Overview of procedural 2D cave generation methods

Eduard Zalyaev
DS2-2016
Innopolis University
Innopolis, Russia
e.zalyaev@innopolis.ru

***Abstract*—This document examines three algorithms of that can be successfully used for procedural cave generation - Perlin noise, Celluar Automata and Diamond Square. All three were applied to a 2D grid made up of 4 textures: water, grass, rock and empty space that is used for cavity indication in the caves. All three algorithms were tested with same height ranges for all types of tiles on the grid. Quantitative evaluation is provided.**

I. Introduction

For a colony simulator developed as a course project we had to come up with a method of map generation. One of the most important aspects of the games in this genre is resource management. The simplest type of such resources can be some kind of ore located in the rock formations. In order to gather and manage these ores they have to be properly generated on the map. Location of an ore can be thought of as cavity in the in the rock formation which can then be replaced with an actual ore. So technically we want to generate some sort of a cave. Since laying out the map by hand is quite a tedious task we can utilize algorithms for procedural heightmap generation.

II. Height to tilemap mapping

Heightmap generated by all examined algorithms consists of a 2D array where each cell's location corresponds to location on the map and value in the cell corresponds to it's height. In the case of developed environment "height" values are discrete and correspond to the class of the tile that will be rendered as a single sprite. So a height of a continuous range [0,1] has to be multiplied by the amount of possible types and floored to the next integer value.

```
float tileHeight = map[x,y];
int tileClass = math.ceil(tileHeight*numClasses);
if tileClass==1 then
    return sprite1;
end
if tileClass==2 then
    return sprite2;
end
...
```

Algorithm 1: Tile sprite selection

III. Algorithms

A. Perlin noise

Perlin noise [1] is a classic example of a continuous noise generation algorithm. 2D Perlin noise is a function that maps a point (x, y) to range $[0, 1]$. Firstly the algorithm generates random unit gradient vectors at integergrid points. It then finds four closest gradients for a given point on the grid and interpolates between them to get the noise value.

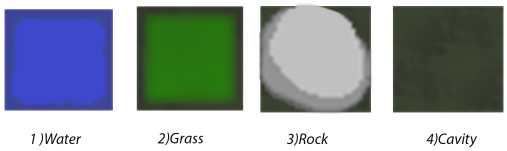

Fig. 1. Examples of sprites used for different height values

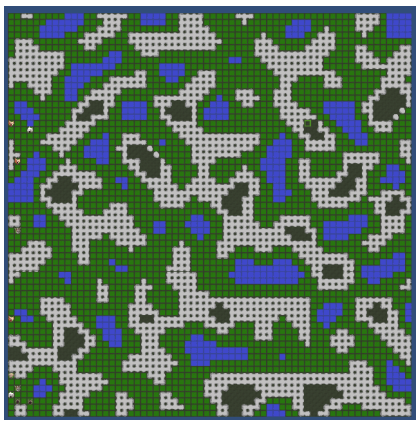

Fig. 2. Terrain generated with Perlin noise

B. Celluar Automation

Celluar automation is a model that can produce random clusters of points. Algorithm starts off with randomly selected points on the that are considered to be "alive" and hold binary value of 1. At each iteration all "alive" neighbours within Manhattan distance of 1 are counted and if this amount is less than a certain threshold ([1,4]) it takes the value of 0. The algorithm stops after a predefined number of iterations. In our case we populate rock formations with cavities. At each iteration all cell's neighbours within the formation are counted and in case if threshold of 3 is covered cell will turn into cavity or will be left as such otherwise it turns into rock. The cells on the rock formation boundaries or outside them are not

considered as ”alive” and cannot be set as such meaning no cavity can generate an entrance to the formation. This reason for doing so is the fact that after certain amount of iterations the formation will completely erode. Water, grass and rock regions initially generated with Perlin noise with no cavity regions. Algorithm is then applied to the generated terrain.

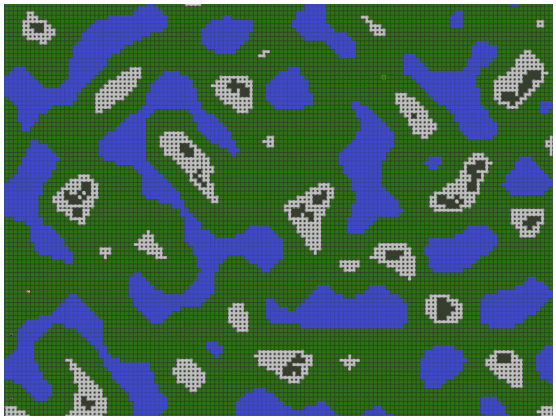

Fig. 3. Cavities generated with Celluar Automation

C. Diamond square algorithm

This [3] algorithm is generally used to produce realistic terrain heightmaps. It consists of two operations:

- **Square step**: set value of the square midpoint to average of it's corners
- **Diamond step**; set value of the diamond corners with an average value of it's center and two closest square corners. A random displacement value is also added to the average in order to increase noise.

