The Free and Open Independent Minecraft Map Maker's Magazine

MAPMAG

ISSUE 11 - WORLDS

Totality
Massive map making feature by ColdFusion

Big Projects Faster
Modular city designs with Weston Ahern

Terraforming
LazerRock25 takes gardening to new heights

Building by Hand
Keith Ross with all the goss.

Generative Design
The competition

Procedural Generation
Make more complex spaces with these tips and tricks

Emerald Chambers
Dieuwt's Roguelike creation

THE WORLD BUILDING ISSUE

Good things come to those who wait, and you have waited a very long time for this issue of Map Mag. This issue deals with the complexity of world building. I hope there is something of interest to everyone who shares a passion for making a complete environment for players to enjoy.

Totality is a complete science fiction Java map that invites you, the player, to explore its winding narrative in a series of robotic themed complete-the-monument challenges. The author Cold Fusion takes us on a similar journey through the process of developing the map and shares learnings from the process in the lead article this issue.

Keith Ross explains the value of hand-crafting your Minecraft creations in a short opinion article. Let us know your thoughts on this topic on the @MapMakingMag Twitter feed.

Weston Ahern explores the craft of building parkour themed city skylines using a variety of tools and methods. In this exposé you will also see the latest updates from Weston's massive forthcoming map *Red Edge*.

Dieuwt takes us far below the ground for the turn-based dungeon crawler *Emerald Chambers*. Strategic thinking and dynamic environments offer a unique twist on world-building!

In-game terraforming is explored in LazerRock25's article rounding out the tools an techniques section of this issue.

As in previous issues you will find lots to explore in this issue with additional articles on topics like procedural generation of content.

Map Making Magazine is a community publication and is open for you to share your stories with the world. We hope to hear from you next issue with your contribution of article or art.

See you around, wherever you are in your world building!

@abrightmoore (Editor)

SUBMISSION GUIDELINES

We are interested in what YOU have to say. Content you make for **Map**[Mag] can be sent to:

mapmakingmag@gmail.com.

The best letters, articles, art, and other work may be selected for inclusion in **Map**[Mag] editions or on affiliate websites and other communication channels. Because **Map**[Mag] is made by the community for the community, **Map**[Mag] is free for readers and we don't pay you for anything. You give us permission to include your work in the magazine.

Any content you submit must be your own work, or work that you have the right to submit. By sending us your work you agree that we may edit it for readability or make changes we think are necessary for the magazine. If we decide to include your work you acknowledge that you have granted us the right to publish your work in **Map**[Mag] and you understand that your work may be quoted or discussed on the internet by anyone in the world without limitation.

All other rights to your work remain with you. You own your work. We are allowed to use it for **Map**[Mag]. It is that simple.

We will credit you by real name, game name, social media account, or another method that you prefer and that we mutually agree. We will not share your email address without your express permission. If you do not tell us how to credit you for your work then you may not be published in **Map**[Mag].

If we refer to you or your work in **Map**[Mag] you acknowledge that we do so in good will and our intention is not to damage or harm.

DISPUTES

Writing about what you enjoy and hearing from other people with similar interests can be great fun. When people are excited about what they are doing sometimes things can get a little heated in a large community. If you have any concerns over what **Map**[Mag] is doing or how we are doing it then please contact us describing your concern. This will allow us to understand how we can do better. We can be reached at **mapmakingmag@gmail.com**.

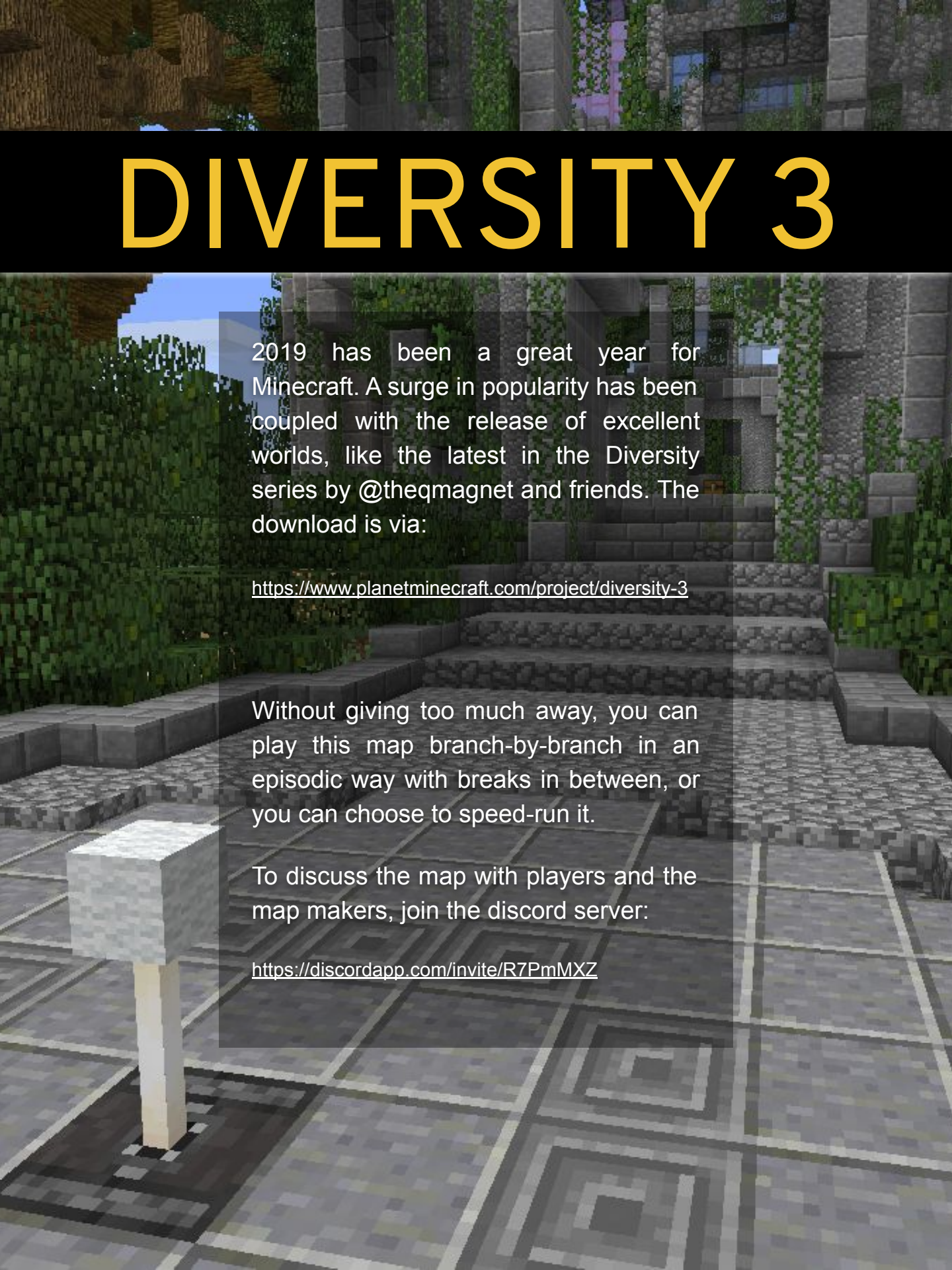
By reading this magazine you agree that the Contributors, Production Team, and anyone associated with this activity are not liable for any damages to the fullest extent permitted under law. You agree that any dispute arising from this publication is governed by the laws of New South Wales, Australia.

The Free Independent Minecraft Map Maker's Magazine

MAPMAG

ISSUES 1-10

http://www.testfordev.com/MapMag



DIVERSITY 3

2019 has been a great year for Minecraft. A surge in popularity has been coupled with the release of excellent worlds, like the latest in the Diversity series by @theqmagnet and friends. The download is via:

https://www.planetminecraft.com/project/diversity-3

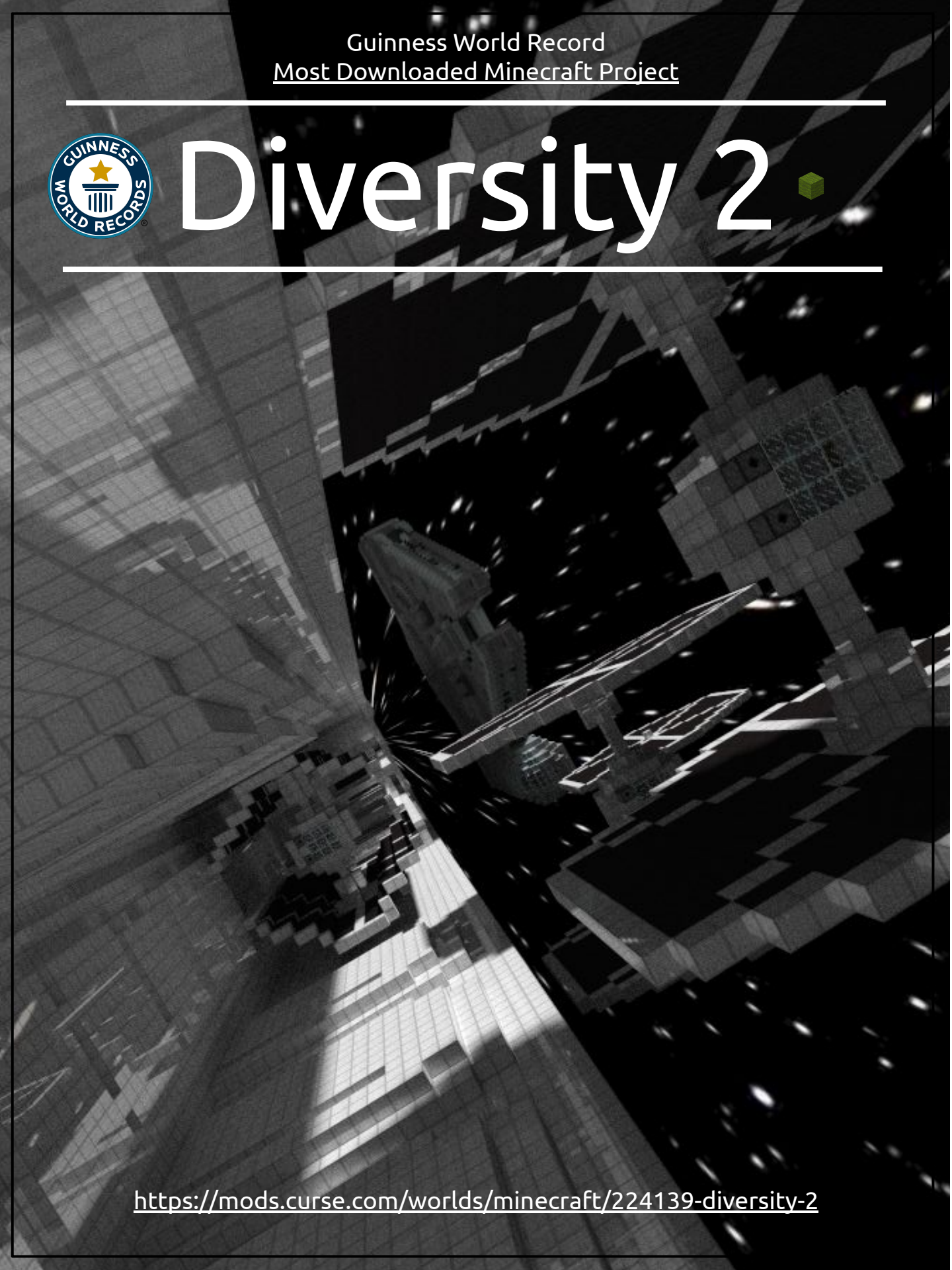
Without giving too much away, you can play this map branch-by-branch in an episodic way with breaks in between, or you can choose to speed-run it.

To discuss the map with players and the map makers, join the discord server:

https://discordapp.com/invite/R7PmMXZ

Guinness World Record
Most Downloaded Minecraft Project

Diversity 2



https://mods.curse.com/worlds/minecraft/224139-diversity-2

Stockholm skyline

Mojang Creator Conference 2019

In May Mojang invited your intrepid editor and a selection of map makers on the Marketplace Partner program to join with a number of YouTube entertainers to come to the home of Minecraft in Sweden and learn about the inner workings of the product teams.

The event was mainly about business planning though there were many opportunities to socialise and discuss projects and collaborations.

Special attention was given to the upcoming Minecraft related game releases.

The developers, community team members, and management were on hand with stories and exciting news about their work. Too numerous to name, all were extremely generous with their time and viewpoints.

You can read all about my experiences in Sweden on this Twitter thread here!

Mojang's headquarters

MAPCADEMY

Get it on the Marketplace!
(Click here)

Congratulations! You have been successful in your application to the Minecraft map maker academy. Your first year studies will include:

- Dungeons
- Dropper
- Labyrinth
- Mob Spawning
- *Death!*

Mini Melbourne, by @EduElfie and friends

TOTALITY

a post-mortem by Cold Fusion

About Totality

Totality is an open-world sci-fi adventure map by Cold Fusion. Crash-landed on Earth, your only hope of escape is to locate four ancient batteries to power your rocket. You must build tactically with an infinite supply of blocks to overcome hordes of robotic foes and find your way home. If you haven't played the map, I strongly advise you to do so before reading this post-mortem due to spoilers. Download the map by clicking here

A Post-mortem? What's that?
This is a forensics term meaning "after death" which has been adopted by game developers to describe their personal reflections and analysis of their work. The goal of a post-mortem is to analyze a project's strengths as well as its weaknesses, and to share any lessons learned over the course of a game's development.

Background

Totality emerged as a fusion of several different ideas I had for maps. I first conceived of the name just after the release of *Monstrosity* in 2013, promising myself that my next big CTM (Complete the Monument) map would be sci-fi and space themed. But the idea remained rather nebulous for several years. The few concrete ideas I had for the map ended up making their way into other projects such as *Diversity 3*.

Separately, I had an idea for a parkour map where the player would have to hurry and patch holes in a leaky underwater dome surrounding a city full of helpless robots. Later it would be revealed that a corrupt corporation was puncturing the dome to create mass panic and sell more of their waterproof robot parts. Although the final map strayed from this original concept, it laid the foundation for *Totality's* futuristic setting, tongue-in-cheek tone, and colorful lore.

Design

Everything finally came together over five years later with the creation of the "holoblock," an endless supply of glass blocks that the player can use for defense, exploration, and puzzle solving. Every facet of the map's design stemmed from this central mechanic.

Using some scoreboard and command magic, this item lets you place glass blocks infinitely.

Before I began designing the map on paper, I identified four design pillars that would become the foundation of the map's design. When making design decisions, I would always think back to these core pillars to make sure every element of the map was working toward the same design goals. *Totality's* design pillars were as follows:

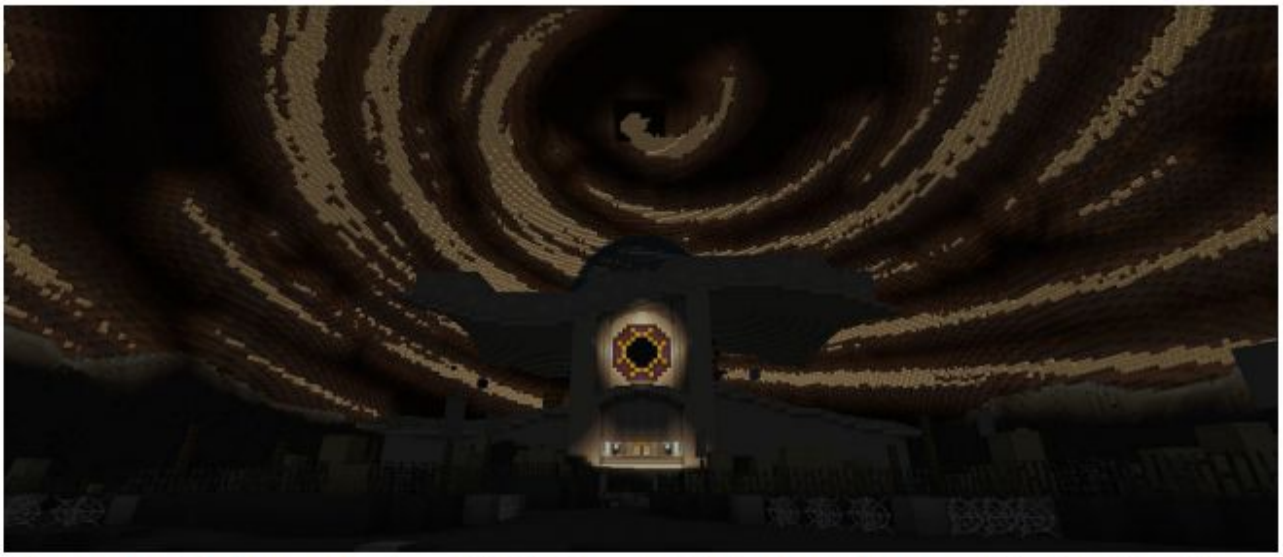
Accessible: I wanted to make a map that could be enjoyed and completed by the average Minecrafter, without losing the tactical building and challenging dungeon crawling that is characteristic of CTM maps. To this end, I determined that the punishment for death should be very light (respawning at a nearby checkpoint) and that there shouldn't be much resource management (mostly unbreakable gear).

