Adaptive Minecraft Village Generation

Christian Oliveros[1], Pedro Rodríguez[2], Luis Díaz[3], Carlos Sivira[4], and Kevin Mena[5]
Universidad Simón Bolívar, Caracas, Venezuela

***Abstract*— The procedural generation of believable human settlements is an interesting and complicated branch of procedural generation of content for games. Minecraft[1] is a Lego-like 3D computer game that provides a well-defined and constrained environment in which to test procedural content generation techniques. Our first approach and entry to the Generative Design in Minecraft Competition[2] tries to follow Minecraft medieval-like villages with minimal terrain modification. It is based on the work of Boelter T. and Tadeu C.[3] which uses a modified A* Algorithm for Road System Generation and a modified Delaunay Triangulation for Village Boundary Generation. For Building Placement, we use an approach based on BFS to place pre-generated buildings from the center of the village to its boundaries.**

I. INTRODUCTION

This is our first entry to the Generative Design in Minecraft Competition[2] (GDMC). Our main goal is to create a framework on to which base future submissions to the competition. We decided to use C# as it is the language we most use, it is relatively cross-platform, it provides access to language level thread parallelism (see Python's GIL[4]), and because Python 2.7 is currently at its end of life.

Our approach to village generation is to minimize the terraforming required to build it and to be close to water, as humans require water to survive and most settlements are found near a water source.

Our Solution consists of the steps shown below:

1) Analyze Terrain for acceptable places where to build and its topology.
2) Select where to put Villages and their sizes.
3) Generate a Road and Bridge System to connect all the villages.
4) Place Buildings inside villages.

II. TERRAIN ANALYSIS

We start by analyzing several aspects of the terrain to generate maps, which include: height without trees (See Figure 1), lava or water; ground-level water; trunks and leaves (treemap); and lava. From the heightmap, we generate a delta map which shows the height relation of each ground block to its eight closest neighbors (See Figure 2). See Figure 4 for some of the maps mixed together.

^{1,2,3,4,5}We are all undergraduate students from Universidad Simón Bolívar, Caracas, Venezuela. We are also members of DELU, the university game development group.
[1]Email: christianol_01@hotmail.com
[2]Email: pedro18fe@gmail.com
[3]Email: ldiazn98@gmail.com
[4]Email: carlos.csivira@gmail.com
[5]Email: kevmena35@gmail.com

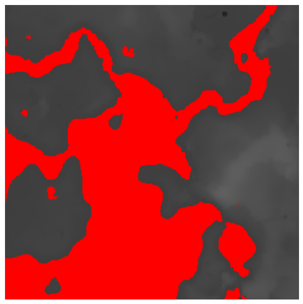

Fig. 1: Example of Height Map. White indicates highest terrain, black lowest terrain, and red indicates invalid terrain.

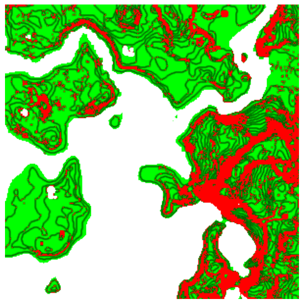

Fig. 2: Example of Delta Map, green indicates acceptable delta with the brightest being the smallest delta. Red indicates an unacceptable delta.

From all the generated maps we then generate an acceptable terrain for construction map, or acceptable map in short (See Figure 3), in which we select blocks that are not trees, in water, in lava or that do not have an excessive slope (this comes from comparing the delta map value of each block against a threshold).

As well, we analyze the water and separate it into bodies of water that could sustain a village. To select which bodies could sustain life, we compare the number of ground level-water blocks to a predefined threshold which we fine-tune.

III. VILLAGE PLACEMENT

For village placement, our algorithm tries to place village centers near water bodies to simulate the need for water by human settlements. This is done by placing random points in the map, that are then moved using A* to positions near the biggest and closest water body while avoiding other villages.

To select which blocks belong to the village, we use the algorithm proposed by Boelter T. and Tadeu C.[3]. Which goes as follows: *"Given a node N and a maximum node count C. We create a circle of radius R centered at N. Then,*

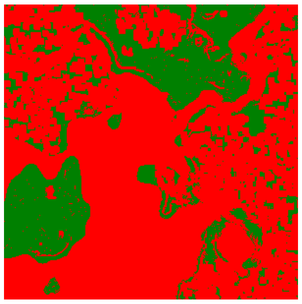
Fig. 3: Example of Acceptable Map. Green indicates acceptable terrain and red unacceptable one.

