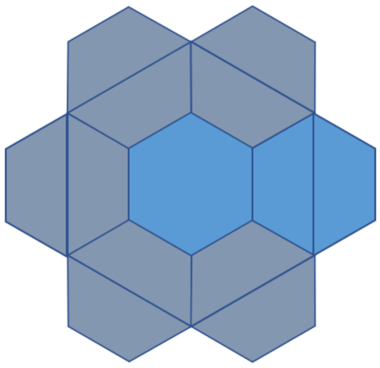
Procedural generation – the code

1 PROJECT IMPLEMENTATION

1.1 TILES

Tiles are encoded in a text file, with its constraints, its content (external ring and centre) and its weight.

Encoding: `[tag] [constraints], [external ring], [centre], [weight]`
Example: `River R 0 0 0 0 0, R 0 0 0 0 0, R, 5`
Visual translation:



1 – Example of a river tile

1.2 TERRAIN

The terrain is generated as a tile vector with a width and height. It is a partial grid that can be represented as such in text form:

```
                                           . .   . .   . .   M .   . M   . .   . .   . .   . .   . .
                                          . • . . • . . • . . M M M M . . • . . • . . • . . • . . R .
                                           . .   . .   . .   M .   . M   . .   . .   . .   . .   R .
                            . .   . .   . .   . .   . .   . .   H .   . .   . .   . .   R R   . R
                           . • . . • . . • . . • . . • . . • . H H H . • . . • . . • . R • R R • R
                            . .   . .   . .   . .   . .   . .   H H   . .   . .   . .   R .   R R
             . .   B B   . .   H .   . .   . .         . .   . .   H .   . .   . .   . R
            . • . B B B . R H H R . W W W W W .       . • . . R H H R R R R R R R R R R .
             . .   B B   R .   H R   . .   W .         . .   R .   . .   . .   . .   . .
    . .   . .   . .   R R   H H   R .   . W               . R   . .   . W
   . • . . • . . • . R • . . H H . R . . W .             . R . W W . . W .
    . .   . .   . .   R .   . .   . R   . W               R .   . W   . W
 . .   . .   M .   . R               R .   W .         . R   . .   H .   W .         . .   . .   . .
. • . . • . . M M . R .             . R . . W .       . R . . • . . H H . W .       . • . . • . . R .
 . .   . .   M .   R .               R .   W .         R .   . .   W .   . W         . .   . .   R .
    . .   . .   . R               . R   . .   . .   . R   . .   . W   . .   W .   . .   . .   . R
   . • . . • . . R .             . R R R R R R R R R • R . • . . W . . • . . W . . • . . R R R • R
    . .   . .   R .               . .   . .   . .   R R   . .   W .   . .   W .   . .   R .   R R
 H H   . .   . R   . .   . .   H .   M .   . .   B B   . .   . .   H .   . W   . .   . R   . .
H H H . • . . R . . • . . • . H H H . M M . • . B B B . W W . R H H R . W W . . R R R R . . • .
 H H   . .   . .   . .   . .   H H   M .   . .   B B   W .   R .   . R   . .   R .   . .   . .
    . .   . M   H H   H M   . .   H .   . .   . .   . W   . R   . M   R .   R R
   . • . M M . H H H H H . . • H H R . W W W W W . . W . . R . M M . . R R R W .
    . .   . M   H H   H M   . .   H R   . .   . W   . W   R .   . M   . .   W .
       . .   . .   . .               R .   . .   W .   . R   . .   . .   . W
      . • . . R . . • .             . R . . • . . W . . R . . • . . • . . W .
       . .   R .   . .               R .   . .   . W   R .   . .   . .   W .
          . R   M .   . .   . .   H R   . .   . .   . R   W .   . .   . W
         . R . . M M . • . . R H H R R R R R R R R R • R . W . . • . . W .
          . R   M .   . .   R H   H R   . .   . .   R R   . .   . .   W .
             R W   . .   . R   H .   R .   . W               . .   . W   . .   . .   . .
            R W R R R R R • R . H H . R . . W .             . H H . W . . • . . • . . • .
             R R   . .   R R   . .   . R   W .               . .   . W   . .   . .   . .
                                        R .   . .         . .   . .   R R   . .   . .   . .
                                       . R R R • .       . • . . • . R • R R R . . • . . R .
                                        . .   R R         . .   . .   R .   . R   . .   R .
                                                 R .   . .   H .   . R         R R   . R   . .
                                                . R R R R H H R R R R .       . • R R R . . • .
                                                 . .   . .   . .   . .         . .   . .   . .
```