A tips system was implemented to help ease newer players into the map. There's also an NPC called the 'Seer' that can give out hints on where to go next.

Nonlinear: With building being the primary focus of the map, I wanted to make sure the map would play to Minecraft's strengths. One of the selling points of Minecraft is that you can reach any place you see, and I decided the map should embrace this

concept. In other words, I wanted *Totality* to be a uniquely "Minecraft" experience — an open sandbox world to explore containing dungeons that you can approach in a variety of different ways.

Stylish: I'm bored of maps that are mostly set in underground cave systems. Right off the bat I knew that I wanted *Totality* to look and feel different from the average map. I wanted it to be whimsical and flashy, with eccentric, modern architecture and visually stunning backdrops. I wanted combat to feel fluid and exciting, which led to my decision to make each piece of equipment feel powerful and useful in its own way.



Shoutout to @abrightmoore whose MCEdit filters enabled me to make some wicked cool backdrops. First a png image was turned into a schematic using SpriteCraft, then the resulting image was stretched into a dome using Hemisphere.

Compact: This fourth design pillar was created pretty much out of necessity. In order to achieve my other three goals, I decided that I would need to keep a pretty close eye on the map's scope. Not only would a massive, 'stylish' city world take ages to build, but it would be much easier to get lost in, and thus make the map less accessible to newer players. The first thing I did when creating the world in MCEdit was place a 200-radius dome, which would go on to contain almost the entire map. I decided that my map should be 'ambitiously small' — a dense and intimate sandbox space instead of a massive, sprawling world.

With all that out of the way, let's get to the meat of this postmortem: what I've identified as the successes and shortcomings of the map.

Successes

Custom Monsters: I believe the custom monsters steal the spotlight as the map's biggest success. At first I was just planning on using default mobs — but they felt out of place in such a futuristic setting. So I set out to create new enemies with unique effects and behaviors. I was very worried during development about the custom mobs growing annoying, so I made sure each one had some sort of weakness that could be exploited by clever players. For example, the Rocket Scientist mobs deal a lot of raw damage but are very slow, making them vulnerable to ranged weaponry and also making them easy to block off with walls. The Crumbling Golems drop a silverfish each time they're hit, making them effective at overwhelming players in straight combat, but knocking the golems into lava or pouring water at their feet makes short work of them. Overall the custom mob behaviors reward creativity in combat, which I believe works incredibly well with the focus on building and exploration.

Areas were built around the designs of the monsters rather than the other way around, which helps make the monsters more fun to fight! For example, this areas features lots of ledges and choke points to make its golem inhabitants easier to deal with.

Visual Style: I believe I nailed the visual design of this map. Buildings are nice and rounded with domes and arches but also have some fun angular accents. My original plan was to make the entire city black and white with only a few splotches of color, and while the city ended up being more colorful than I envisioned, I still think this intention shows and makes the city really pop. Minecraft has a very limited selection of

modern-looking blocks so I'm very pleased of what I managed to pull off. Making the map relatively small in scale paid off in the visual department because it meant I could give every building plenty of attention. It also meant I could add fun little details like traffic cones and gumball machines using player heads.

If you look close enough, you might even be able to find the TARDIS!

Length: Another strength derived from limiting the map's scope, I believe the map's relatively short playtime does it favors. Bigger isn't always better, and sometimes the best plan for a game is to get in, show something cool, and then get out before the novelty wears off. *Totality* is approximately four hours long, which I believe is long enough to showcase its central holoblock mechanic but not so long that the areas and challenges start to feel the same. And for players who are itching for more, the fact that the dungeons can be played in any order adds a little replayability.

Workflow: Due to the release of Minecraft 1.13, I had to adopt an interesting workflow for this map that was taught to me by qmagnet. With The Flattening removing numeric block IDs, MCEdit is no longer an option for map development. However, after using it for over eight years, building without MCEdit just doesn't feel right to me. Instead, I built the map entirely in 1.12. The master copy of the world is still a 1.12 world. Any features requiring 1.13, such as monster spawners and certain items, would then be automatically placed and set up by a function that would run upon world conversion. This allowed me to build in 1.12 but use the powerful new 1.13 command syntax. Additionally, because spawners and items were placed with commands, it made it trivial to make balance tweaks and fix errors.

Shortcomings

Poor Signposting: I took a lot of precautions in this map to make sure players wouldn't get lost. For example, I decided to place the 'monument' in the center of the map as a landmark to help orient players no matter where they are in the city. I also built the dungeon entrances before designing the layout of the city, so they would hopefully be big and imposing and stand out.

Unfortunately, this wasn't enough — players still complain about feeling lost. In fact, I believe the reason why players feel lost is precisely *because* I built the dungeon entrances first. While the dungeons are arranged in a nice, orderly fashion, the rest of the city came out haphazard and maze-like, making the overworld feel much larger than it actually is. If I had been more flexible with the dungeon placement, perhaps it would have made the overworld easier and more intuitive to navigate. The lesson here is to design big, open areas like this more organically, paying attention to how the level flows instead of establishing a rigid system of dungeon placement.

Building all of the dungeons first made arranging the city a nightmare.

Performance, or Lack Thereof: One of my design pillars was to make the map compact. I wanted a lot to explore in a small amount of space. Obviously I didn't learn any lessons from building *The L'brour Mansion,* because Minecraft really doesn't like dense maps like this. Especially not Minecraft 1.13. I had to come up with a lot of fancy tricks and optimizations just to get this map to run on an average PC.

The biggest optimization I implemented is entity pop-in. Armor stands and item frames seem to take a heavy toll on 1.13's framerate even when they're too far to render, so npcs far away from the player are deleted and only resummoned if the player enters the area that they're in. Particle effects were another challenge and had to be kept to a minimum. These optimizations have done a lot to help, but if you're planning on making a dense map like this then you should definitely be very mindful about performance, testing on several machines throughout development.

NPCs and decorations are deleted when the player is far away to help ease the load on rendering.

Vague Marketing: I made the classic mistake of being so focused on development that I didn't put enough effort into the marketing materials for the map. By the time the map was finished, I didn't have a trailer, and I was still fairly uncertain how to describe it. It's kind of an adventure map and kind of a CTM, but it doesn't perfectly fall into either genre. "Build your way to victory with an infinite supply of holoblocks as you explore Sol Corp City and unravel its dark and mysterious past …" was my original tagline, and in retrospect, it just doesn't work. It hints at there being building and exploring, but beyond that, it's just too vague to get anyone interested or excited. And in a world where first impressions are everything, a vague call to action is a critical blunder. Without the support of the CTM community and the folks at minecraftmaps.com, *Totality* could have easily slipped under the radar. Never underestimate the importance of good marketing!

Too Much Exposition: Perhaps the weakest part of the entire map is the beginning sequence. The story begins with the player crash-landing on Earth, but besides some fancy sound effects and particles, the player's background is explained

solely through text. Bradmall's excellent Let's Play video series reveals what *Totality's* opening could have been — a quirky, cinematic experience to set the stage for the coming adventure.

As players make their way through the map, they unlock short playable vignettes that explain the backstory of the map. These were added near the end of development after playtesters commented that the plot was confusing and somewhat boring. While the vignettes helped clarify the story, I believe they could have been taken a step further by making them more interactive. Exposition with a fancy backdrop is still exposition after all. If I could start the map over again, I would put less emphasis on exploration and more emphasis on engaging story delivery.

I've now shared what I believe to be the primary strengths and shortcomings of *Totality*. Post-mortems always interest me because they can expose some of the design decisions we make subconsciously that don't serve our overarching vision. They can also expose tension between the very pillars of a game's design — in this case, I struggled to make the map simultaneously accessible and nonlinear because I was too strict with the level design.

Fellow mapmakers — I strongly encourage you to try writing one of these reflections for yourself. In our community we talk a lot about hard (technical) skills but rarely discuss soft (design and production) skills. There is a lot that we can learn from each other. Write a post-mortem, and post it for the world to see!

About the Author

Cold Fusion is a self-described "gameplay nerd" who has been developing Minecraft maps for eight years. He is best known for his work on the *Diversity* maps, as well as for his challenging CTM experiences. Check out his maps by visiting coldfusionmaps.com.

STEPHEN REID

@IMMERSIVEMIND

Immersive Minds

Working with people to develop skills for:

- Work
- Learning
- Life

Communication
Citizenship
Critical Thinking
Numeracy
Analysis
Evaluating
Teamwork
Problem Solving
Creativity
Literacy
Negotiation
Justification
Empathy
Decision Making
Enterprise
Self Confidence
Judgment

ICT in Education...

Using technology creatively to enhance learning across the entire curriculum and...

- Outdoor Education
- Employability
- Environmental Science
- Study Skills
- Motivation/Aspirations
- Alcohol Awareness
- Anti-Bullying/Cyber-Bullying
- Entrepreneurship
- Internet Safety
- Social Media Engagement

Global Network...

Active projects with teaching staff in these countries...

Pioneering Games-Based Learning...

Using games and play to enhance and support curriculum learning and life skills development, in children and adults...

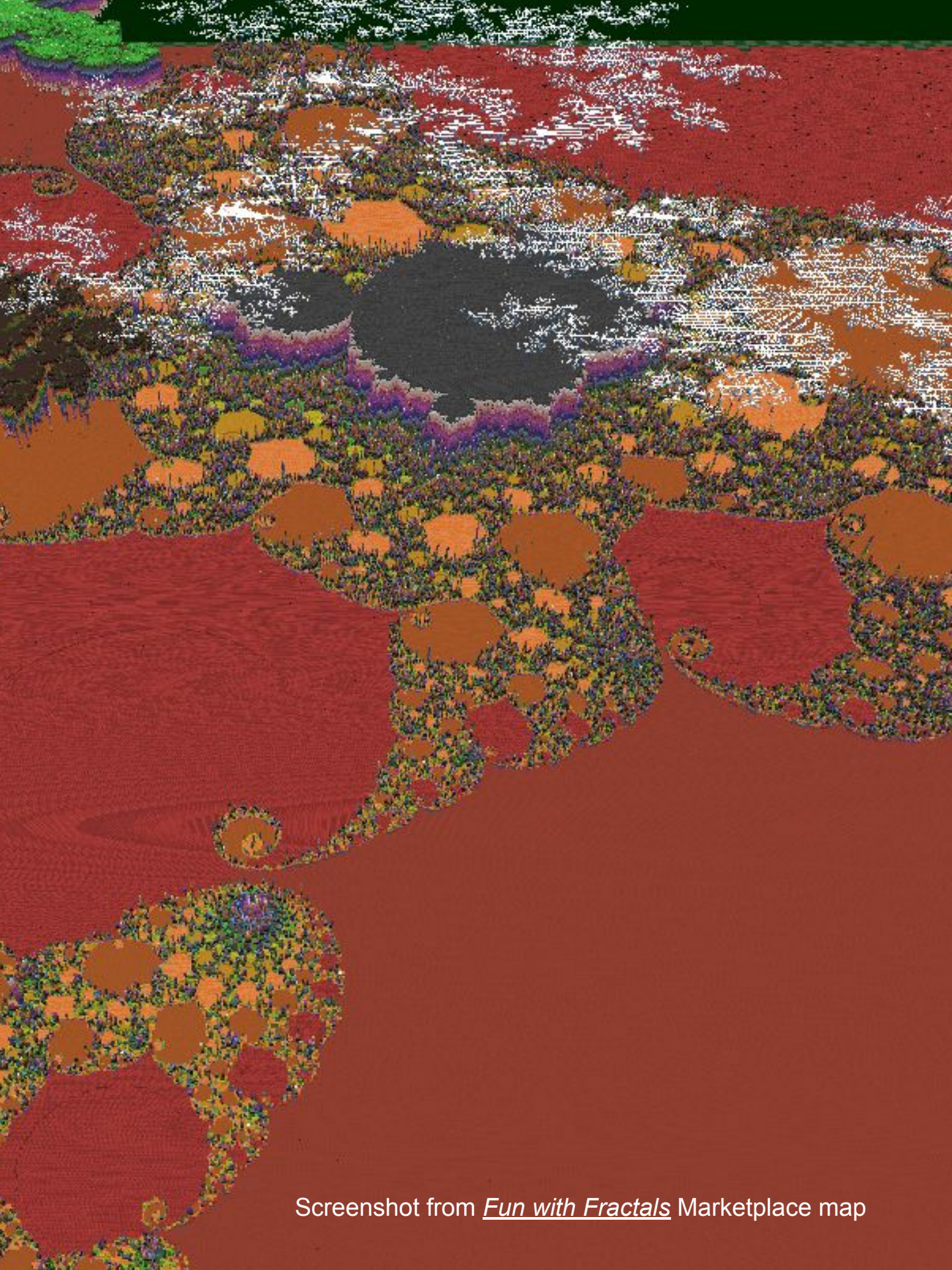
- Pupil Training
- Professional Development
- Resource Development
- Teaching
- Public Speaking
- Project Management
- Consultancy
- Coaching
- Curriculum Design

Minecraft in Education...

Using Minecraft to support learning across the curriculum...

A global Minecraft server dedicated to training and supporting teachers and parents.

In-Game CPD sessions per month



Screenshot from *Fun with Fractals* Marketplace map

PROCEDURAL GENERATION vs BUILDING BY (VIRTUAL) HAND

By Keith Ross

You've heard that old saying "you can't teach an old dog new tricks"…
Well, I am that "old dog". The extent of my modification repertoire is creating custom texture packs, using command blocks to achieve various effects and basic (and I do mean basic) redstone circuits.

