Worktest FatShark

Goal:

In this project the goal is creating a procedural level generator in Lua. Which is going to create multiple rooms in command prompt using ASCII. This can be exported as a text document which later can be run in a level creator that places out specific tiles based on the ASCII sign. The signs for the tiles are the following:
Door: “ +”
Floor: “ .”
Wall: “ #”

The process:

If we build up the scenario that I am an intern on Fatshark and this is a task I got, my process would look like the following:
At first I would research different types of level generation and find some algorithms suited for the task. Then I would present them for other colleagues working in this area and decide together if any of the algorithms is fitting for the task.
If none of them would be fitting, I would see if we could ball some ideas around that could change some of them to fit the project. If that wouldn't be doable, I would go back to searching for other algorithms and repeat the process.

Feedback from other people working on the project is something very important to me since then we can improve and shape the code together with ideas that are more fitting for the project. It also helps to keep everyone on the same page.

Since I am working on this task myself, I decided to simply pick the one I find the most fitting to creating a dungeon. In this case, binary space partitioning(BSP) was the most fitting for the task.

In BSP you create a large room, in this case a size of 50 * 50 square. Then you split the room either horizontally or vertically and repeat the process for a set amount of times. When it comes to the split, it does it randomly with a 50/50 chance however the code keeps track on how many times it has split in each way. If it has split vertically at least a set amount more times than horizontally, it will split horizontally and vice versa.

In the process, the code creates all the rooms first and adds them to a table that will contain all the rooms at the lowest level. Once that is done the code will loop through the walls of all the rooms, it will place a door if there isn't one placed on that entire wall. This check is to prevent a wall having multiple doors since some rooms share the same wall.

After that I export the code into a text file so it can be recreated in a level creator. Here is an example on a level created by the algorithm:

```
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
# . . . . . . . . . . # . . . . . . . . . # . . . . . # . . . . . # . . . . . . . # . . . . . . . . #
# . . . . . . . . . . # . . . . . . . . . # . . . . . # . . . . . # . . . . . . . # . . . . . . . . #
# . . . . . . . . . . # . . . . . . . . . # . . . . . # . . . . . # . . . . . . . # . . . . . . . . #
# . . . . . . . . . . + . . . . . . . . . + . . . . . # . . . . . + . . . . . . . + . . . . . . . . #
# . . . . . . . . . . # . . . . . . . . . # . . . . . # . . . . . # . . . . . . . # . . . . . . . . #
# . . . . . . . . . . # . . . . . . . . . # . . . . . # . . . . . # . . . . . . . # . . . . . . . . #
# # # # # # + # # # # # # # # # + # # # # # . . . . . + . . . . . # # # # + # # # # + # # # # # # # #
# . . . . . . . . . . . . . . . . . . . . # . . . . . # . . . . . # . . . . . . . . . . . . . . . . #
# . . . . . . . . . . . . . . . . . . . . # . . . . . # . . . . . # . . . . . . . . . . . . . . . . #
# . . . . . . . . . . . . . . . . . . . . + . . . . . # . . . . . + . . . . . . . . . . . . . . . . #
# . . . . . . . . . . . . . . . . . . . . # . . . . . # . . . . . # . . . . . . . . . . . . . . . . #
# . . . . . . . . . . . . . . . . . . . . # . . . . . # . . . . . # . . . . . . . . . . . . . . . . #
# # # # # # # # # # # + # # # # # # # # # # # # + # # # # # + # # # # # # # # # # # + # # # # # # # #
# . . . . . . . . . . . . . . . . . . . . # . . . . . . # . . . . . . # . . . . . . . . . . . . . . #
# . . . . . . . . . . . . . . . . . . . . # . . . . . . # . . . . . . # . . . . . . . . . . . . . . #
# . . . . . . . . . . . . . . . . . . . . + . . . . . . # . . . . . . # . . . . . . . . . . . . . . #
# . . . . . . . . . . . . . . . . . . . . # . . . . . . + . . . . . . + . . . . . . . . . . . . . . #
# # # # # # # # # # # + # # # # # + # # # # . . . . . . # . . . . . . # . . . . . . . . . . . . . . #
# . . . . . . . . . . . # . . . . . . . . # . . . . . . # . . . . . . # . . . . . . . . . . . . . . #
# . . . . . . . . . . . # . . . . . . . . # # # # + # # # # # # + # # # # # # # # # # + # # # # # # #
# . . . . . . . . . . . # . . . . . . . . # . . . . . . . . . . . # . . . . . . . . . . . . . . . . #
# . . . . . . . . . . . # . . . . . . . . # . . . . . . . . . . . # . . . . . . . . . . . . . . . . #
# . . . . . . . . . . . + . . . . . . . . + . . . . . . . . . . . # . . . . . . . . . . . . . . . . #
# . . . . . . . . . . . # . . . . . . . . # . . . . . . . . . . . + . . . . . . . . . . . . . . . . #
# . . . . . . . . . . . # . . . . . . . . # . . . . . . . . . . . # . . . . . . . . . . . . . . . . #
# . . . . . . . . . . . # . . . . . . . . # . . . . . . . . . . . # . . . . . . . . . . . . . . . . #
# # # # # # + # # # # # # # # # # + # # # # # # # # # + # # # # + # # # # # # # # # + # # # # + # # #
# . . . . . . . . . . . . . . . . . . . . # . . . . . . # . . . . . . . # . . . . . . # . . . . . . #
# . . . . . . . . . . . . . . . . . . . . # . . . . . . # . . . . . . . # . . . . . . # . . . . . . #
# . . . . . . . . . . . . . . . . . . . . # . . . . . . + . . . . . . . + . . . . . . + . . . . . . #
# . . . . . . . . . . . . . . . . . . . . # . . . . . . # . . . . . . . # . . . . . . # . . . . . . #
# . . . . . . . . . . . . . . . . . . . . + . . . . . . # # # # + # # # # . . . . . . # # # # + # # #
# . . . . . . . . . . . . . . . . . . . . # . . . . . . + . . . . . . . # . . . . . . # . . . . . . #
# . . . . . . . . . . . . . . . . . . . . # . . . . . . # . . . . . . . # . . . . . . # . . . . . . #
# . . . . . . . . . . . . . . . . . . . . # . . . . . . # . . . . . . . + . . . . . . # . . . . . . #
# # # # # + # # # # # + # # # # # # + # # # . . . . . . # . . . . . . . # # # # + # # # . . . . . . #
# . . . . . . . . # . . . . # . . . . . . # . . . . . . # . . . . . . . # . . . . . . # . . . . . . #
# . . . . . . . . # . . . . # . . . . . . # # # # + # # # # # # + # # # # . . . . . . # . . . . . . #
# . . . . . . . . # . . . . # . . . . . . # . . . . . . . . . . . . . . # . . . . . . # # # # + # # #
# . . . . . . . . + . . . . # . . . . . . # . . . . . . . . . . . . . . # . . . . . . # . . . . . . #
# . . . . . . . . # . . . . # . . . . . . + . . . . . . . . . . . . . . + . . . . . . # . . . . . . #
# . . . . . . . . # . . . . # . . . . . . # . . . . . . . . . . . . . . # . . . . . . # . . . . . . #
# # # # # + # # # # . . . . + . . . . . . # # # # + # # # + # # # # # # # . . . . . . + . . . . . . #
# . . . . . . . . # . . . . # . . . . . . # . . . . . . # . . . . . . . # . . . . . . # . . . . . . #
# . . . . . . . . # . . . . # . . . . . . # . . . . . . # . . . . . . . # . . . . . . # . . . . . . #
# . . . . . . . . # . . . . # . . . . . . # . . . . . . # . . . . . . . # . . . . . . # . . . . . . #
# . . . . . . . . + . . . . # . . . . . . + . . . . . . + . . . . . . . + . . . . . . # . . . . . . #
# . . . . . . . . # . . . . # . . . . . . # . . . . . . # . . . . . . . # . . . . . . # . . . . . . #
# . . . . . . . . # . . . . # . . . . . . # . . . . . . # . . . . . . . # . . . . . . # . . . . . . #
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
```