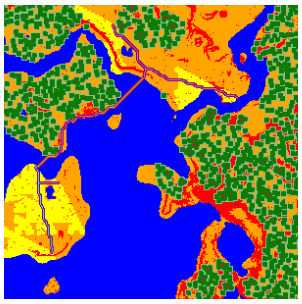
Fig. 4: Example of Road Map, TreeMap, Village Map, and Water Map mixed together. The Villages are marked Yellow and the valid space for villages is orange. The roads are purple with its center brown. Trees are green with their trunk brown.

we select the acceptable nodes that are the farthest in the X and Y axis from N and create new circles of radius R centered on those nodes. Now we count how many acceptable nodes exists that are contained in three or more circles. If the number of counted nodes is less than C, we start the loop again, creating new circles from our last nodes, and counting the number of acceptable nodes." Our main modification to this solution was to use circles under the Chebyshev Distance, which are squared. This was done because this distance generates bigger villages for the blocky world of Minecraft, where we can fit more easily buildings that are hard to rotate compared to a continuous world.

IV. ROAD SYSTEM GENERATION

Once all the villages are placed, we generate the Road System by using A*. The first road is placed between the center of two random villages, then we generate roads from the rest of the village centers to the closest road they can find, as proposed by Boelter T. and Tadeu C.[3], to get a system that mimics real-life road development. Our modification consists of penalizing crossing water, height changes, and leaf-cutting.

V. BUILDING PLACEMENT

The first step of our building placement algorithm is sorting every point of the villages by the value of the deltaMap at that point, going from the gentlest slopes to the steepest ones. Next, we proceed to test four rectangles rotated around the point. The criteria for the Rectangle test is how far it is from another house's Rectangle and how much terrain will be modified if we build a house inside of it, this includes, placing blocks for the house foundation or removing them to flatten the terrain. If the Rectangle is usable, a house type is chosen depending on the distance to the village seed and placed. Finally, we generate a road from the door to the nearest main road. See example of results in Figure 5.

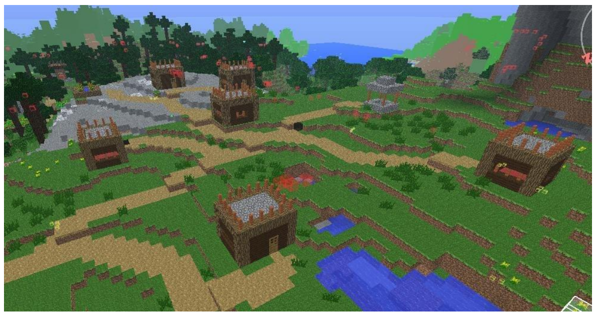
Fig. 5: Example of Building Generation.

VI. CONCLUSION AND FUTURE WORK

We are satisfied with our work done for our first entry to the Generative Design in Minecraft Competition. We created a C# framework that lets us export and manipulate blocks while taking advantage of C#'s Task System for parallelism. As well, we accomplished our goal of generating villages that try to fulfill the need of the would-be humans that live in it. Our code is available at Github[5].

For future work and entry to the GDMC, we would like to add the exploration of aquifers for water-well construction and food industry generation based on the village size. Furthermore, we would like to explore the use of L-System and Wave Function Collapse (the use of a SAT-solver for Procedural Content Generation, See explanation by Michel É.[6] and by Karth I. and Smith A.[7]) for building placement and online/offline building design.

REFERENCES

[1] Mojang Synergies AB. (2020, jun) Minecraft. [Online]. Available: https://www.minecraft.net/

[2] The GDMC Organizing Committee. (2020, jun) Generative design in minecraft competition. [Online]. Available: http://gendesignmc.engineering.nyu.edu/

[3] T. Boelter Mizdal and C. Tadeu Pozzer, "Procedural content generation of villages and road system on arbitrary terrains," *2018 17th Brazilian Symposium on Computer Games and Digital Entertainment (SBGAMES)*, 2018, Available at http://www.sbgames.org/sbgames2018/files/papers/ComputacaoFull/188241.pdf.

[4] T. Wouters. (2017, aug) Global interpreter lock. [Online]. Available: https://wiki.python.org/moin/GlobalInterpreterLock

[5] C. Oliveros, P. Rodríguez, L. Díaz, C. Sivira, and K. Mena. (2020, jun) Delu: Entry for the generative design in minecraft competition. [Online]. Available: https://github.com/Grupo-DELU/DELU-MINECRAFT

[6] Élie Michel. (2020, may) Twitter thread explaining wave function collapse. [Online]. Available: https://twitter.com/exppad/status/1267045322116734977

[7] I. Karth and A. M. Smith, "Wavefunctioncollapse is constraint solving in the wild," in *In Proceedings of the Eighth Workshop on Procedural Content Generation (PCG2017)*, 2017, Available at https://adamsmith.as/papers/wfc_is_constraint_solving_in_the_wild.pdf.