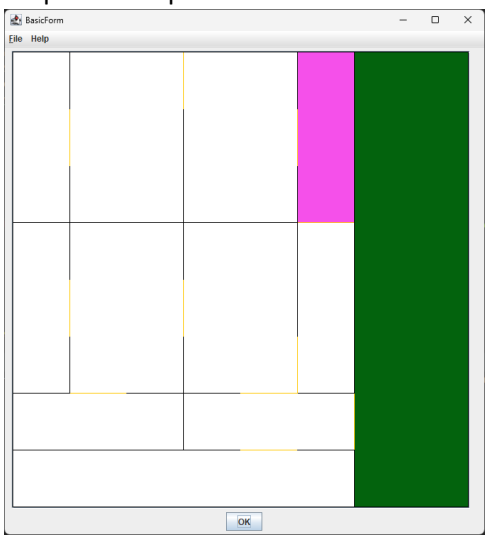
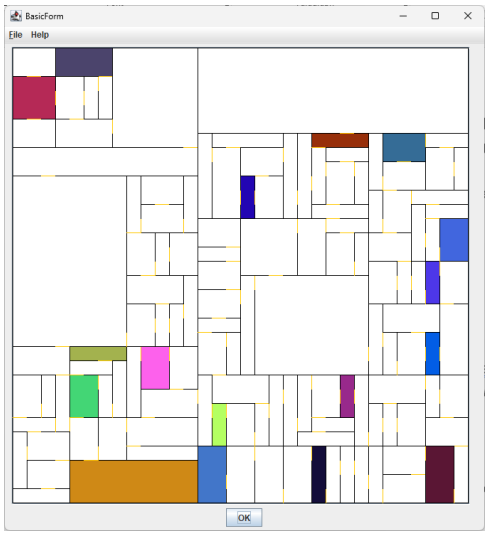
COSC 1P03 — Assignment 4 — Recursion and Abstraction and Also Art

Okay, so hear me out: I want to design layouts for the floor plans of buildings, but I *also* want to express my appreciation for Piet Mondrian and neoplasticism! This means I need to divide up the floors of a building, but I won't be going for the traditional "hallways" and "functional workplace" approaches (because those are *boring*).

Let's look at a couple quick examples:



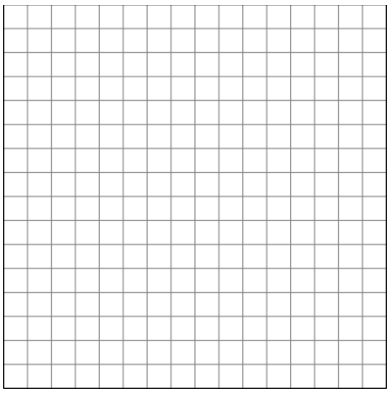


Here, you'll see that we've broken the floors up into arbitrarily-separated rooms. As noted above there's no hallways, but there's still the ability to travel from *anywhere* on the floor to *anywhere* else.

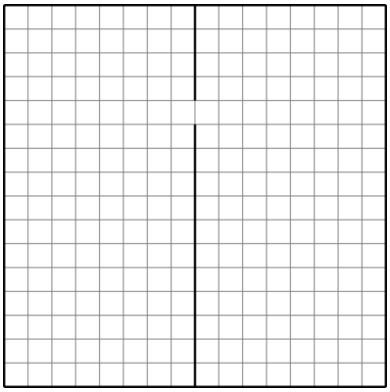
The principle is actually pretty simple:

- We treat the available space on a floor as effectively a grid of cells
  - That grid may have arbitrary granularity (hence the two examples above: same dimensions; different cell sizes)
- The grid does **not** include the walls; instead the walls are boundaries *between* spaces
- We start with initially one large room
- Any given room may be 'split up' into two rooms, with a door along the newly-added wall separating them
  - Since it's recursive, those two rooms may themselves each be split up into two... you get the idea
  - A 'wider' room will be split *horizontally* by a *vertical* wall; a 'taller' room will be split *vertically* by a *horizontal* wall. Since there's no compelling reason to pick either for a 'square' room, we'll just say that *also* gets split *horizontally* by a *vertical* wall
- We'll get to the 'cut-off' (stopping point) a bit further down

Let's walk through a couple of the first steps to start one:

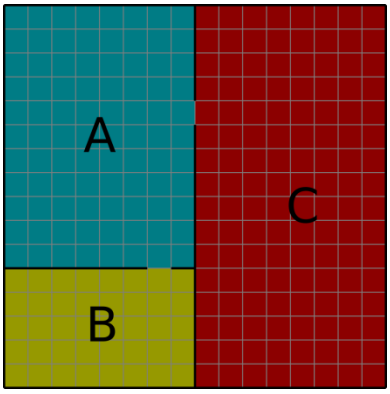


Here, we've decided to divide our space into 16×16 spaces. As it's a square, we'll favour a vertical line. We have 15 possible positions for a wall (between any two cells). Just for ha-ha's, let's start with the middle?



Here you can see we've added the door (depicted as a 'gap' for now) *roughly* ¾ of the way up.
We would then recurse onto *both* of these rooms.

