Journal Algorithms Course Q4 Nils Meijer

Task split down into steps:

1. First, create 2 equally sized rooms.
2. Either vertically or horizontally, split the rooms into 2 new parts. Do not include doors yet to keep it simple.
3. Generate a new position for the new parts, as long as they do not overlap with the “old” rooms.
4. Repeat from step 2 until 4, and until a certain number of rooms has been reached.
5. Add doors for all existing rooms.
6. Make sure the rooms do not get split at the location of doors in that room

Task/problem:

Generate multiple rooms with a random but minimal size. Connect rooms with doors. Each connection to unique rooms has 1 door.

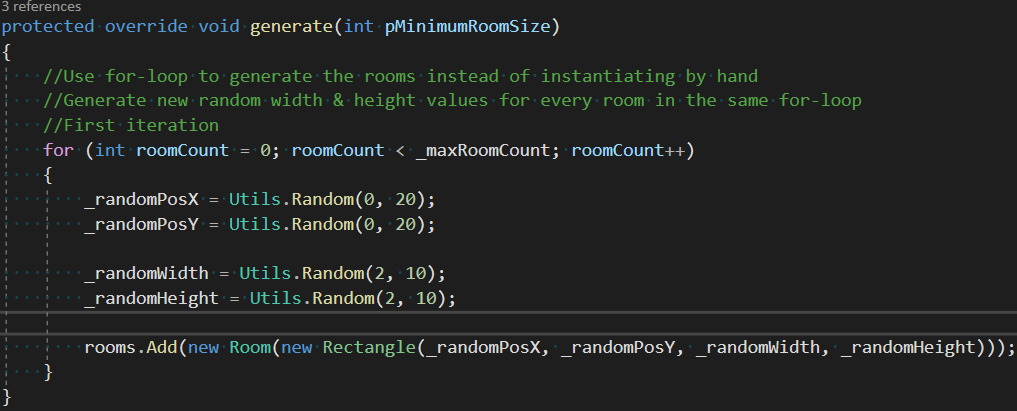
25/05/2020

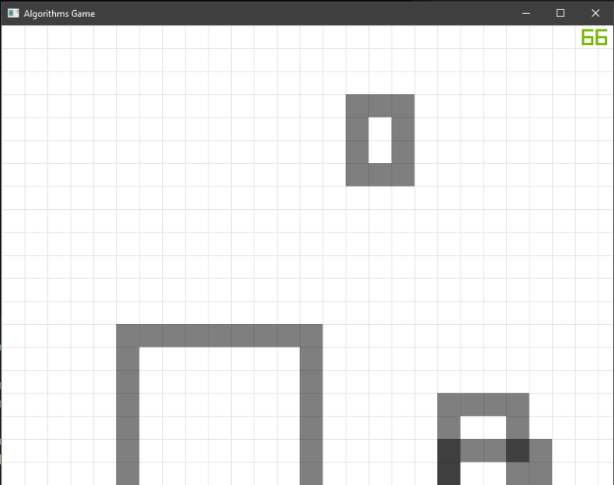
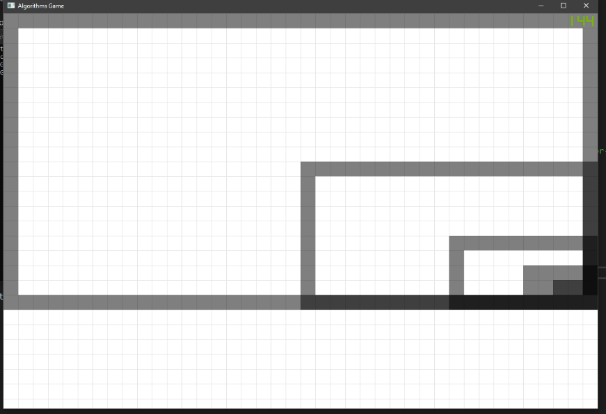
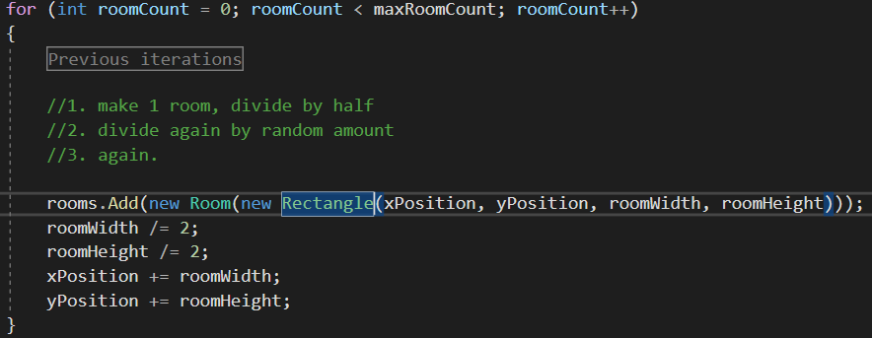
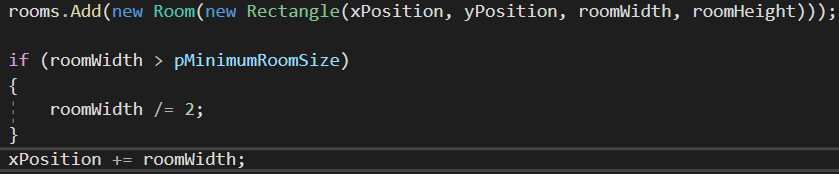
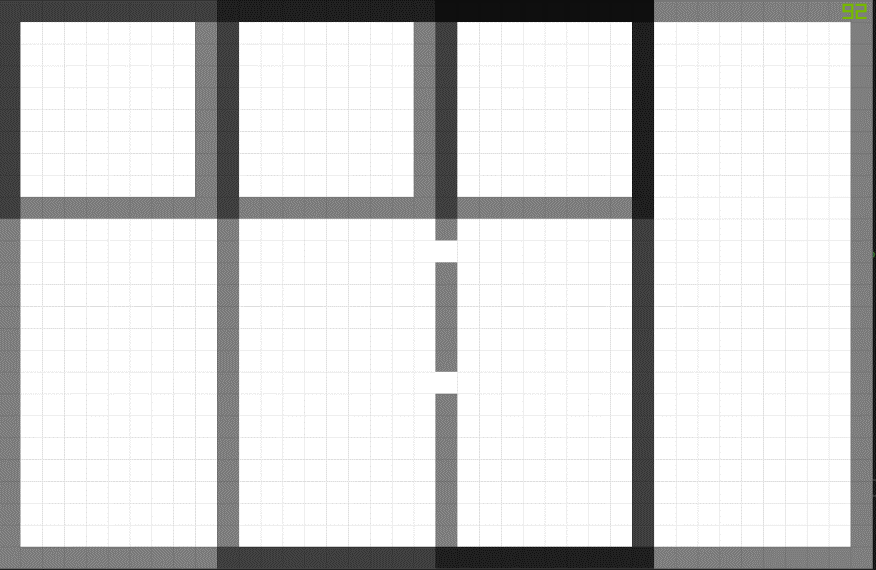
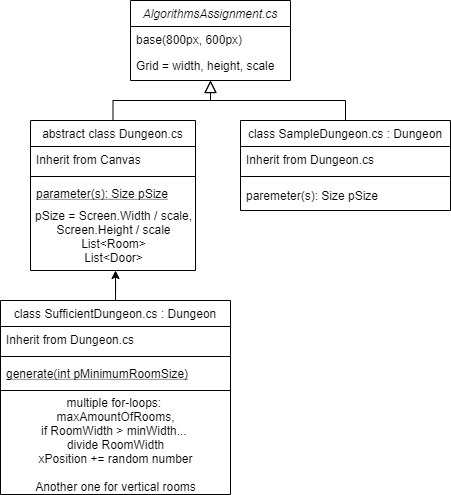
22/05/2020

Studied “Self-study” PowerPoint slides. Researched possible problem solving approaches, and see what combination of approaches fits best.