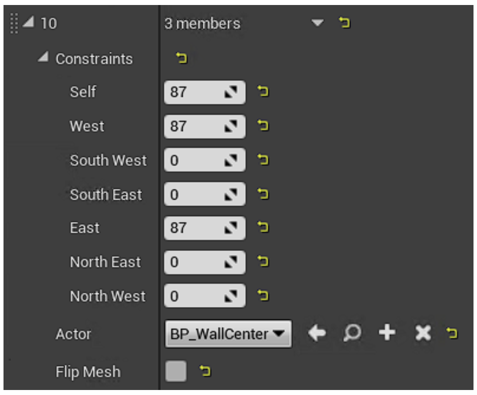
2 – A generated terrain (text form)

The `•` and `.` represent empty locations. The `•` are tile centres and `.` are external tile parts. The hexagonal shape used for the tiles can be recognized, especially at the edges.

Each letter represents an element: R for Road, W for Wall, M for Mine, B for Building, H for Hidden (the tile exist but nothing can be placed here).

1.3 Terrain conversion in Unreal Engine

The terrain is translated to spawn 3D actors in the world. A road spawns for each R, but differently depending on its neighbours (it needs to be linked to nearby roads). Same for the walls that are linked together to shape coherent structures.



10	3 members
Constraints	
Self	87
West	87
South West	0
South East	0
East	87
North East	0
North West	0
Actor	BP_WallCenter
Flip Mesh	☐

3 – Tiles translation with Unreal

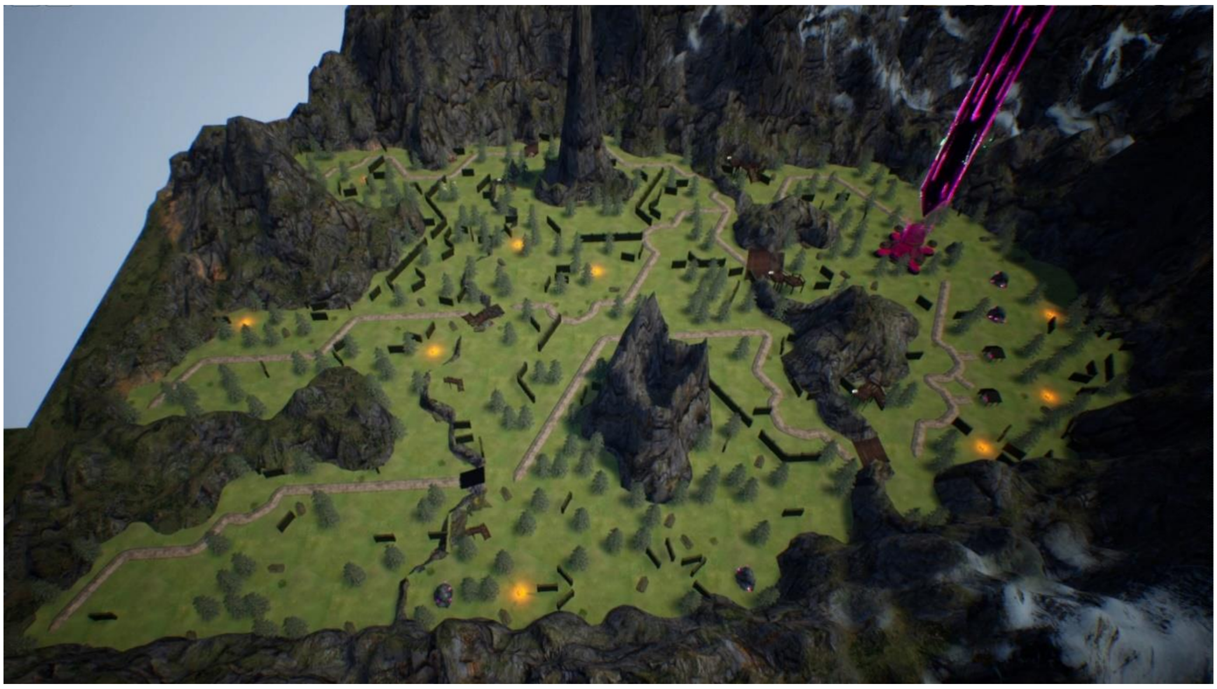
On the left, an example of a translation of a wall at the centre of a tile:

« If I am a wall (W = 87) and there is a wall west and east, then, I am going forward »

Possible rotations are also checked for those constraints. For instance, W south-west and north-east also translate to a forward wall.