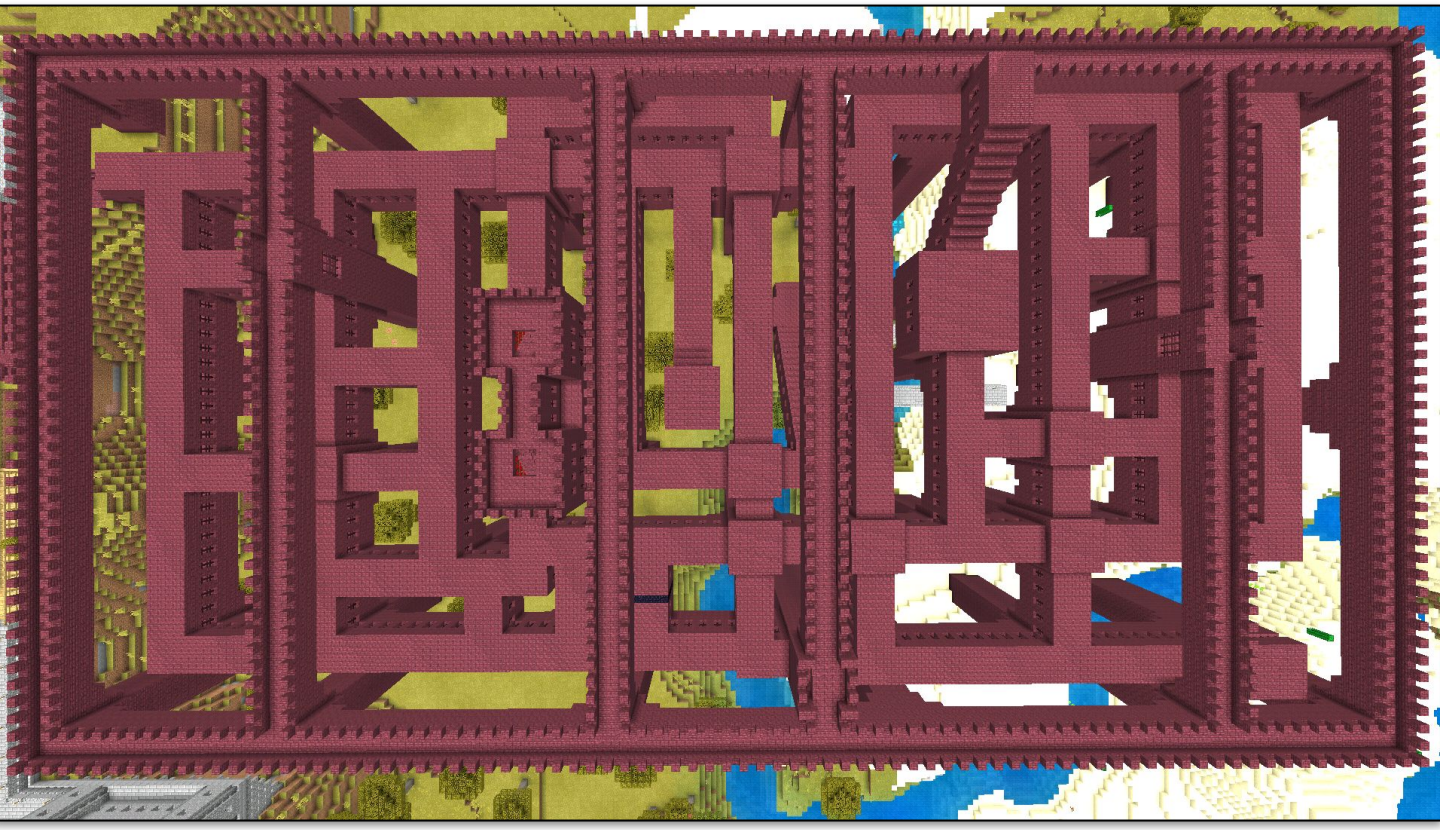
The idea of using 'code' to automatically create structures in a world, is as far as I'm concerned, defeating the whole object of the Minecraft platform. Yes, I understand that it can be used only to create structures that already exist 'in-game' and that as a result, it will save a lot of time if you want those structures within easy reach as an integral part of your build. But…..

What about the satisfaction of knowing that YOU have built EVERYTHING on that map, block by block, from the ground up. Surely that has to count for something?

Using code to shortcut the building process is akin to giving a box of Lego bricks to a robot, handing it the instructions and telling it to get building. It totally negates the Minecraft concept.

I have hand built a number of in-game structures on one of my maps including a Nether Fortress, a Woodland Mansion, an Ocean Monument, a Jungle Temple and a massive Village.

PROCEDURAL GENERATION
VS
BUILDING BY (VIRTUAL) HAND



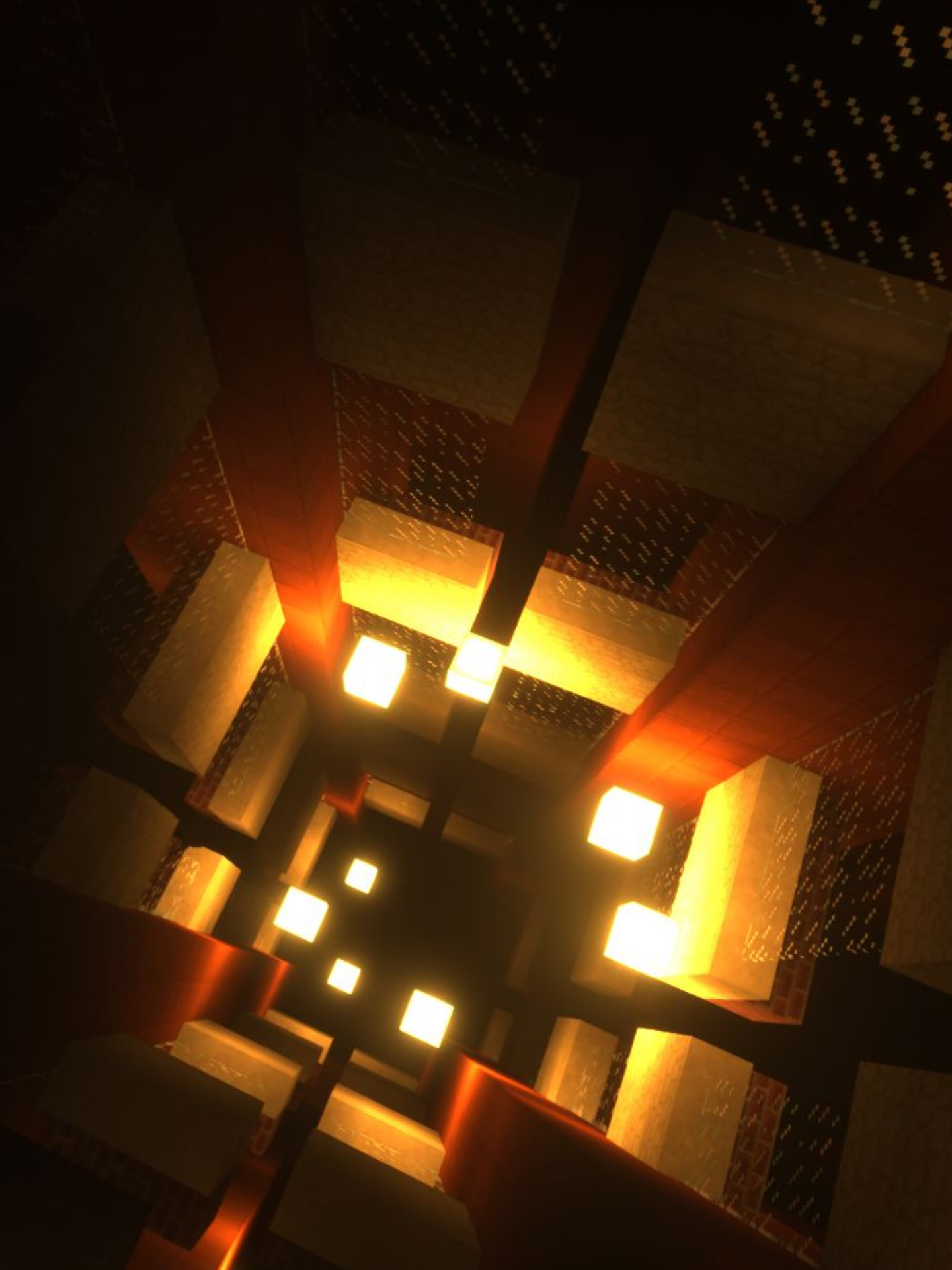
Yes, these structures took time to hand build, but being able to (virtually) step back from the build and see how it was coming along, was far more rewarding than what I imagine I'd feel after entering a bit of code and running it.

I'm sure that Procedural Generation has its place and that it will benefit builders for various reasons, but I think I'm going to stick with the tried and tested and more satisfying method of hand placing the blocks for my builds.

The world containing all these builds is called Mesaville and can be found on MCPE-DL

https://mcpedl.com/mesaville-map/

7th March 2019
k_ross69
vintagegamer069
keithross39



BIG PROJECTS FASTER

Weston Ahern

Mirror's Edge Catalyst

A look at Modular Design In a Parkour Map

What Makes A City?

- **Districts**

Residential, Commercial and Industrial areas - typically broken up into low, middle, and upper-class pockets.

- **Landmarks**

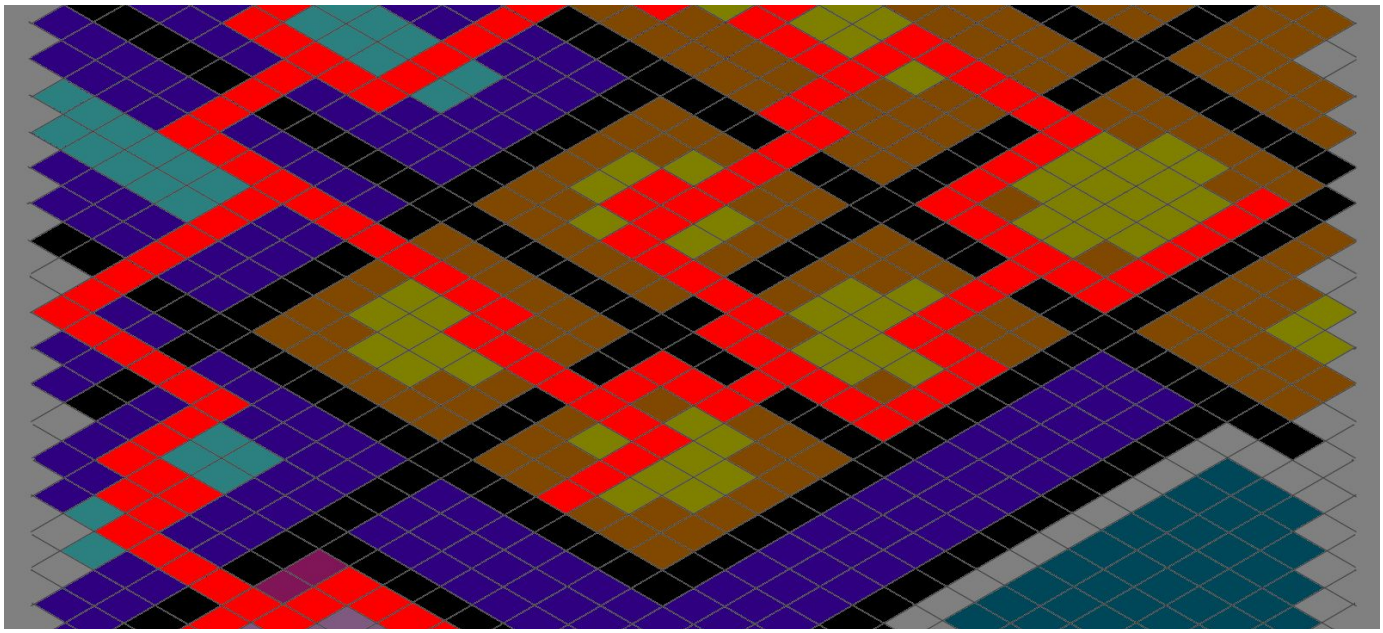
Big visually distinct & easy to recognize structures that help the player to orient themselves in a larger world, and understand where they are.

- **Public Services**

Police Stations, Fire Departments, Schools, Government Buildings, Arts & Museums, and the many forms of Transportation that make up a large city from Highways, Subways or Train Stations, to Bus Stops, Airports, and Parking Garages.

- **Ads, Billboards, Logos**

Big, bright colorful signs and business names, telling the residents of the city where to shop, what to eat, which cars to buy, and who each building belongs to.



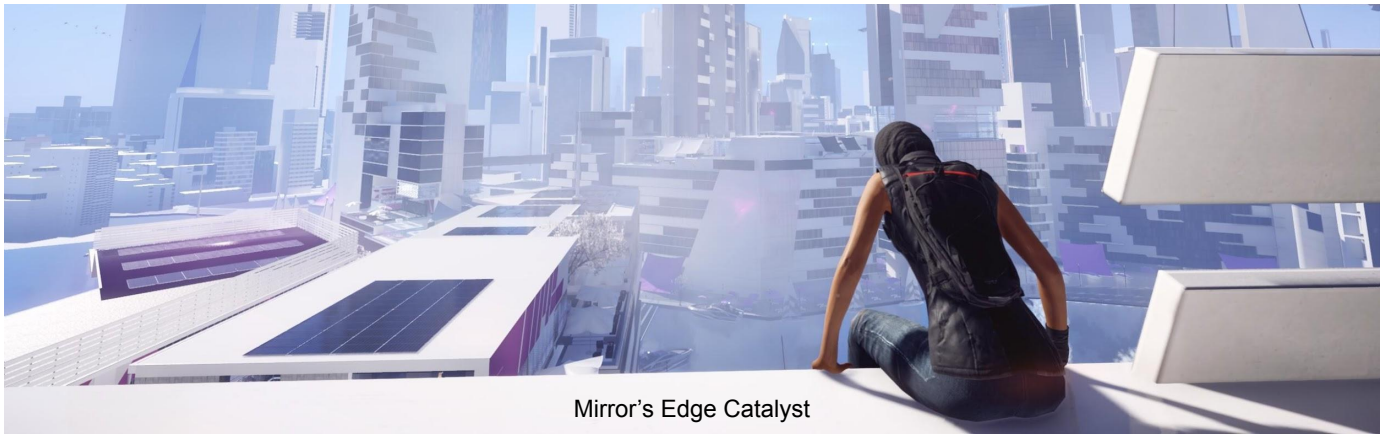
Playable Area in Red

One of the disadvantages of creating a 3D space for a player to explore, is accounting for every possible direction the player may look out at from any number of locations inside the game.

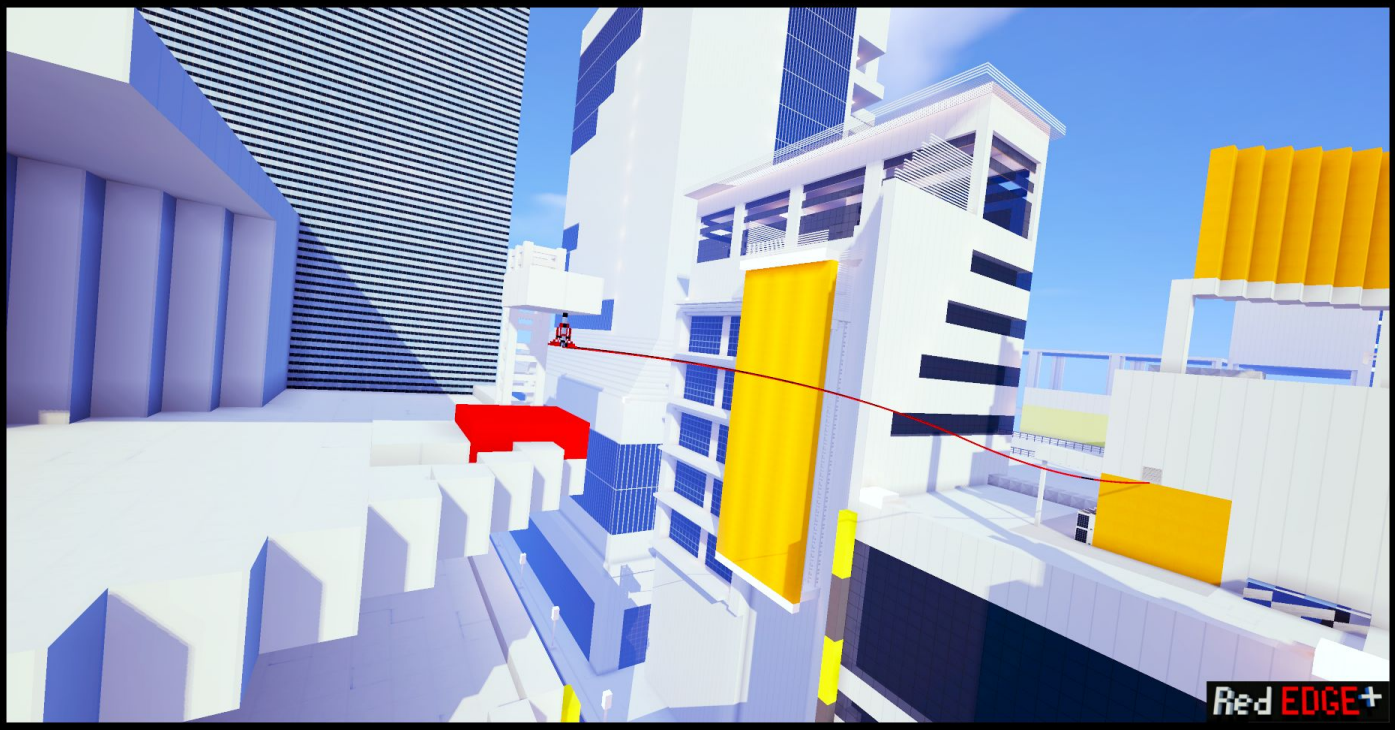
In a 2D game, the background is usually represented (in various levels of complexity) by a panorama or painted image of some sort, to visually indicate what would be seen in the distance of that location.