* Initialized flowchart design. Studied structure of the algorithm, what class is a child of what class.
* Where are the rooms and doors being created?
* Subclassed the Dungeon.cs with SufficientDungeon.cs. Iterated first attempt of room generation in code

Testcode:





Reworked the for-loop.

Implemented minimum room size

Finally got a bit further…

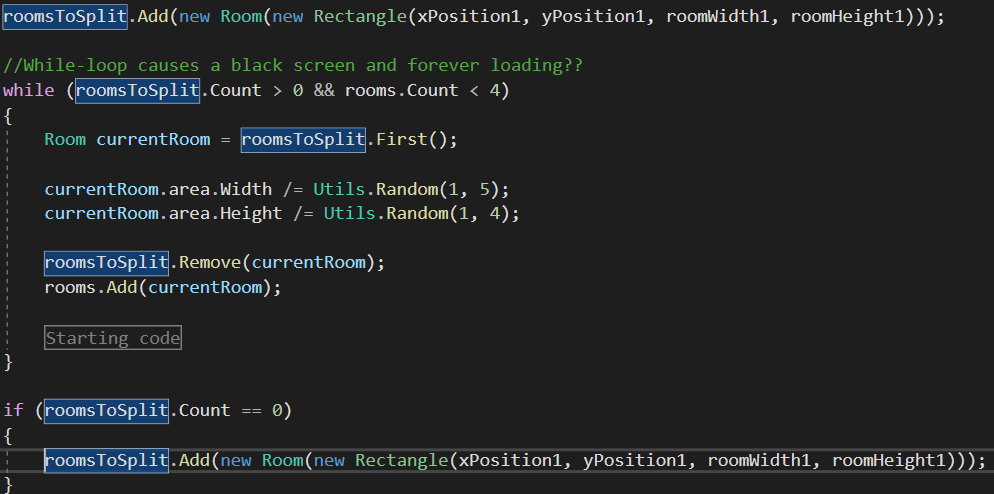
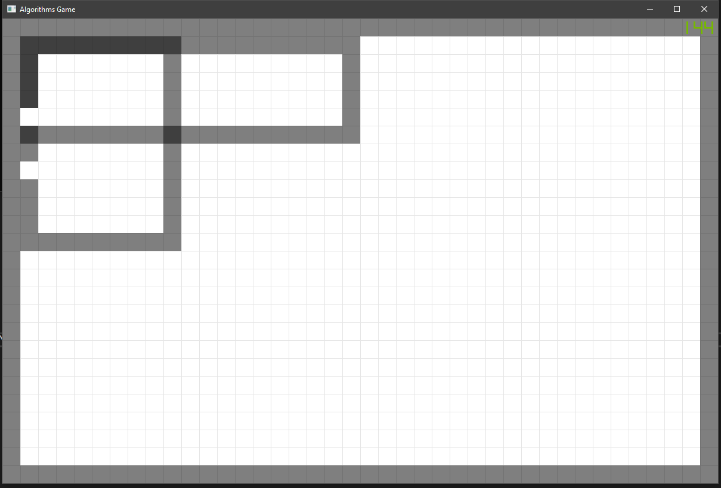
27/05/2020

First visual implementation of the dungeon generation.

Still working on making sure the rooms do not overlap.

Iterating on better room size, to prevent very small

rooms.



Fixed the “black-screen issue”. The problem  
was that the “rooms” list didn’t have any   
elements, and yet it still received access   
requests. I made sure it would never be   
empty, and a new room is generated on click.