There is also an option to check the symmetries of those constraints: we do not need to rewrite them for each variation.

Using it for all the tiles on the terrain gives us a full map with all environmental actors well placed!



4 – In-game terrain

2 THE CODE

Let's dive in! A lot is happening for coordinates management for the hexagonal tiles, or for the grid management that I will not talk about in here. The code is available if you want to have a look. I will mainly talk about parts that seem important and interesting to develop here.

So here we go!

2.1 DESERIALIZATION

The tiles used for the generation are deserialized for the file `Tuiles.txt`. Then they are stored in `GenerationData`, a class that contains a dictionary of `WeightedTile` (defined in that class) representing the tiles.

At that point are defined our `Element<T=char>` – modified `char`. They handle the interactions between the tiles. For instance, a `Bridge (P)` can be placed on a `Road (R)`, but not on `Void (.)`. Those rules are defined in `Config`, a pImpl (privately Implemented) struct in `Element<T=char>`. I first wanted to make it a static variable available to all `Element<T=char>`, but I could not initialise it in the header file. I then used a *Privately Implemented Singleton*, which works the same for my use case, and is giving more control.

2.2 PARTIAL GRID

The grid used is a partial grid, with holes (used for mountains in our game, or fixed terrain structures). It is represented by `PartialGrid<Grid>`, which specifies a standard grid. It accepts any grid type as long as it has tiles and a neighbour's relationship between them.

Note: A partial grid is also a grid that matches those criteria.

2.3 STRUCT PARAMETERS

The procedural generation has a lot of parameters. The function `buildPath` for instance, one of the main functions used, it was defined as:

```
void buildPath(tile_ptr start,
               tile_ptr goal,
               tag_t tagStart,
               tag_t tagGoal,
               tag_t tagPath,
               element_t linkingElement,
               std::function<bool(tile_ptr)> startCondition = trueCondition,
               std::function<bool(tile_ptr)> stopCondition = falseCondition,
               std::function<bool(tile_ptr)> goalCondition = trueCondition)
```

Most of those parameters were then given to sub-functions called by `buildPath`. So instead, those parameters have been regrouped in a struct `PathData`. The signature `buildPath` then became:

```
void buildPath(PathData pathData,
               std::function<bool(tile_ptr)> startCondition = trueCondition,
               std::function<bool(tile_ptr)> goalCondition = trueCondition);
```

2.4 STD::FUNCTIONAL

I left the possibility for the user to use its own predicates for some part of the procedural generation. For instance, in `buildPath`, predicates are used to define conditions a tile must meet to be the start or the end of a path. Another predicate handles a stop condition for the algorithm (if a path intersects with another for instance).

2.5 VECTORMATH

I created a little library in `VectorMath.h` which contains iterator operations. It allows me to directly use `vector`, instead of using `begin()` and `end()` each time I need an bulk operation. It especially allows me to use `vector` coming right out of another function without having to store the result and reuse it after. I do not consider the different allocators that could be used, those functions are more for convenience than a release for everyone to use.

3 INITIAL DESIGN MODIFICATIONS

I initially planned to create custom tiles with my 3D assets directly included in them. That would have prevented the translation from text to Unreal Engine, with all assets perfectly placed. Instead, I used a conversion from char to 3D assets (see 1.3). That would have saved me some time in the long run, but I had to start integrating the assets before the algorithm was ready for them.

The grid was passed as a reference parameter. At each sub-function call, I had to pass the grid as a parameter. Instead, I used a pointer to the grid, a member of `Generation` (the class handling the procedural generation). I took the generation rules away from the `Generation` class to another one: `GenerationData` which contains the available tiles. This class can be used by any `Generation` class (if the same elements are required, to apply them on multiple grids, for instance).

4 POSSIBLE IMPROVEMENTS

Some classes are not templated while they could (and should be). I did not need it as I only used one type of tile, but that could be useful in the future.

Some functions might not have the best name and are dependent on our specific use case. That is mainly due to a lack of time to work more on this part of the project.

5 ILLUSTRATIONS