As stated earlier the doors are a “ +” sign, the floor is a “ .” sign and the walls are a “ #” sign.

If my colleagues and seniors give the level creator a pass, the next step would be recreating a level in the game project where it will set out specific tiles depending on the ASCII sign at the coordinates.

After that I would create a couple of levels in the game project and show them to my colleagues to see if they wanted to give some feedback. I would also throw out some ideas about potential enemies and hazards that could be placed out in some of the rooms.

A part of the task for me would be to see what my colleagues would like to see in the level. If they want the levels to be created during game time and have enemies and hazards placed out as a part of the level generation, or if a couple of levels should be created beforehand so enemies and hazards could be manually placed out in them.

The customization of the level creator would depend on the feedback my colleagues would give and ideas we would come up with together.

I would say that is the way I would handle this task at FatShark. I would begin with some research on how it could be done and show a list of different algorithms that fit the task. After that I would decide on one together with my colleagues and start implementing it in code.
Once the level creator is done, I would show some levels created in it and ask for feedback to see if something would need improvement and change some values if necessary.
The next step would be to see if I should add some more features to it or leave it at that and go to the next task.
If my colleagues would like some more features I would be up for taking in suggestions and coming up with ideas together in order to customise the algorithm in a way that would fit the game project.

I decided to write about the technical discussion part as well, since it had some interesting points for discussion. It also gave me some interesting ideas that I might implement in the future for the project.