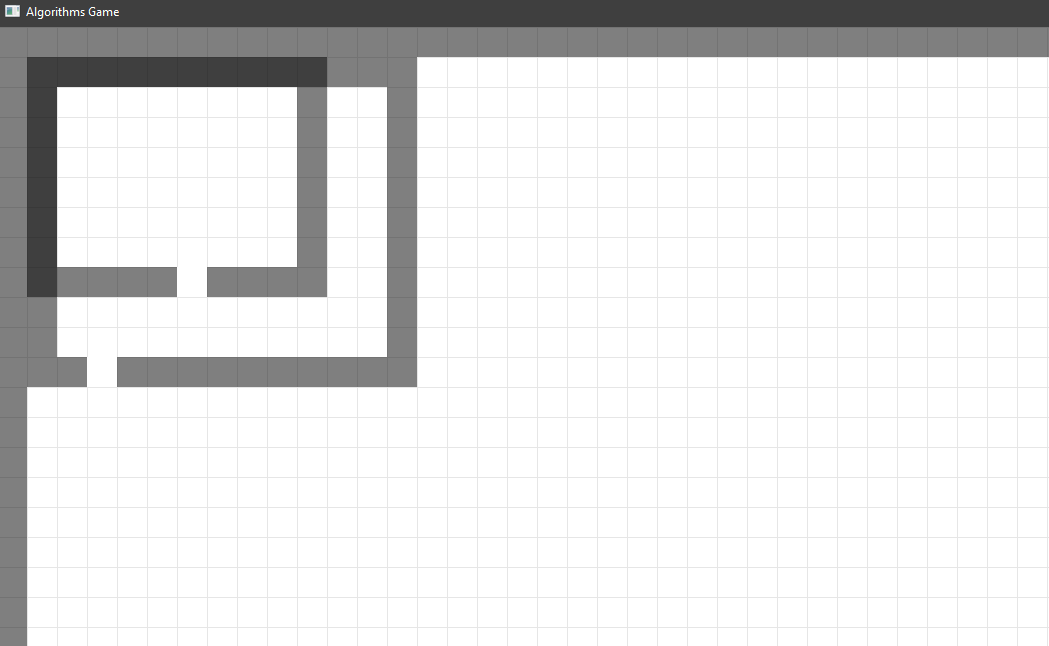
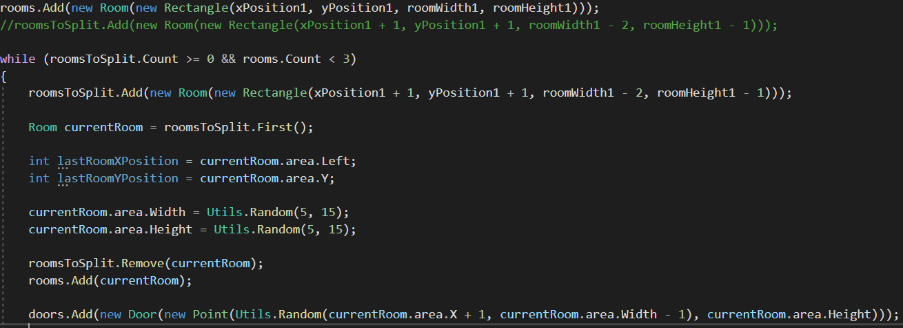
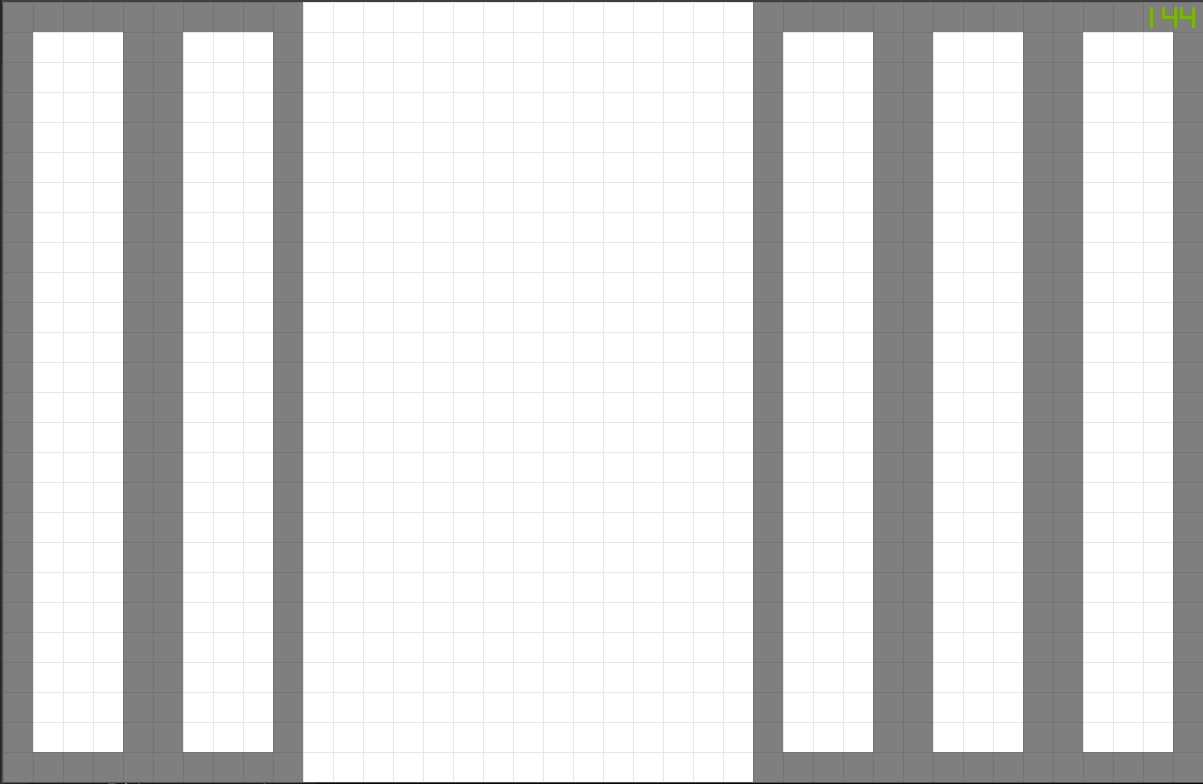
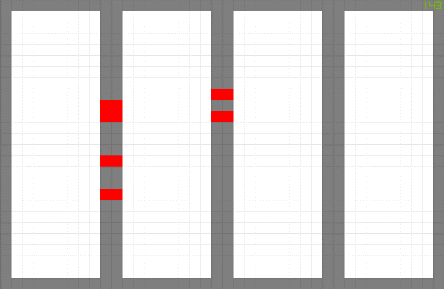
Added doors to the splitted rooms; the algorithm in itself is nowhere near perfect, but it does work.

29/05/2020

01/06/2020

With some help, I found another way to create the rooms. Currently though, the program doesn’t start (properly). As far as my current knowledge goes with different debugging approaches, it’s a problem with displaying the rooms.

The rooms *are* being created and removed,   
so that isn’t the problem.



Solved the overlapping problem during the   
lab, but for some reason the rooms are still  
randomly being positioned, even though there  
isn’t a single relevant Random.Utils call.

Maybe dividing the width is the cause for the   
randomness?

Discovered the roomWidth1 in

“roomsToSplit.Add(new Room(new Rectangle(xPosition1, yPosition1, roomWidth1, roomHeight1)));”

actually counts as the area the rooms can be spawned in on the x-axis (which makes sense because it’s the first room in roomsToSplit and the split code takes that room and divides based on the last width…)

Added the doors (and changed the color to red for debugging).   
For some reason,

The room to the right doesn’t have any doors, so that’s something   
I’ll have to figure out as well.

Improved the code a bit, but haven’t   
really gotten anywhere.

Added doors at places where it would -somewhat- makes sense. The thing I keep struggling with is that the rooms keep overlapping.

02/06/2020

05/06/2020