PROG 37198 Game Engineering

Lab 2 – Random Terrain Generation

Introduction

The game we will be building throughout the semester is an artillery game similar to classic like scorched earth and worms. The games are partially defined by the terrain they are played on, which is usually a 2D "slice" of the world that has hills, mountains, valleys, etc… While these terrains can be hand generated, a large part of the tactics and strategies of these artillery games lies in adapting to a wide variety of terrain features. This can be achieved by using procedurally content generation (PCG) to create the terrain. PCG, roughly speaking, is the use of intelligently guided randomness to create something that adheres to a set of basic rules while having a wide variety of outcomes. A big part of PCG is the concept of intelligently guided randomness, which often means we want to use random "noise" as opposed to truly random values.

Required Readings

http://www.redblobgames.com/maps/terrain-from-noise/ This is one possible use of noise and has some interactive demos you can use. Note that this generates 3D terrain, where as the lab will only require 2D terrain.

http://www.redblobgames.com/articles/noise/introduction.html Once you have read the first article to see the power of noise, come back to this one for an explanation of how it all works and some practical examples of noise being used to generate 2D terrain.

https://bitesofcode.wordpress.com/2016/12/23/landscape-generation-using-midpoint-displacement/ The midpoint displacement algorithm is another classic terrain generation algorithm. While it can be adapted to generate 3D terrain (usually via an adaption of midpoint displacement called diamond-square), midpoint displacement is quite suitable to generating 2D terrain as well.

https://gamedevelopment.tutsplus.com/tutorials/generate-random-cave-levels-using-cellular-automata--gamedev-9664 Cellular automata are a gigantic family of procedural content generation tools that have applications in almost all areas of computing. One of those uses is in gaming – cellular automata can be used to generate terrain. It is particularly suited to generating cave levels.

https://www.cs.helsinki.fi/group/goa/mallinnus/lines/bresenh.html Bresenham's line drawing algorithm is a very useful algorithm for determining which pixels are part of a "line" that is defined by two end-points in whole pixel values. When drawing a map that consists of multiple connected points you can use the algorithm to figure out how which pixels should be "terrain" and which should be "sky".

http://gpolo.awardspace.info Cool demo of the line algorithm. Not in English but easy enough to use.

Optional Readings

http://www.gameprogrammer.com/fractal.html#midpoint While I highly recommend the entire article, the main focus here is the midpoint displacement algorithm being used in 1 dimension to generate 2D terrain.

http://www.somethinghitme.com/2013/11/11/simple-2d-terrain-with-midpoint-displacement/ Another midpoint displacement algorithm explanation, with a demo and some code.

The Lab

You are going to create a simple map generator program. This program will allow you to generate maps that are suitable for our artillery game.

Prelab Questions

For purposes of this questions, “Good” terrain is subjective but should generally follow the maxim of “looks like it could realistically appear in nature”.

1. Using a pen and paper, generate 8 different terrains using the following algorithm:
    a. Start with a 6x6 square;
    b. Randomly generate a number **n** between 1 and 6 using a 6 sided die. Place a point at x = 0, y = **n**;
    c. Continue to generate numbers between 1 and 6 and placing them on the map, increasing x after each roll until you have six points on the map;
    d. Connect the lines to create the final terrain.

   Label the terrains as A-H.
   Do the terrains look good? Why or why not?
   Draw terrains A and B on a separate piece of paper with a horizontal exaggeration of 10. Do the terrains look good? Why or why not?
2. Repeat the above terrain generation using 2 dice instead of 1. You should generate a single 12x12 terrain, with **n** being calculated by taking the sum of the values from the two dice. You will need to generate 12 points. Draw terrains C and D with a vertical exaggeration of 2 and horizontal exaggeration of 2. Which terrains looks good? Why or why not?
3. Repeat the above experiment using 2 dice again. This time you should generate a single 36x36 terrain, where **n** is being calculated by taking the product of the two dice. You will need to generate 36 points. Draw terrains E and F with a vertical exaggeration of 6 and a horizontal exaggeration of 6. Which terrains look good? Why or why not?
4. Repeat the above terrain generation using the following modification:
    a. Start with an 18x18 square;
    b. Place the first point at x = 0, y = **9**;
    c. Continue to generate the remaining points by randomly generate a number **n** and **m** between 1 and 6. If **n** is even, place the next point at y=**[previous y] + m**. If **n** is odd, place the next point at y=**[previous y] - m**. You will need to generate 17 points (the first point is free).