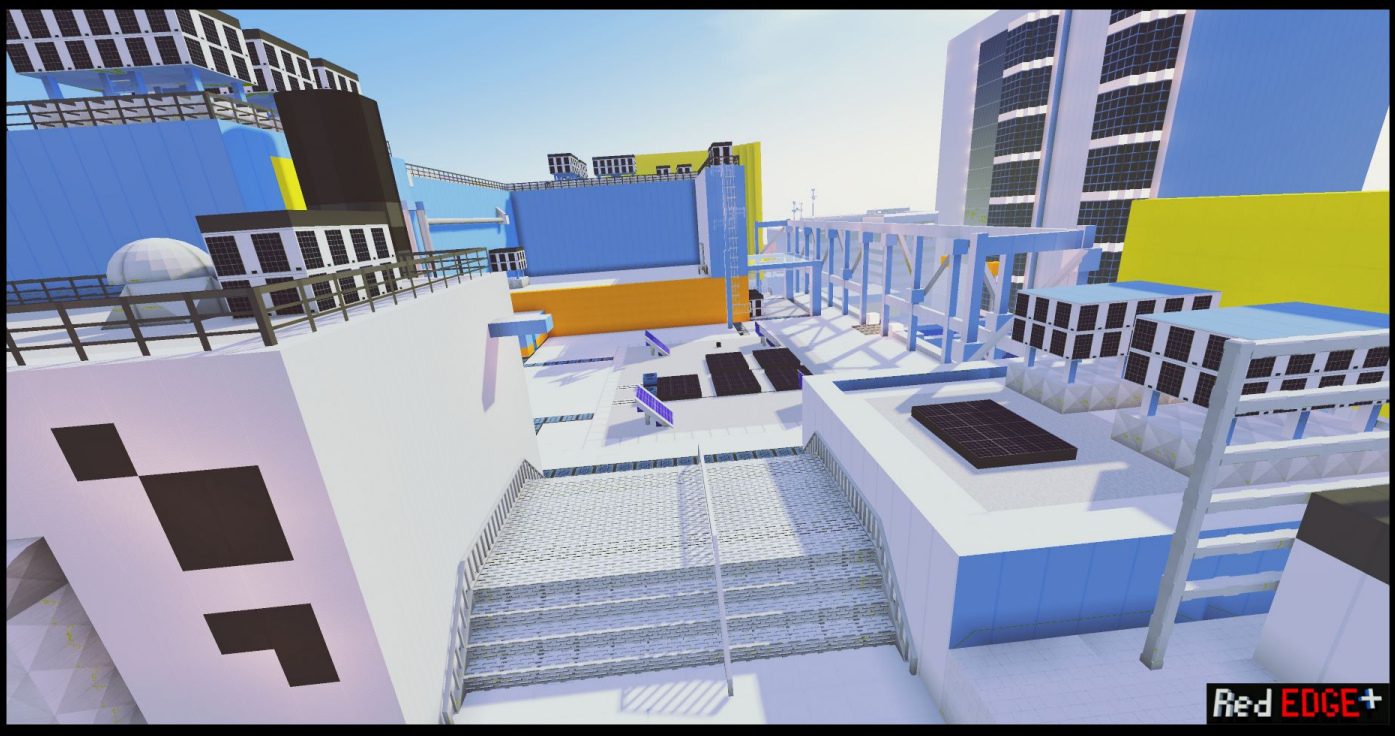
Working in Minecraft - in a 3D environment, means that if we want our city to look bigger than can really be explored, we still need to BUILD structures in the distance for the player to see. In a parkour city, where the player will be climbing over rooftops and looking out over them from above - this gives our city a sense of depth, where the final layer to any distant view is the sky itself.

This helps us to understand how our city might be laid out, but how are we going to start such an enormous task?



Mirror's Edge Catalyst

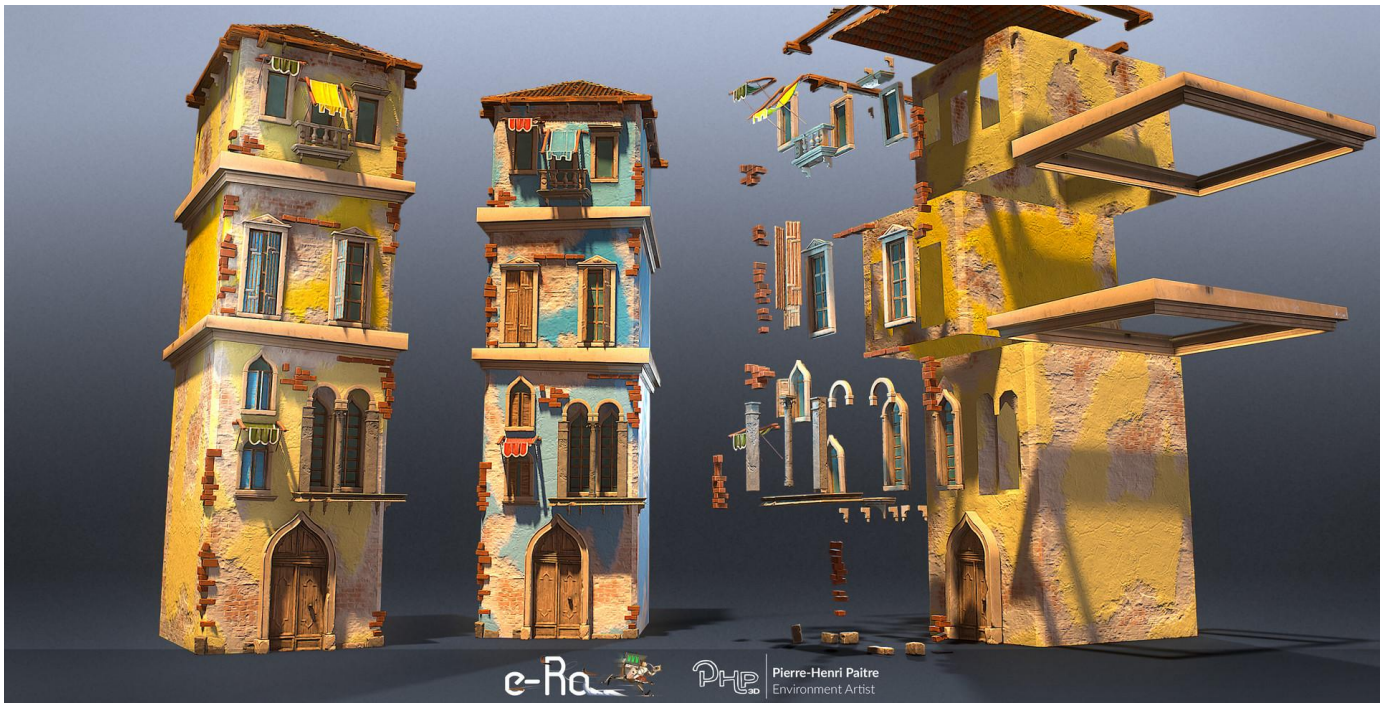

Red EDGE+


Red EDGE+

Modular Design

Asset Workflows for Modular Level Design: By Joshua Kinney

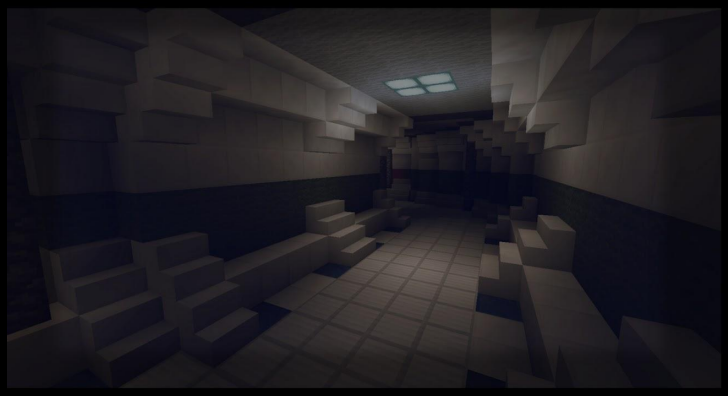
Modular design allows us to quickly arrange pre-built objects into a large connected set piece. Getting the hard work done and freeing up time to focus on the smaller details.



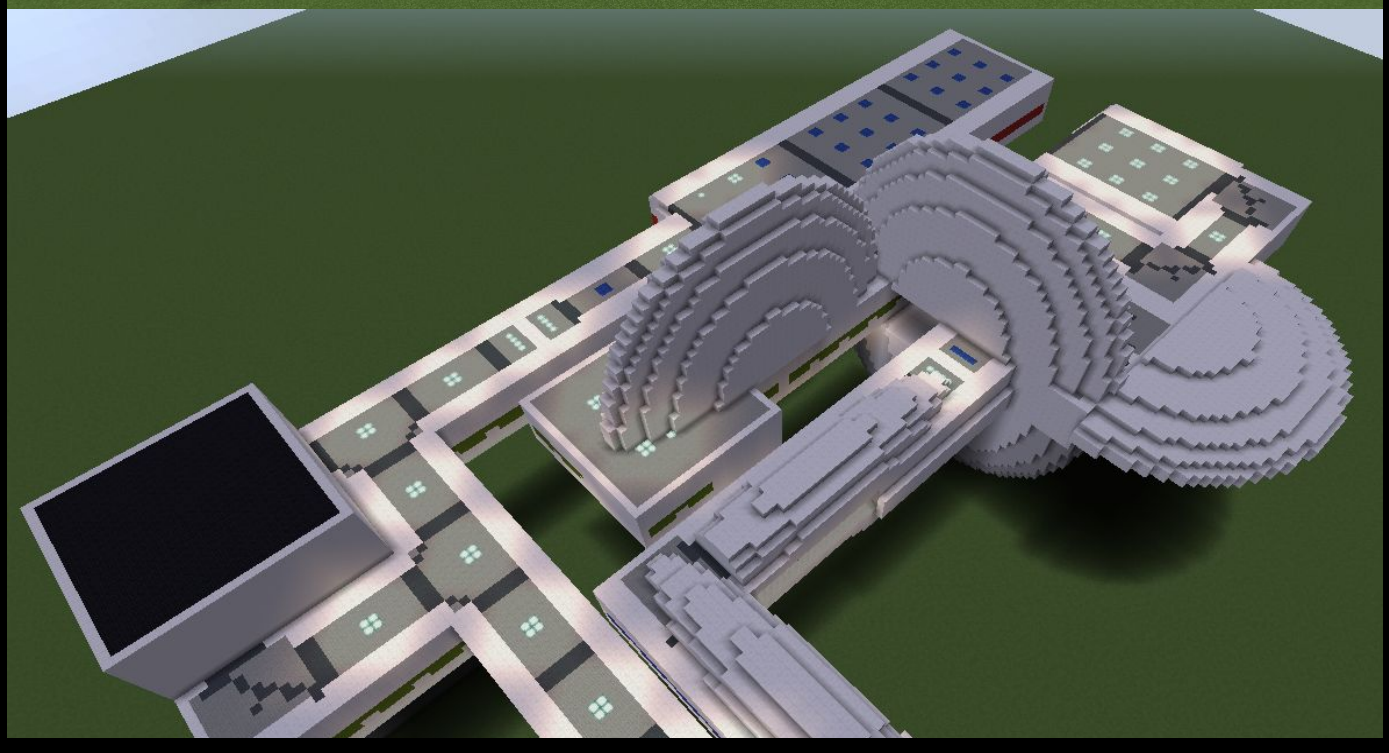
The first step in this process is the creation of our reusable assets - common things we will be re-using all the time in our map. Houses can be built with various types of doors, windows, decorations, height differences, and rooftops.

City streets can be broken up into straight pieces, turns, intersections, dead-ends, street-lights. We might create trees, bushes, or benches to fill in a more natural landscape.

These assets are the paint we will use to bring our 3D environment to life - but we still need a paintbrush...




Modular Space Station in Vanilla MC

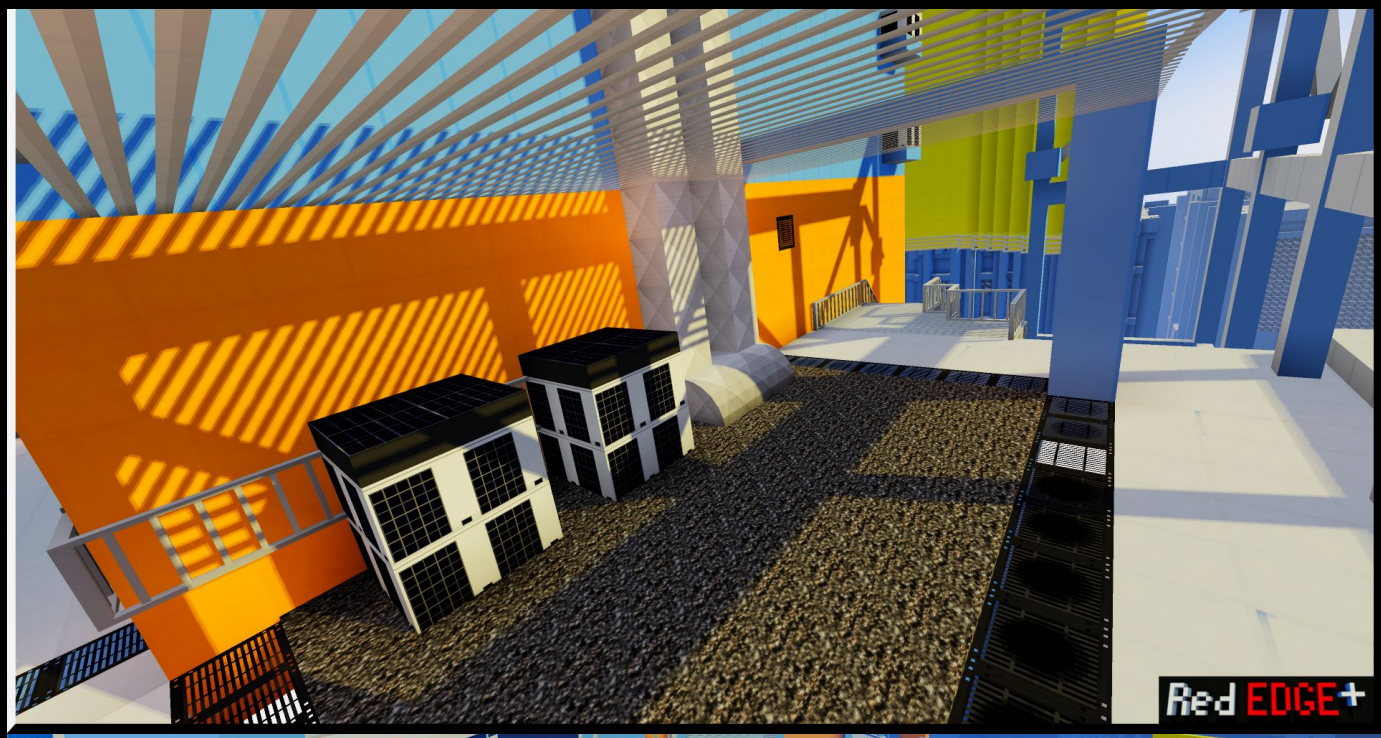
MCEdit

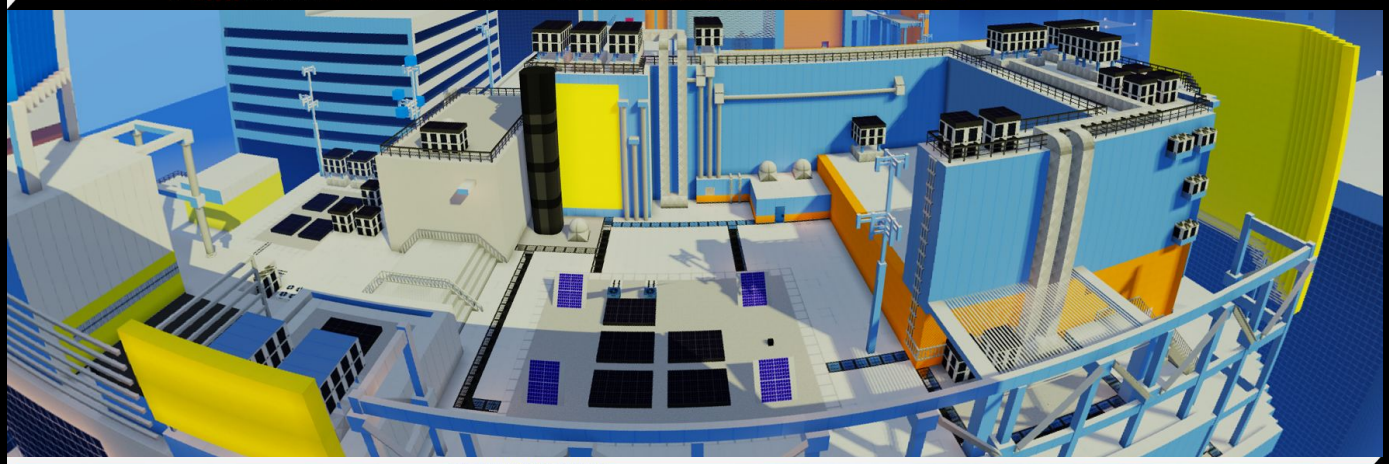
A tool with a multitude of features - MCEdit allows us to make big changes to the environment of our maps in just a few swift clicks of the mouse. Among its many uses, the ability to make a 3D selection in our map and save that highlighted selection, gives us the ability to start using our pre-built modular assets.