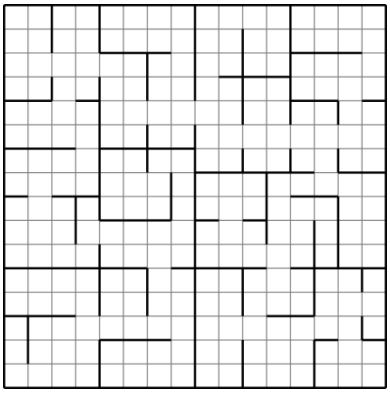
The left room is 'taller', so we split it vertically with a horizontal wall:



(The colours and letters are just for easier reference)
Notice that:

- The doors may be placed anywhere along the wall, so long as they're within the 'grid' boundaries, and always 'one space'
- Though we can further and further subdivide A/B, we'll *eventually* need to return to C to split it up too
- We'll eventually 'run out' of spaces for further subdivisions (and doors)
- Because doors are 'cut' into newly-added walls, we'll never need to touch a wall again after its creation

So, what's left? Sooner or later, they'll all be divided "enough". e.g.:



We're still missing three things: doors, the cutoff for 'when to stop dividing', and "the Piet Mondrian part".

- Doors are easy; we could've been doing them the whole time: when installing a wall and leaving a gap, simply make an **orange** line instead of the gap
- The cutoff is *relatively* simple: allow for a cutoff value where a room having either dimension at or below that value will never be further divided; also include *some* probability to stop even sooner than that. When testing, 'below 3' is a nice cutoff to try using, along with a 'quitting proclivity' of ~5%

    - This doesn't mean it's impossible for a room to have e.g. a height of 2: a room 20 tall could be divided into one of 18 and one of 2 (or even one of 19 and one of 1!)
- *Some* of the final room-floors will have their floor painted a random colour
    - This will occur with *some* probability, to be decided specifically when a room will no longer be further divided

Overall, it seems pretty doable, right? Or maybe not? This task is specifically designed to require significant planning *before* coding.

Some basic requirements are included, but as always: this relies on accumulated experience in COSC 1P03 (and 1P02). You're expected to know *basic* standards. If not, **ask**.

Basic requirements:

- First and foremost: this is a *recursive* algorithm. It must be done recursively, with recursion
    - It must also be done *correctly* with recursion. e.g. don't be peppering instance variables about, willy-nilly. That's objectively bad. Instead, remember that *parameters* are a thing!
- Some closely-related values need to be defined. e.g. the width and height of the 'floor', both in pixels and in 'cells'; but also the size of a cell, and the number of cells in each direction. *Some* of those will be calculated from the others
    - Let the user select the non-calculated ones. To be clear: this doesn't mean "let the user edit/fix your code, recompile, and do all the actual work"
- **All** UI elements are based on the Brock library you used for 1P02/1P03.
    - For the drawing, use TurtleGraphics (either directly, with a `TurtleDisplayer` sized to the correct dimensions, or by adding a 'Canvas' to a `BasicForm`). Don't use Picture for this
    - Don't forget you can set the turtle's speed (including to zero, which is paradoxically *massively* faster)
    - When it comes time to filling in the colour splashes, that's just lines of a randomly-selected colour
- You need some abstraction here
    - Of course, *procedural* abstraction: multiple methods/functions to break up the behaviour
    - Naturally also *data* abstraction. You should have more than one class here, and if you're ready to pass 1P03 that much *should* be obvious. If not, please ask questions during tutorial, and talk to your instructor *in person*
        - Specifically, the class that's dealing with the recursive algorithm will have no concern over the precise actual resolution/granularity of the 'floor'. Rather, design and implement a `Floor` to act as a simplified grouping of cells, wherein you can add divisions according to e.g. cell indices
        - Worth noting: the only instance variables I used were in my `Floor`; it wasn't necessary anywhere else
- You need to actually format your code correctly. No, seriously. If you have a bunch of garbage blank lines, or bizarre indentations, then your ***maximum*** possible grade will be reduced to 50%, but it could easily be even lower
    - The two most plausible explanations are either trying to obfuscate code, or not respecting the marker's time. Neither is acceptable conduct from an adult
- Your code *will* be commented appropriately. It's a good thing you attended lecture and paid attention, when everyone was reminded of examples of what to do (and what *not* to do)

Important note: this style of problem *could* be tackled with quite a few different approaches (like, say, roughly a dozen different categories, of many more implementations than that), but only *one* that *you* may use. In other words, please complete the task as described, to avoid a case of an 'earned zero' from 'correctly solving the wrong problem'.

Additional tidbits:

Everyone should already know these, and it's literally impossible to provide an exhaustive list, but some common things to watch out for:

- Please develop the entirety of your solution on a single machine, from start to finish within the same instance of IntelliJ. If for some reason this is not possible, include an explanation with your submission. Failure to do so will result in a zero. You may not decide to submit additional components after the assignment closes
- Similarly, when submitting, include:

- The entire project
- Some sample executions, and/or whatever would help the marker evaluate your work
- (Don't bundle the Brock library *with* your project. You should be able to configure an IDE's SDK by now, or you should seek assistance with that, in a prompt and timely fashion)
- Make sure everything's glommed together into a single .zip file (not .rar, .7z, .tar.gz, etc.)

- You'll be using the `static` keyword for your main method, only. There are many valid uses in general for that keyword, but not in *this* task. If you do it elsewhere, it's wrong. Don't be wrong
- The permitted `import`s are `java.awt.Color`, `Media.*`, and `BasicIO.*`. Nothing else
  - As mentioned in class: no, that doesn't mean you can use disallowed libraries by simply omitting the 'import' keyword
  - In fairness, if you're even tempted to use anything else, the approach is also simply wrong
- There's a Help Desk, a tutorial leader, and an instructor who very very very much likes questions

You'll notice the precise algorithm itself isn't explicitly defined; that's because it's explained. So please read carefully (and feel free to ask questions). Similarly there are values you need to receive from the user, and you need to work that out yourself. You can do this; just plan things out, and really think things through before you start coding.

Additional sample floor plans

Honestly, just because this was fun to do!