Fig. 4. Algorithm visualization

Corner values of the initial matrix have to be randomly set first. Diamond and square steps form a new set of four smaller squares within the matrix. The process repeats on smaller squares until the whole matrix is initialized.

IV. Evaluation

A. Metrics

- **Average speed**(Avg. Speed): map generation speed
- **Cavity to formation count ratio** (CtFC): ratio of closed cavity and rock formation amounts. If smaller than 1 then most formations have only one cave or don't have them at all.
- **Cavity to formation area ratio** (CtFA): shows how hollow rock formations are.

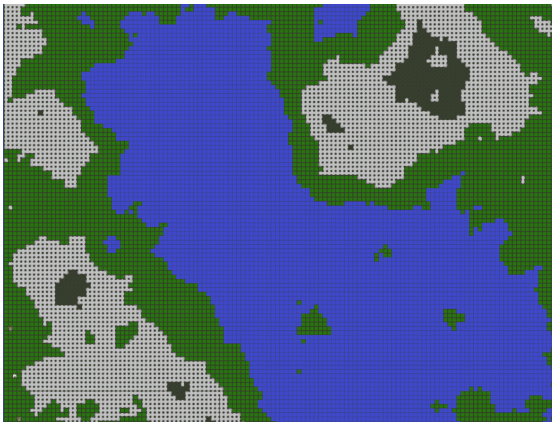

Fig. 5. Terrain generated by Diamond Square algorithm

B. Values

Setup:128x128 grid.For B 1 epoch is chosen because multiple tests showed significant change in metric values.

	A	B	C
Avg. Speed	2.3ms	8.6ms	5.5ms
CtFC	0.72	0.61	[0.0,0.68]
CtFA	0.1217	0.1254	[0,1.96]

*CtFC and CtFA values remained constant during all tests

V. Conclusions

Quantitative tests showed that Celluar Automation and Perlin noise models show little to no variance in cave formation characteristics and overall look repetative in some way. However visually CA produces more variance in cavity shapes in comparison with elliptical blobs of Perlin noise. On the other hand Diamond Square algorithm is able to produce large biomes with lakes and small or big caves inside rock formations. It can also be used purely for terrain and cave generation in combination with CA which might sparsely populate rock masses with resources.

References

[1] K. Perlin Improving noise In *ACM transactions on graphics (TOG), 2002*
[2] G. Martin Mathematical Games The fantastic combinations of John Conway's new solitaire game ”life” In *Scientific American. 223*
[3] F. Alain, F. Don, C. Loren Computer rendering of stochastic models In *Communications of the ACM. 25*

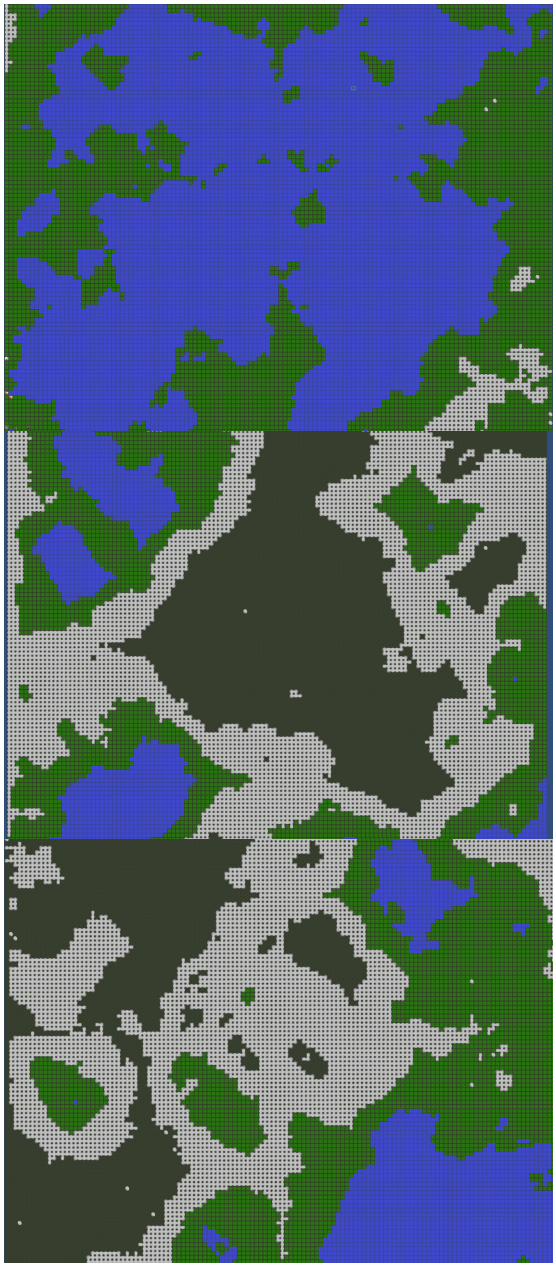

Fig. 6. Additional Diamond Square algorithm examples