It's as easy as loading a tree, copying it, and pasting it around our environment as much as we like. We can rotate it around, change the color of the leaves, use it like a stamp - until we have an entire forest in just a couple of minutes.

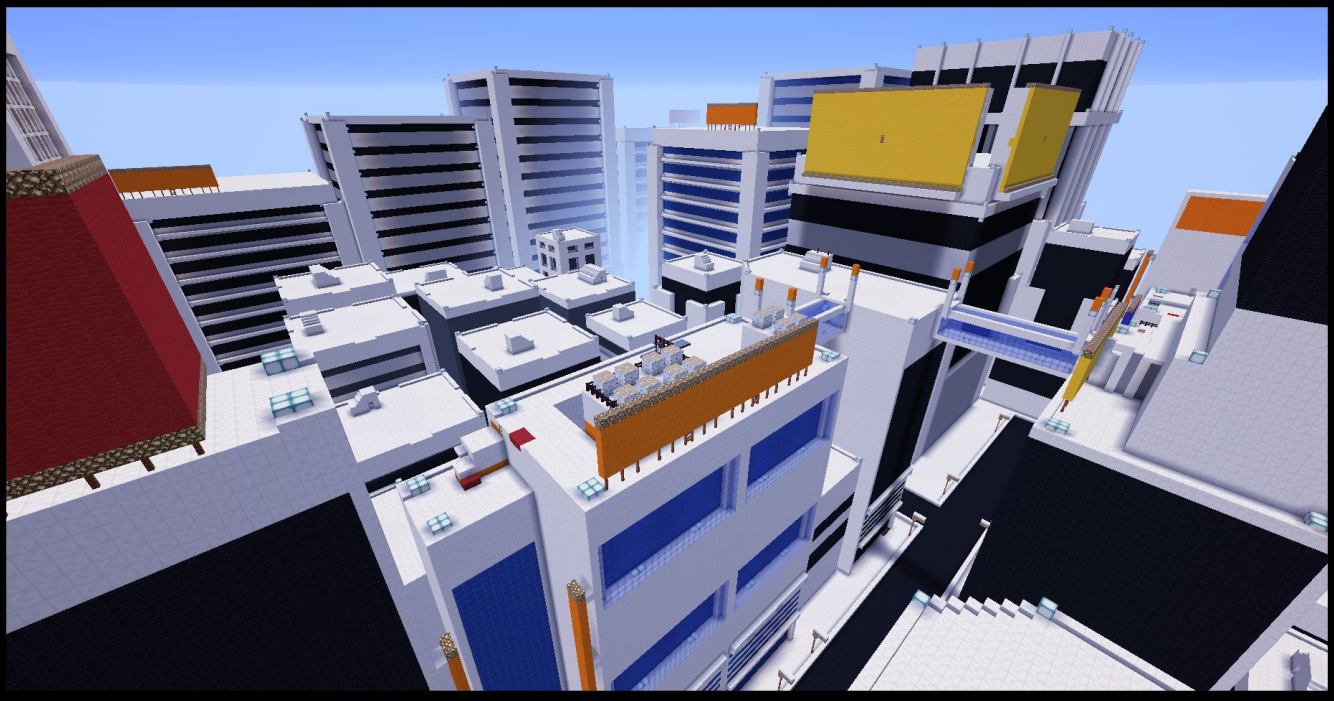
It might look a bit odd to see the exact same tree all over the place though, and that's why creating a multitude of different assets (*see previous page)* can help to add variety to our build, and look a bit more organic.

The same goes for our big parkour city - MCEdit helps to get the big, time-consuming work done fast: Make a skyscraper, dress it up with windows, put some billboards on the roof, add all the lighting. When that can all be accomplished in an hour or less, it leaves more time to focus on the details that matter, the things the player is going to see and interact with the most. So the next question is - What goes on a rooftop?


Red EDGE+

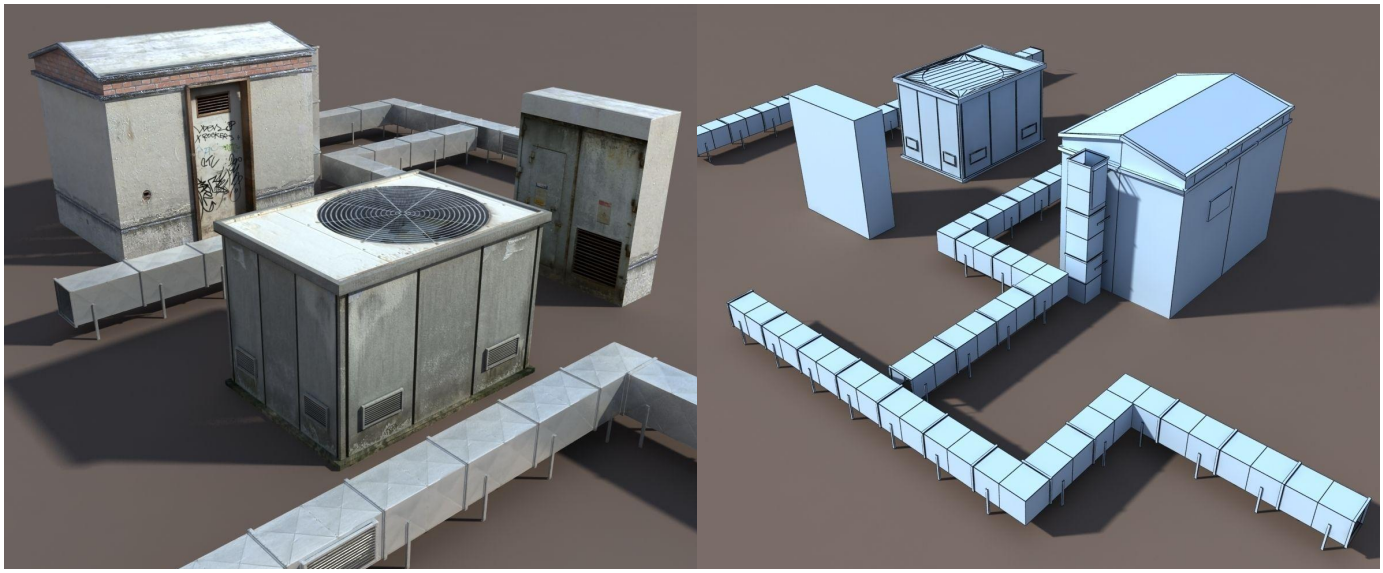



Red EDGE+




RedEDGE VANILLA

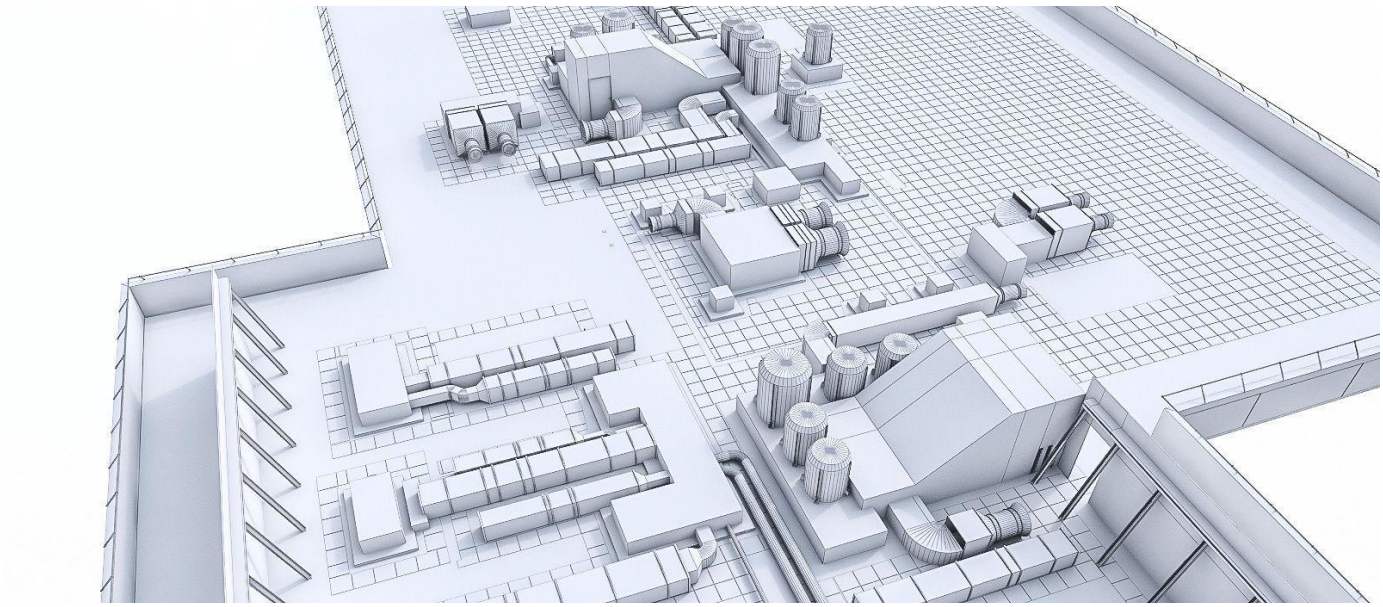
What Makes A Rooftop?



Roof Access, AC Units, Air Vents, Railings, Solar Panels, Pipes, Fences, Security Cameras.

In our parkour city, rooftops are the playground our player will interact with - and that's what it should feel like, a playground. Our goal is to create interesting and unique obstacle courses - and give the player the freedom to choose how to get through.

Will they take the slow-but-safe route? Or the difficult-but-fast route? This is our chance to be creative, to turn these towering rectangles into believable buildings.



Putting It All Together

We have our pre-built assets, we have the tool to paint our 3D environment with all the different things we've made, and we've freed up a lot more time to be creative - and plan out fun, interesting things for the player to run, jump, and climb over on our rooftops.

From this point on, the sky's the limit! Modular Design has given us the ability to pull together huge, creative environments quickly - and use the time we do have to add all the little details that make such a big city feel lived-in, and fun for the player to explore.

Weston Ahern

RedEDGE	rededge.carrd.co
My Website	djear.com
Twitter	@Huepow00
YouTube	TheDJEAR
E-Mail	djear.info@gmail.com

Screenshot from *Diversity 3* by @theqmagnet and friends


IT'S TIME TO GO PRO.
NEW MAP-MAKING OPPORTUNITIES AWAIT YOU AT WWW.PATHWAY.STUDIO.
@PATHWAYMC

Emerald Chambers

Minecraft's first "traditional" Roguelike; a shameless self-promotion

By Dieuwt who doesn't really know what he's doing but oh well!

What's a "roguelike"?

The term "Roguelike" comes from an old computer game Rogue. In the game, you had to traverse randomly generated dungeons, while battling enemies, finding items, and trying to reach the end. Certain aspects of the game were new at the time; these eight "rules" indicate how "rogue-like" a game is.

- **Randomly generated dungeons:** so that no run is ever the same, and nothing can be predicted.
- **Permadeath:** once you die, there is no way to continue, and you must start over. There are no exceptions.
- **Turn-based gameplay:** you get as much time to overthink any decision as you wish.
- **Non-modal gameplay:** all actions have to be avaialble at any time (no abilities or items are "unlocked" on the way).
- **Complexity options in actions:** there have to be multiple ways to do most actions, depending on your situation.
- **Resource management:** all potions, weapons, food and other items are very limited, and it is required to find out optimal strategies to use them.
- **Hack-and-slash-based gameplay:** monsters are defeated through stabbing them in the face, and there are little peaceful options.
- **Unknown items and regions:** players have to explore the area's, their inventory and the combinations that differ every run by trying out their luck.

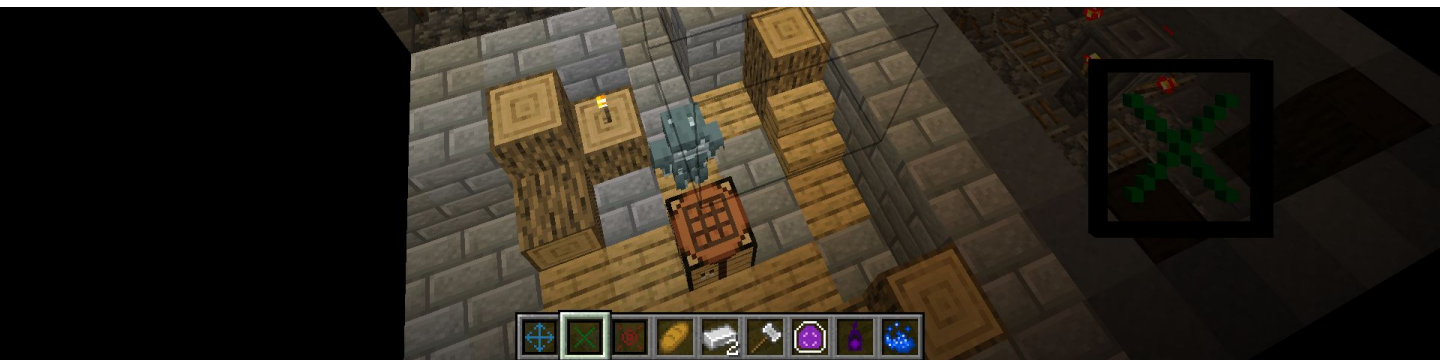
For more info about roguelikes, visit the Wikipedia page. (https://en.wikipedia.org/wiki/Roguelike)

So what is Emerald Chambers?

Minecraft has had dungeon crawlers in the past. They were all pretty good, but none of them were *really* Roguelike; mainly because they had no turn-based gameplay, In fact, Emerald Chambers is the sequel to another dungeon crawler I made, Ruby Caverns. But I wanted to go further into the genre.

Emerald Chambers is the first map in Minecraft that takes all the traditional Roguelike elements into play: the game is top-down, everything is fully turn-based, and there is no in-game progression. There are currently 3 classes to play as, with many items to collect, spells to cast, enemies to slay, bosses to battle, and a total of 11 floors to explore!

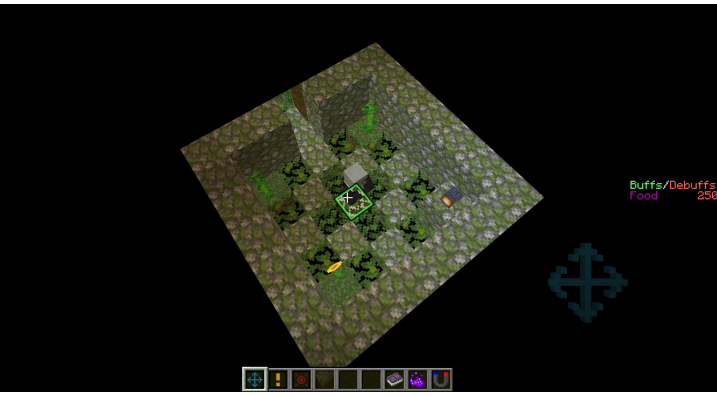
All this comes at a hefty price (apart from your computer partly breaking down): the game is **VERY** hard. You have a small health pool, healing is strict, and enemies are powerful and gain strong abilities later on. Even the free items on the ground cannot always be trusted: weapons and armor might be cursed, orbs may backfire, and some items will not help you while taking up precious inventory space. Use what you get, and get what you use!