   Draw terrains G and H with a vertical exaggeration of 3 and a horizontal exaggeration of 3. Which terrains look good? Why or why not?
5. Bring the hard copies of all of your maps to the next class.

Lab Programming

Working with a partner (groups of 2, 3, or 4 if necessary) build a level generator tool. This tool should generate .png files or .bmp files of size 600x400. You do not necessarily have to generate a y-value for each possible x-value, since you can define the terrain as a series of lines and use Bresenham's line algorithm to figure out the intervening pixels. Your map should color all "sky" or "clear" pixels as blue (RGB 0,0,255) and the "ground" or "terrain" pixels as green (RGB 0,255,0). This tool should allow you to create a level using EACH of the following level generation algorithms:

1. Any one of the dice rolling algorithms described above;
2. Midpoint displacement algorithm;
3. Using "noise" of your choice with 2 different modes (either colors, or trig); and
4. Using cellular automata (cave level) with 2 different modes.

Your program should run from the command line and use a command-line parsing algorithm like getopt to control the output. Each run of the program should either output a usage message or, if the right arguments are used, create a .png file or a .bmp file in the local working directory that contains an image of the generated map. You should have arguments that allow you to customize the behaviour of each algorithm at run time:

Flag		Description
General Options		
-h		Print the help message
-g	*algo*	Required – selects generation algorithm n=1-4
-s	*seed*	Optional – use a fixed seed *seed* instead of using system time as the seed
-f	*filename*	Required – filename output Bonus – prompt for overwrite if file exists already
-F	*filename*	Bonus – automatically overwrites if file exists already
Dice rolling algorithm Options (algo = 1)		
-d	*sides*	Optional – number of sides on the dice used, if not set assume 6
-r	*rolls*	Required – number of points to generate horizontally (intervening points are interpolated using Bresenham's line algorithm)
Midpoint Displacement Options (algo = 2)		
-n	*rough*	Required – Roughness value
-C		Optional – If not set, cap heights to the height of the world. If set, rescale the generated values such that they fit within the world's bounds
Noise Options (algo = 3)		
-m	*mode*	Required – A string for either noise colors ("pink", "brown", "red", etc…) or trig functions ("sin", "cos"). Must have at least two modes.
Cellular Automata Options (algo = 4)		
-a	*ruleset*	Optional – A integer that corresponds to the ruleset used by the cellular automata. Defaults to 1. Must have at least 2 rulesets.
-e	*gens*	Required – Number of generations of the algorithm that should be run for
-p	*percent*	Required – Percentage chance that each tile of the world should initially be set as "terrain"

Bonus Code!

Looking for a bigger challenge? Try expanding one or more of the above algorithms to have additional command line arguments. For example you could use arguments to allow the midpoint displacement algorithm to specify the starting values for the left and right-most points.

Lab Questions

1. Take a look at the maps generated with the midpoint displacement algorithm. Do those maps have any “hollow” areas? Do those maps have any “overhangs”? What I mean by those terms is a situation in which a point in the map that has no terrain is directly below a point in the map that does have terrain.
2. Take a look at the maps generated by cellular automata. Do those maps have any “hollow” areas? Do those maps have any “overhands”? (See Q1 for a definition of those terms).
3. When generating terrain with cellular automata, how did you handle points on the edges which have fewer than 8 neighbors?
4. Which algorithm is most suited to generating a level that looks like a mountain range? Explain your answer.
5. Which algorithm is most suited to generating a level that looks like the interior of a cave system (i.e. is deep underground)? Explain your answer.
6. What algorithm would be best suited to create terrain that has mountains and caves? Provide some pseudo code for your algorithm. Your answer can combine multiple algorithms.
7. For each of the algorithms, which ones would be best suited to creating terrain in batches (i.e. 600 pixels at a time) that can be tiled seamlessly (i.e. can be placed next to another generated piece of terrain and have all the edges appear to line up). Explain your answer for each of the four algorithms.

Marking Scheme

Algorithms

/10 Die rolling (arguments, processing, line algorithm, output)
/10 Midpoint (arguments, processing, roughness, output)
/10 Noise (arguments, processing, 2 modes, output)
/10 Cellular Automata (arguments, processing, 2 modes, output)

/40

Program

/5 Process command line arguments
/2 Uses existing parsing library instead of inventing your own

/7

Submission Requirements

/5 Submitted the source code/project as a zip with all unnecessary files removed
/5 Code is well documented
/5 Submitted project includes a .bat file I can use that generates 10 maps that show off features

/15

Bonus

/? Impress me!

Total

/62