Technical discussion:

“How well does the design take into account the requirements of a roguelike game?”
A simple way to decide a start point and still have a hint of randomness in it would be to find a room around the sides of the level and place a stairway in the middle of the room to indicate that this is where you came from. The player would be placed on the tile underneath the stairway.
When it comes to the end of the level I would find a random room on the opposite side of the level.
For example, if we look at the level I generated earlier. If the start is at the room to the far left and second lowest, then I would check for the sides furthest away from it. In this case it would be the top side and right side. Then I would randomly pick a room that is at one of those sides.

“Similarly, can we figure out how to ensure that the player will have to encounter a few challenges?”
A boss could easily be placed at the end room and we could lock the stairway to the next level while the boss is alive. If we wanted more bosses along the way, we could check for rooms in the direction from the end to the start and place out a couple of bosses semi-randomly. Where we make sure there is a certain amount of rooms between all the bosses.

My suggestion would be to have one boss for each level at the end and place out weaker enemies in groups throughout the level. My way to go would be to create groups of enemies with between 1 and (number of levels).
The code goes through all the rooms and checks by random if a group should be placed in that room.
I would set it so small rooms can only have one group, middle size rooms can have two groups and large size rooms can have three groups. The size check would be based on (roomWidth + roomHeight). The check will be done as many times as groups the room can hold. It would require testing, but I believe a 60% chance for a room to have a group would be a good value.
I would also set a minimum requirement of the amount of groups that has to be placed in each level. If the requirements wouldn't be met, I will use recursion and do the checks again, but skipping rooms that already have enemies.
When it comes to amounts and values it would require testing and I would ask my colleagues for feedback both in balancing and ideas of what I could add.

"How would we modify the project to make it generate different kinds of levels?"
When it comes to raising the difficulty for each level, the idea with groups of enemies comes in handy. The groups can get larger the higher the level number is.
I could also change the size of the entire level depending on the current level. This would result in even more rooms with potential enemies and I could add a check for even larger rooms so they could have even more groups of enemies. Another way to make the difficulty match the level would be making the enemies and bosses scale with the level.
I could also tweak how many times the algorithm uses recursion, this would create more rooms which could be useful if the size of the level changes.
When it comes to the difference between man-made and natural looking dungeons. This algorithm is designed to create a man-made dungeon, which is the inside of a building with different levels where you ascend or descend to stronger levels.
If my goal was to create a more natural looking dungeon, I would use another algorithm for that. BSP worked great for creating the inside of a building. However, it is not suited for making it look natural.

One of the algorithms I could use would be Cellular Automata(CA). In CA, I create a grid with tiles that are either solid or empty by random. After that it checks each tile on the grid and its neighbours. Depending on the amount of neighbours, the tile will be either solid or empty. The rules are:

- If a solid tile has less than two neighbours that are solid, it becomes empty
- If a solid tile has two or three neighbours that are solid, it will remain solid.
- If a solid tile has more than three neighbours, it becomes empty.
- If an empty tile has exactly three neighbours that are solid, it becomes solid.

This creates a level that simulates a natural cave. Just like in BSP and procedural generation algorithms overall, there are certain values that you can change which changes the structure of the level.
You can change the value for the chance of a tile to start as solid.
The list of rules comes from “the game of life” which is the base of CA. We can change the values in the rules to get a more fitting result. For example, we can say that:
“If a solid tile has two or five neighbours that are solid, it will remain solid.”.
This would create a level that had thicker walls and less room to walk in. This could work well in a game that is more about sneaking around.
Or we could say:
“If a solid tile has less than three neighbours that are solid, it becomes empty”.
This would create larger and more open areas to walk on. Which could be useful for a game that requires a lot of movement in combat.

When it comes to creating rivers of lava. An algorithm that could be used to create a lava river would be drunkard walk(DW). It creates a path which changes direction based on a set percentage chance. We could set the chance for changing direction to 30%. This would create a path that goes straight for the most part, but would change direction at some points. This could qualify for the randomness of a flowing river of lava. We could also set the amount of tiles the algorithm will go through, in order to create a longer or shorter path for the lava to flow or have the path keep on going until it hits the border. I could also add so the river splits into two parts based on a percentage chance.

This could work well in CA, since the slight randomness of DW would align with the solid tiles in CA. However, it wouldn’t really match with BSP. Since BSP splits the grid up in rooms that are strictly rectangular, a river of lava using a DW wouldn’t really fit there. It would be too random.

Something that could work would be placing out a start point and an end point inside a room and using AStar to create the path. Another thing I could do would be creating lava puddles or lakes randomly placed out in large enough rooms. So we would have a lava pool instead of a lava river.
A way I could create the lakes and pools would be picking a tile in a room as a centre position and using the flood fill algorithm to fill out a set amount of tiles with lava tiles with the walls as borders. The width and height of the puddle/lake would depend on the room it is in as a start. Then it would come to simply testing sizes and get feedback from colleagues in order to achieve a fitting size for the lava puddles and lakes.
The same would work if we wanted to use water, since they are both fluids I would treat them in the same way. Whatever would fit in the level the most.
It all comes down to customising the algorithm and combining it with other algorithms in order to fit the project in the end.