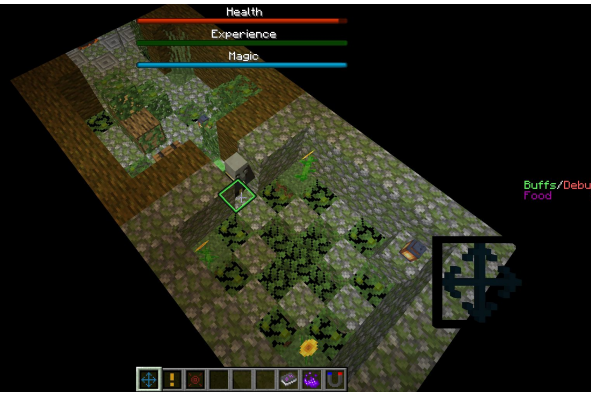


Controls

How does a 2D game in a 3D work? Let me quickly guide you through the controls, by starting a new run.

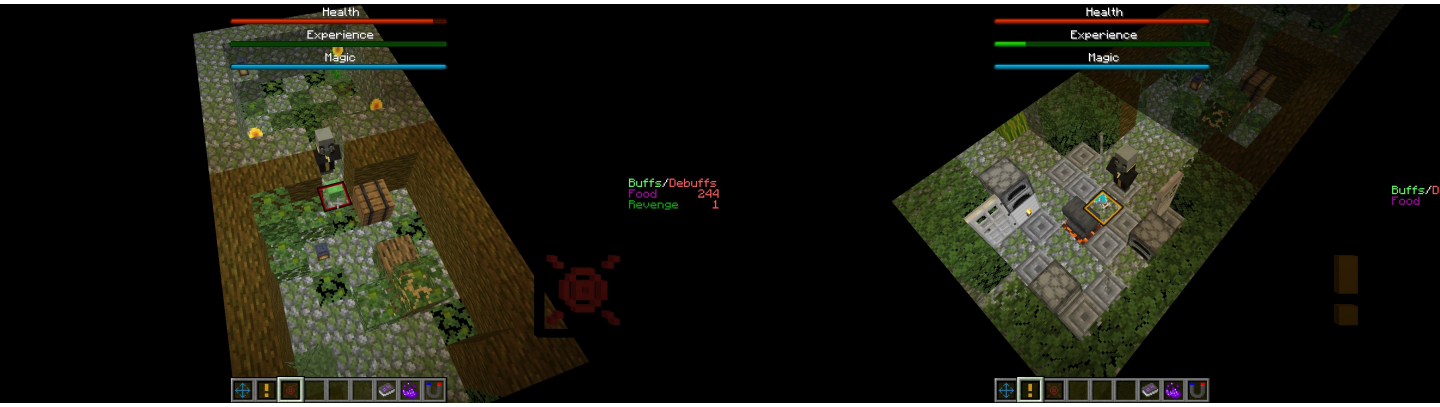
Every run starts in the Lobby. You can pick your character: the Blaze, the Spirit, or the Evoker. I'll go with the Evoker, my favorite class, for now. You already get your starting items in your inventory, to check out their powers. Right-click the staircase to begin a run!



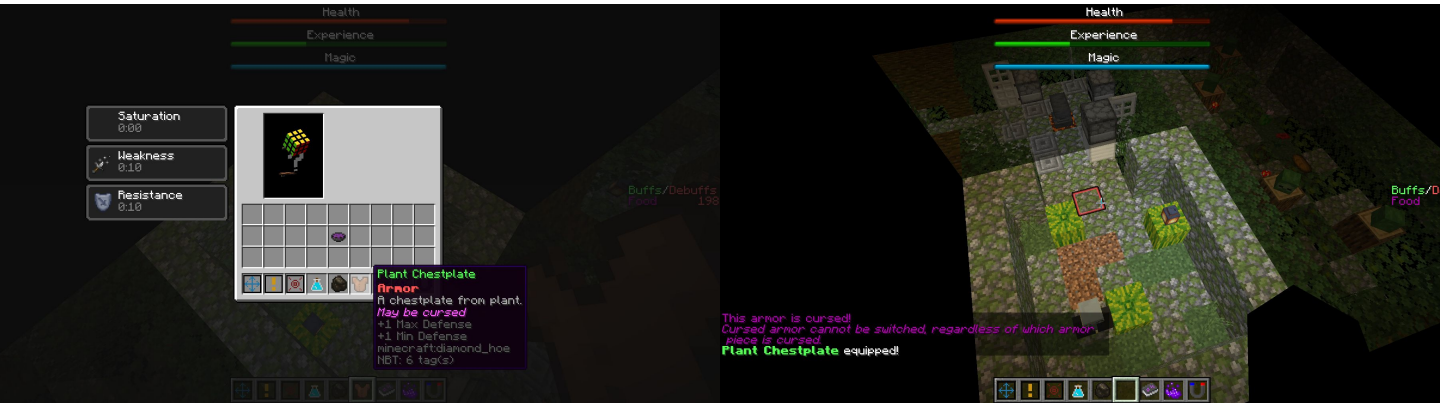


After a small cutscene about your backstory, you'll find yourself at the start of the **Deep Woods**. You have three important "buttons" in your inventory: **Move**, **Use** and **Attack**.
That glowing square is your Cursor: your current selected square in the 2D space. If you, for example, click **Move** on a free space, your character moves there!

It looks like an enemy is in the way! Sure, you can keep walking, but it will attack you. To kill it, use the **Attack** button on the enemy, which must be standing next to you (horizontally, vertically or diagonally) until it dies.



Moving on, more and more of the dungeon will be revealed. Now, you'll see two helpful things on the path: a Utility and an Item on the ground. If you stand next to or on an item, use **Use** on the item to pick it up!
Utilities exist to spend resources on. Currently we don't have anything useful, so we move on.



A piece of armor! It says it might be cursed, though… can we take the risk! Of course we ca- oh, it's cursed. Great, now we cannot equip any armor until it gets de-cursed!
The potion turns out to be a Potion of Power. By taking the risk and drinking it, my character gained the Power effect for 20 turns, which increases attack damage! Next time the same potion is found, it will say its effects in my inventory.

That’s the game!

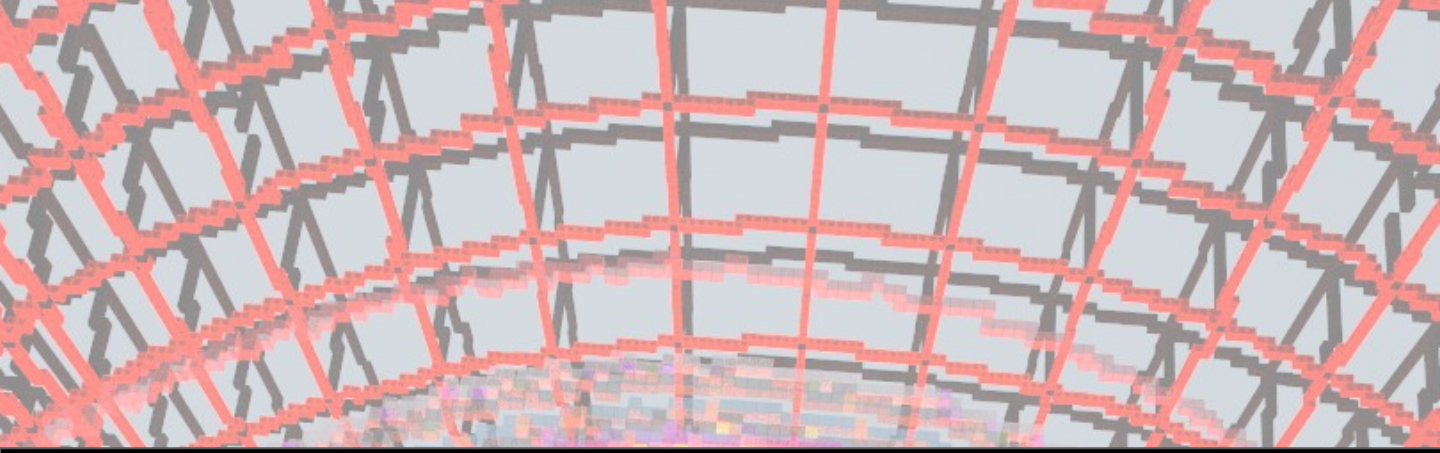
That is how the game works! Aiming your cursor, right-clicking with the right item in your hand, and you can do anything. There are many things to see and collect, like:

- ★ 3 characters, all with different playstyles, abilities, and final bosses!
- ★ 10 potions, 10 scrolls and 10 casts all with random color codes!
- ★ More than 20 Tools that have various uses!
- ★ 20 Weapons and 20 Armor pieces, all with different stats!
- ★ More than 15 Utilities to spend your resources on!
- ★ 19 different bosses, over 11 different floors!
- ★ 9 Challenges for the hardcore dungeon crawlers!

And many more things such as secrets, lore, dangers, everything you’d expect from a dungeon crawler. Also, I frequently update the map to fix bugs and add new features! Here are some more screenshots to look through. Good luck!

Get Emerald Chambers on http://www.minecraftmaps.com/adventure-maps/emerald-chambers
Get the prequel Ruby Caverns on http://www.minecraftmaps.com/adventure-maps/ruby-caverns


MCCONTENT
The future of
Minecraft maps
is almost here.
Here to help you develop the next Minecraft masterpiece. Join and get started today, because it all starts with you.
mccontent.net/signup
YOU'VE BEEN INVITED TO JOIN A SERVER
MCContent
DISCORD
Join



Making Terraforming fun and effective at the same time!

By LazerRock25

Youtube for map release trailers:

https://www.youtube.com/channel/UCWAkGLwbtRb4Twmnq5wO0hw?view_as=subscriber

Twitter:

@LazerRock1

Terraforming your map and making it have custom terrain really makes it come alive. The problem is, Terraforming is boring to do by hand, but McEdit doesn't currently work for 1.13, so you're kind of forced to manually place each block by hand, right? WRONG!!! Using the all-powerful power of command blocks, you can effectively terraform your map without having to spend hours placing blocks manually.

I will be showing you how to terraform the pro way by using this boring forest as an example.

Picture 1: Boring Forest

TECHNIQUE 1: Podzol spam

When terraforming a forest, remember the golden rule- spam podzol and coarse dirt. To do this, use the magic of command blocks. Make a command chain in this order:

1. /execute as @e[type=snowball] at @s run fill ~1 ~1 ~1 ~-1 ~-1 ~-1 podzol replace dirt
2. /execute as @e[type=snowball] at @s run fill ~1 ~1 ~1 ~-1 ~-1 ~-1 podzol replace grass_block

Power the chain and voila! Every time you throw a snowball at some ugly dirt (or grass block), it will turn it into podzol! Yay!

To turn ugly dirt and grass into coarse dirt, use these commands in a chain instead:

1. /execute as @e[type=snowball] at @s run fill ~1 ~1 ~1 ~-1 ~-1 ~-1 coarse_dirt replace dirt
2. /execute as @e[type=snowball] at @s run fill ~1 ~1 ~1 ~-1 ~-1 ~-1 coarse_dirt replace grass_block

TECHNIQUE 2: Better Trees

Eww, look at those ugly trees. We have to replace them with better ones. First, design your perfect tree.

Picture 3: Perfection

That's a much better tree! Now, to replace the ugly default trees with your masterpiece, the clone command will come in handy. First, get the coordinates that you need to clone the tree. Put a block in to opposite corners, and note down the coordinates.

Spam time!!!

Pictures 2a and 2b: Podzol Spamming

Picture_4:_Coordinates

Next, remove the blocks you used to get the coordinates (here they are the two stone blocks), and make a command chain in this order:

1. 1: /execute as @e[type=arrow,nbt={inGround:1b}] at @s run clone -105 105 -126 -99 118 -132 ~ ~ ~ replace force
   (replace the underlined numbers with the coordinates you noted down).

2. 2: /kill @e[type=arrow,nbt={inGround:1b}]

And voila again! Every time you shoot an arrow, a new, beautiful tree will emerge from where the arrow landed.

Picture 5: Beautiful Trees

TECHNIQUE 3: Filling in

Looks good and all, but the trees removed some of the land… Let's fix that!! To do this, we'll use some ocelots! Make a command chain in this order.

1. /execute as @e[type=ocelot] at @s run fill ~1 ~-1 ~1 ~-1 ~-1 ~-1 grass_block
2. /effect give @a minecraft:speed 1 2 true

Next, spawn some ocelots.

Picture 7: The Chase

After you spawned in some ocelots (around 30 is good), go into survival mode (/gamemode survival) and chase them around! You'll see they replenish the forest floor!

When you're finished, type the command /kill @e[type=ocelot] in chat.

PS: You might want to repeat technique 1 after this, as the ocelots have probably replaced all the podzol with dirt.

TECHNIQUE 4: Bombs Away

Hmm, now everything looks a bit flat… Why not use some creepers?
Fly above the terrain, and use this command every time you're flying over something a bit flat.

```
/summon creeper ~ ~ ~
{Invulnerable:1b,ExplosionRadius:3b,Fuse:30,ignited:1b}
```

You can change the underlined numbers to modify the explosion radius and fuse.

Picture 8: Creeper Bombs

You can also use automatic bombs… just summon a few bats using this command:

/summon bat ~ ~ ~ {Passengers:[{id:"minecraft:tnt",Fuse:50}]}

PS: It might better to summon them on the forest floor, because if you summon them in the sky, they won't blow up much.

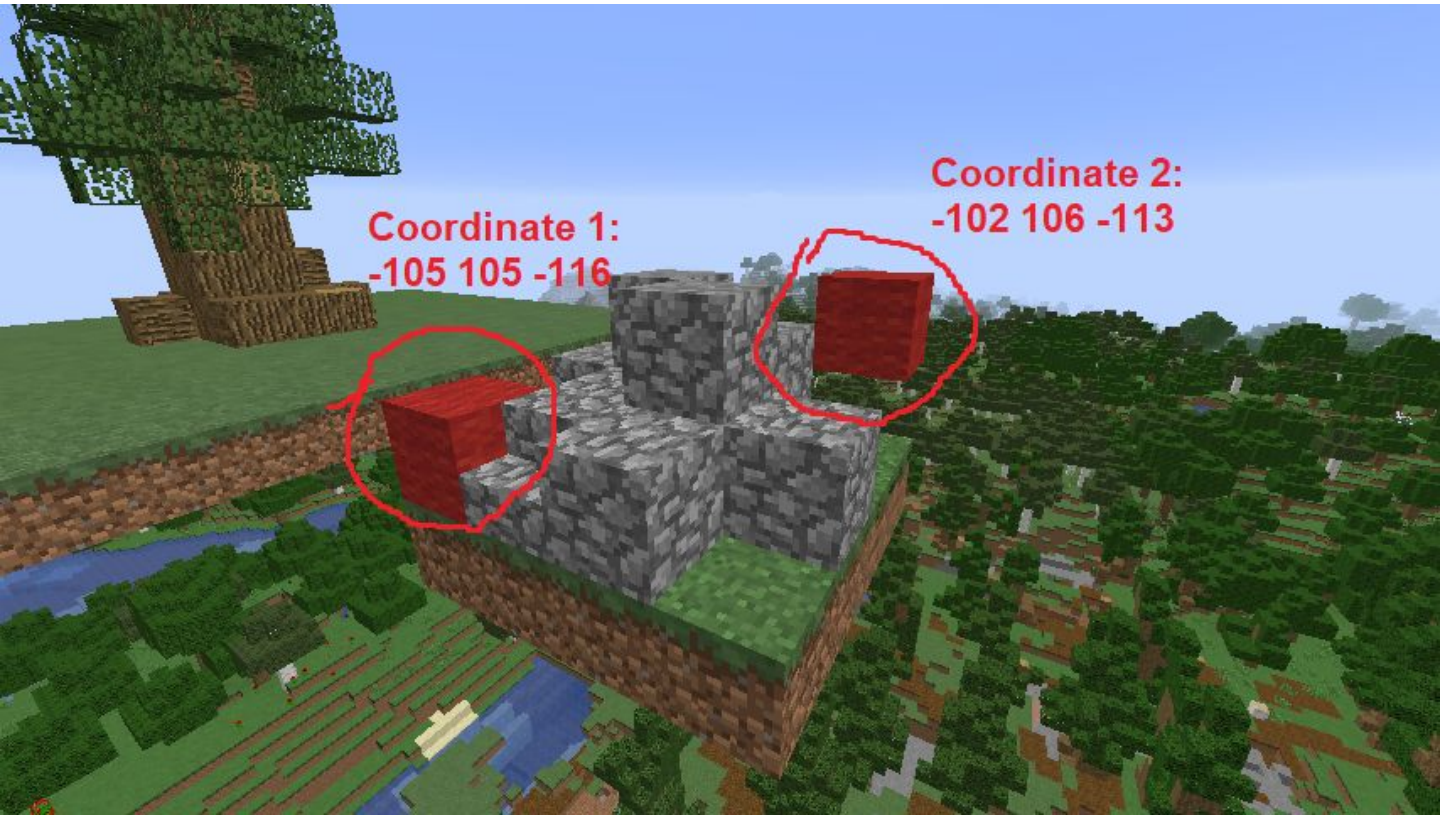
Picture 9: Bat Bomber

TECHNIQUE 5: Structures

You can also make a few small structures, such as bushes and rocks. To do this, first make a small structure/bush/rock/etc…

Picture 10: Small Rock

Next, note down the coordinates, like you did with the tree.



Picture 11: Note down coordinates

Remove the blocks you used to get the coordinates (here they are the two red wool blocks). Now, make a command chain in this order.

1. 1: /execute as @e[type=arrow,nbt={inGround:1b}] at @s run clone -105 105 -116 -102 106 -113 ~ ~ ~ replace force
   (replace the underlined numbers with the coordinates you noted down).
2. 2: /kill @e[type=arrow,nbt={inGround:1b}]

Voila once more! Every time you shoot an an arrow, a small structure will appear! You can make as many structures as you like! You can also make land appear to make hills! You can also spawn bushes on top of each other to make trees!

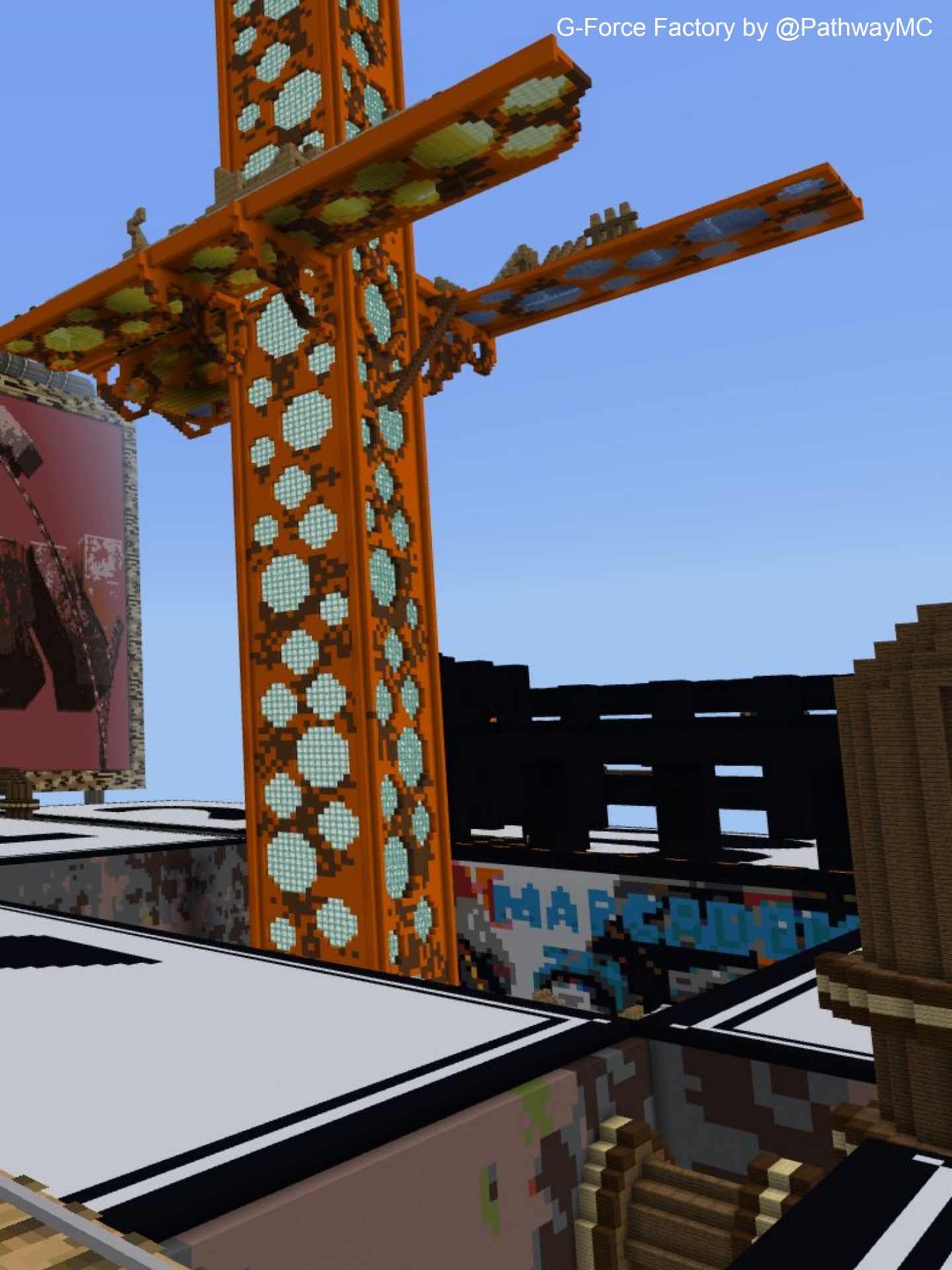
Pictures 12 and 13: Isn’t it beautiful?

CONCLUSION

Terraforming is a great way to make your map feel alive. And now it's easier to do than ever! All you have to do is remember two simple golden rules:

1. Spam podzol
2. Spam TNT

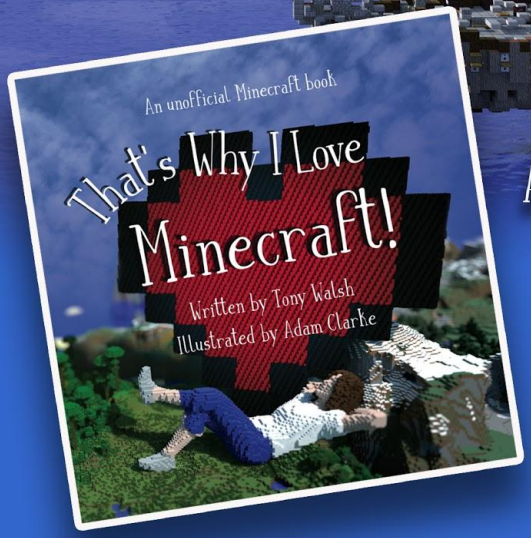
And that's how you turn an ugly default forest into a beautiful (if somewhat blown-up) terraformed forest.


G-Force Factory by @PathwayMC

"THAT'S WHY I LOVE MINECRAFT!"

It sums up beautifully what every Minecraft player feels and what anyone who is yet to play can never fully understand.

Stampy Cat



A stunning celebration of the video-game that is helping to build a better world

Written by Tony Walsh
Illustrated by Adam Clarke

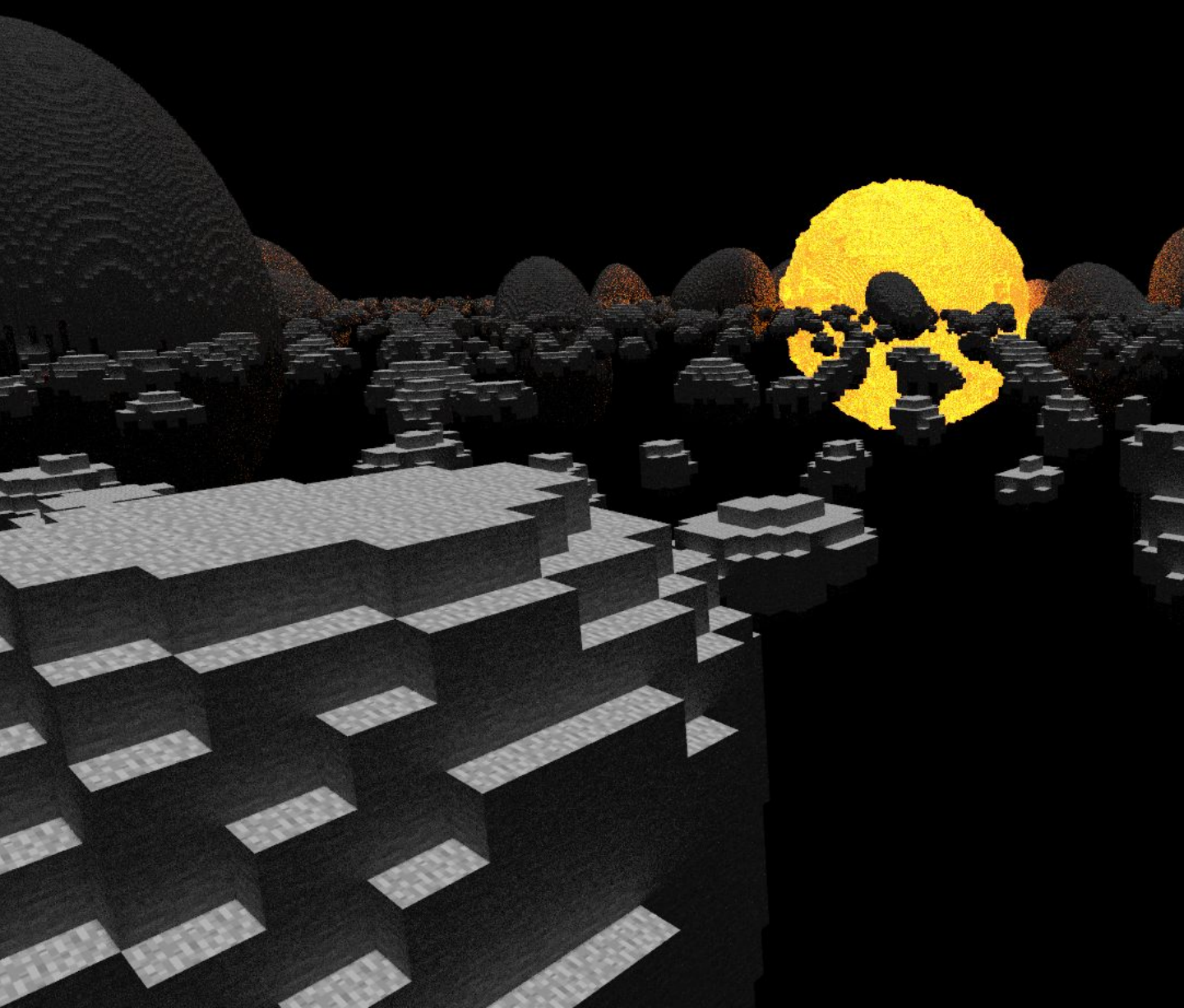
Published by The Wizard and The Wyld

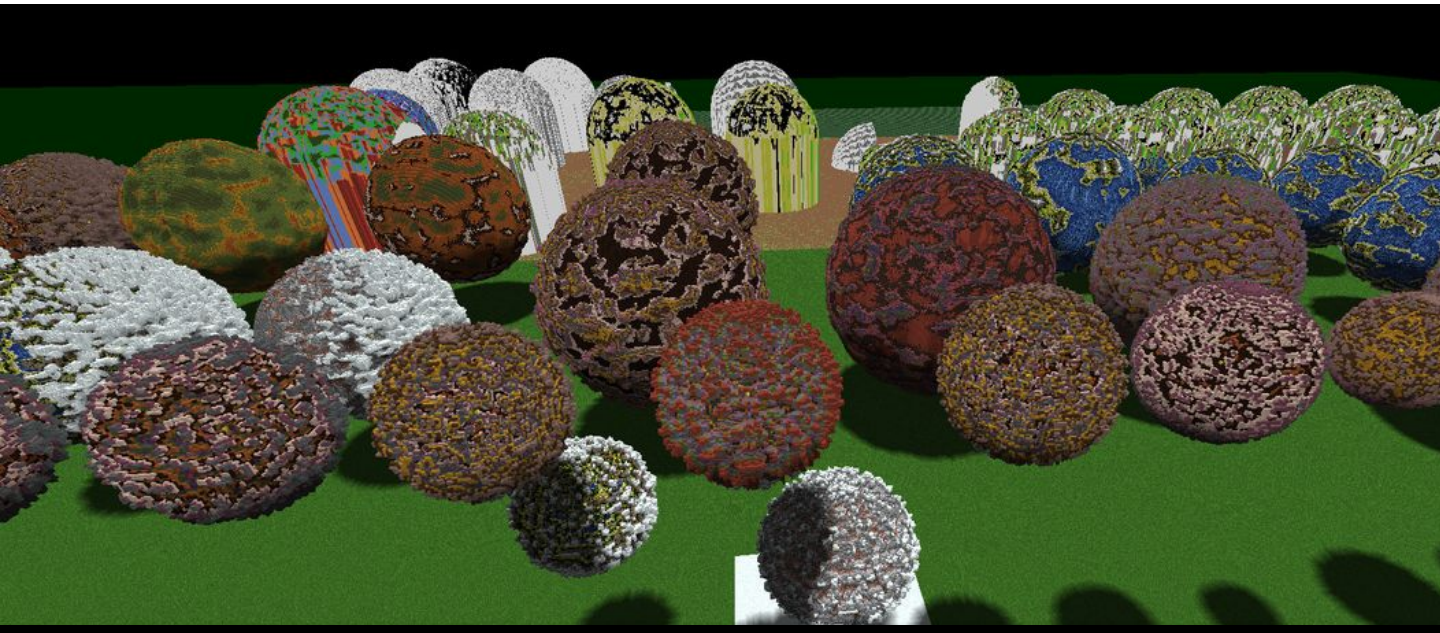
Paperback £10.00

www.why-i-love-minecraft.com

The Wizard and the Wyld

Procedurally Generated Environments





Procedural Planets

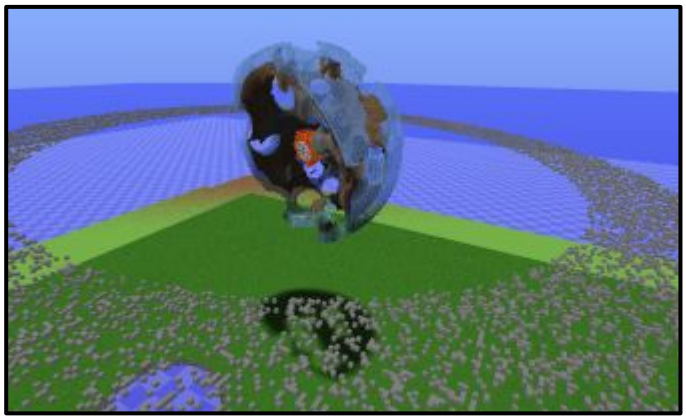
There is something absolutely magical about spherical objects in Minecraft. Whether it is through the projection of minecart riding falling sand or block-wearing armour stands, spheres are a great way to add interest to a map.

It can be very tricky to get them right when building by hand, and so here is a method that involves a bit of scripting in MCEdit and which is available for non-commercial use by anyone who is looking for popping planets into your map.

I have developed a couple of approaches to generating spherical planets. The first is through the use of an MCEdit filter called “THAT’S NO MOON”, available for download for non-commercial use here:

http://www.brightmoore.net/mcedit-filters-1/thatsnomoon

THAT’S NO MOON creates a sphere in the selected area of the world which has layers of material at different depths. Then, if the map maker chooses, the sphere can be eroded by pock-marked craters. It is also possible to have half the planet erased to create a sculpture that looks like a waxing or waning moon. Finally there is an option for a tilted debris ring.

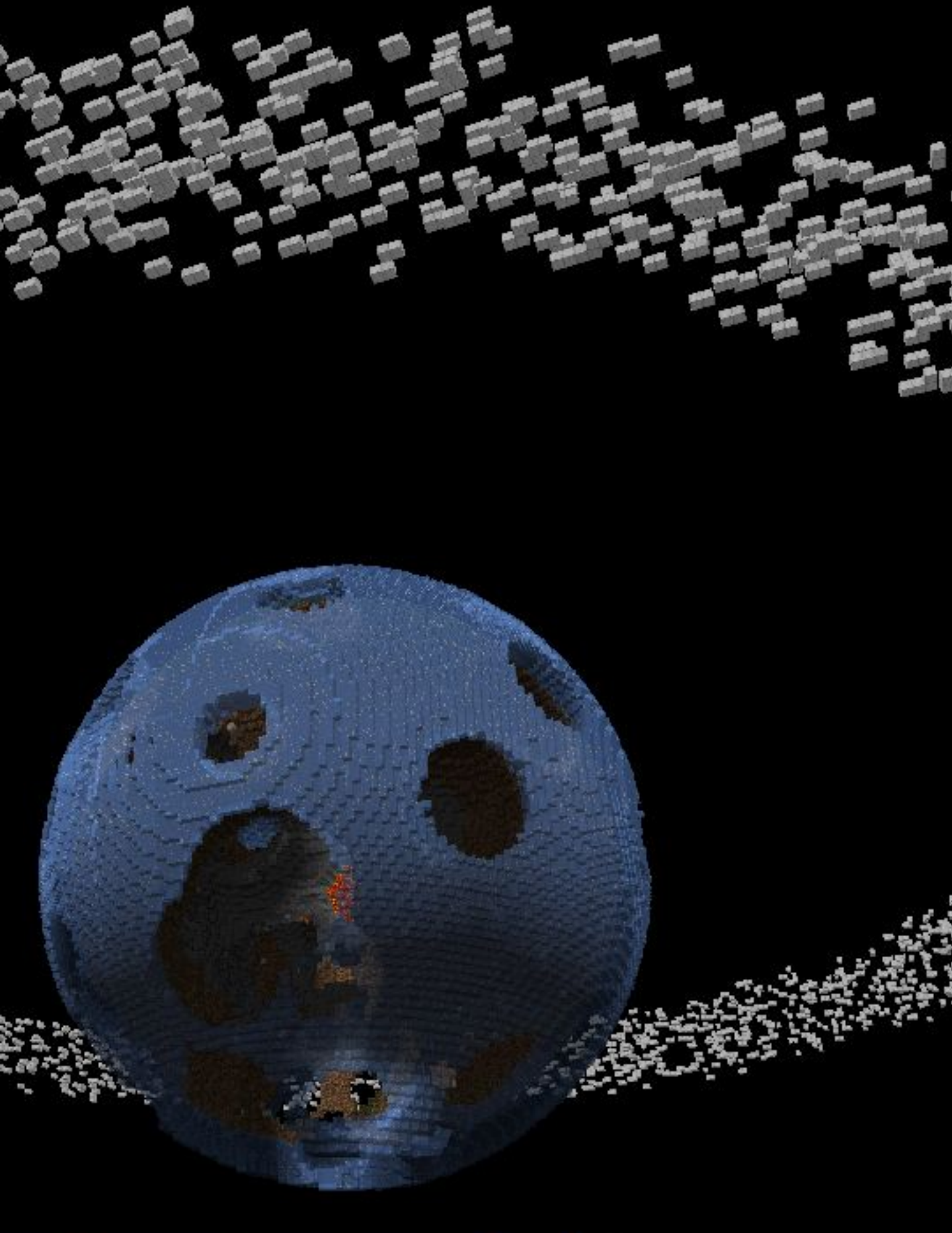


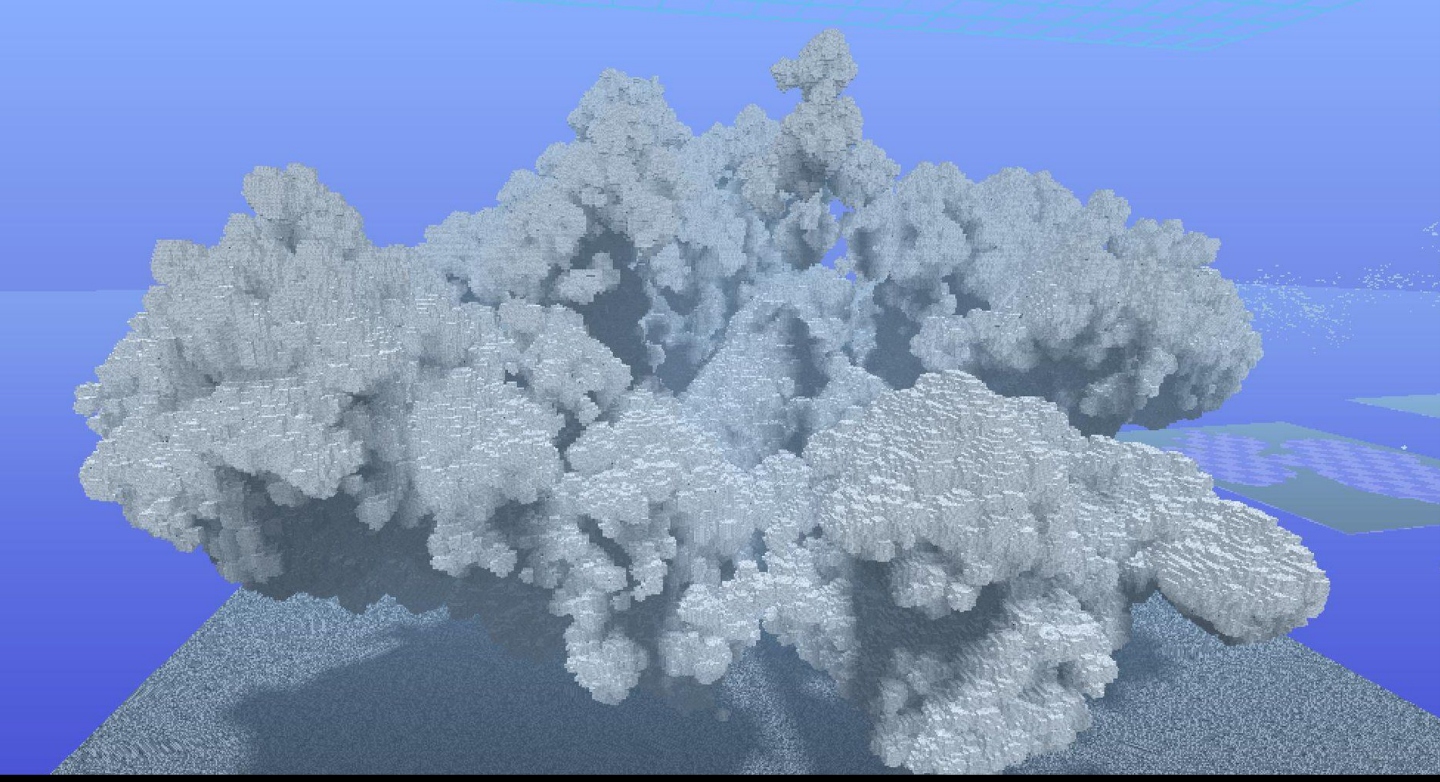
The second method is through the procedures implemented in the filter called “WORLD FOUNDRY”, which is available for non-commercial use here:

http://www.brightmoore.net/mcedit-filters-1/worldfoundry

The method used here is to create a triangular mesh and slightly randomly offset all the vertices to create different heights in the surface of the sphere. The materials chosen to map each height of the world are placed based on how far away the block position is from the centre of the planet. In this way, mountainous landscapes can rise above an ocean, or deep valleys.

The WORLD FOUNDRY filter has been used successfully in Jigarbov’s successful Marketplace CTM maps called “SOLAR” and “SOLAR 2”. It is also a focus of a large build called the Library by DestructiveBurn (shown at left).





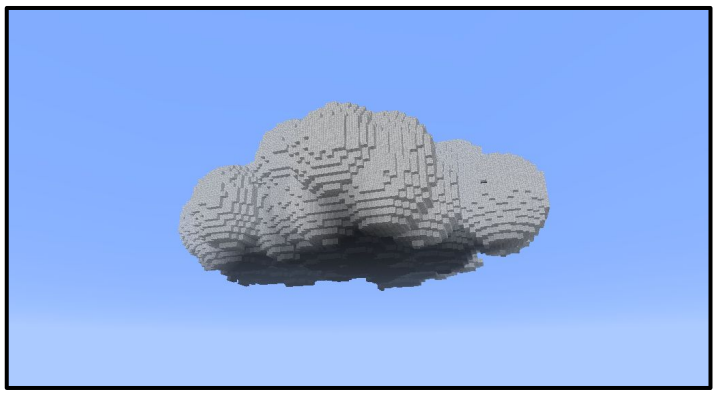
Clouds

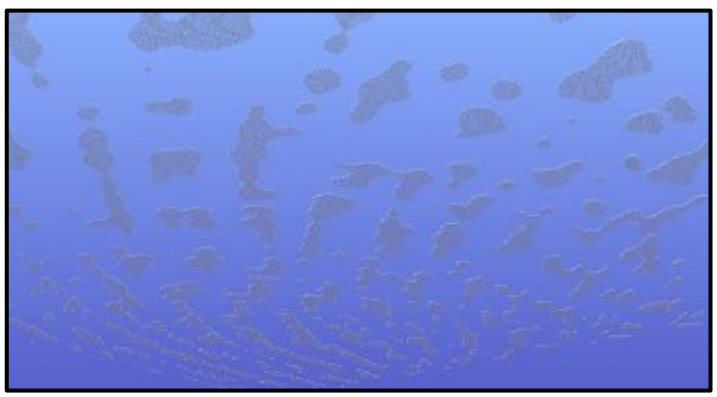
Sometimes you want to take your player somewhere unusual, and a cloud environment can be a perfect place to explore grand castles, air-borne monsters, or just soar with an elytra.

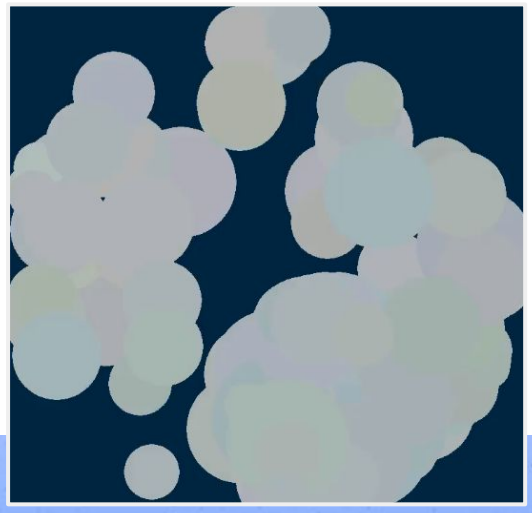
Builds involving clouds in Minecraft can be achieved using a variety of methods. One simple approach is to pack spheres of various sizes together to make a dense cluster of wool or glass.

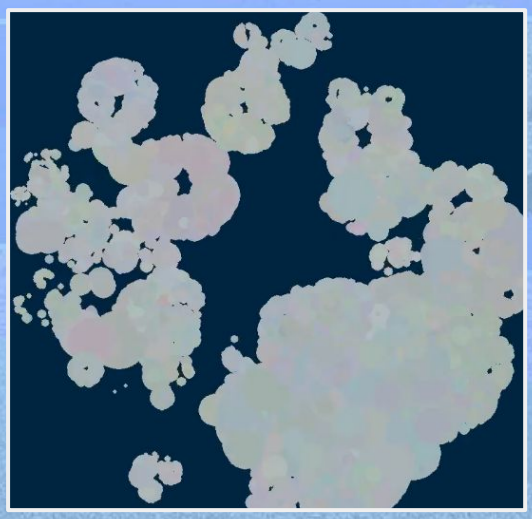
Another approach, suitable for high up whispy cloud layers, is to selectively plot the amplitudes of an interference field. This involves slightly more trigonometry than with spheres, so you can use the “CLOUDS” filter from here if you are working on a non-commercial project:

http://www.brightmoore.net/mcedit-filters-1/clouds





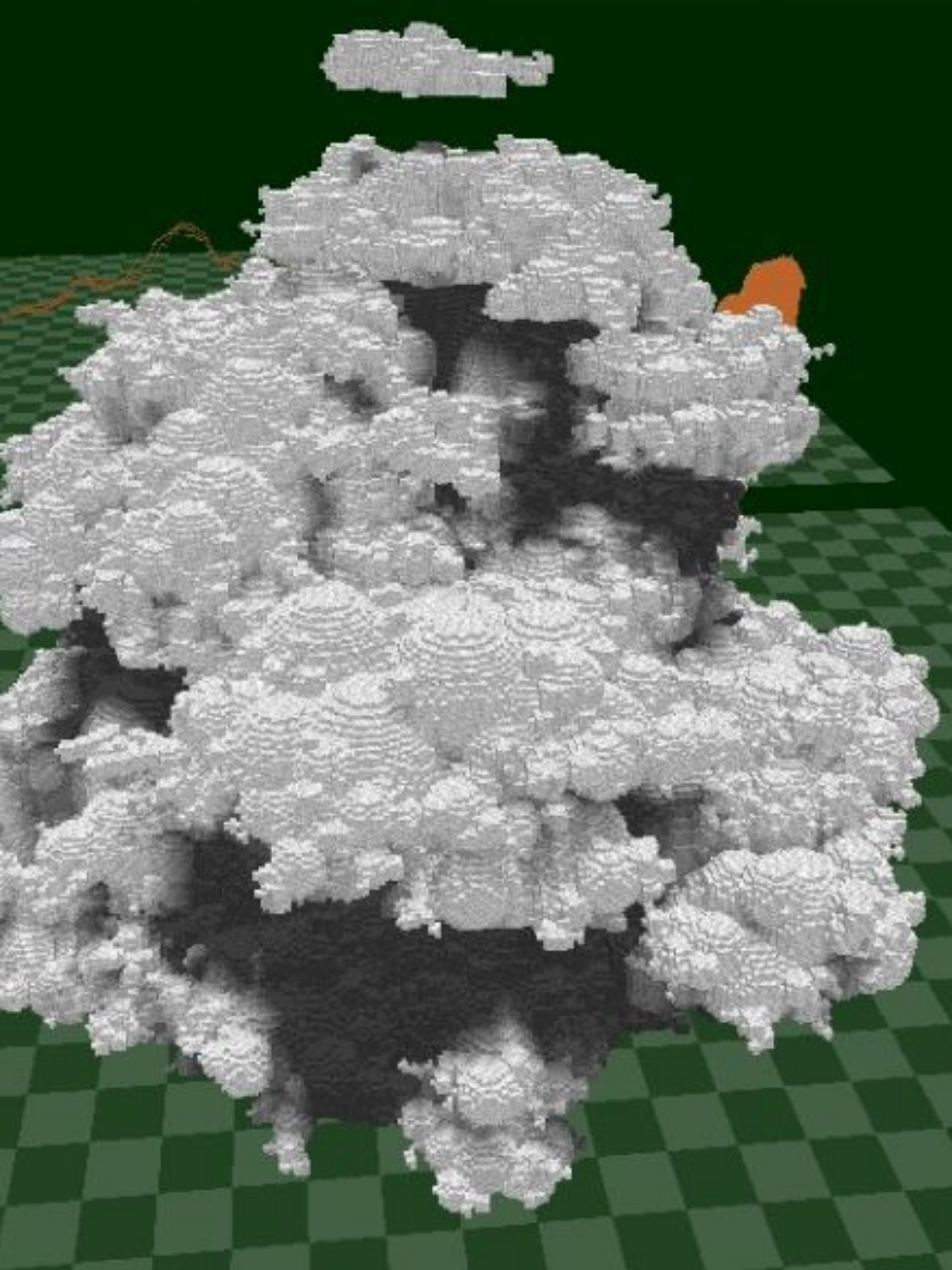




Fractal Clouds

Clouds can also lend themselves to procedural generation using a fractal approach where the volume of a cloud is iteratively decomposed into sub-spheres until the desired resolution is achieved. This approach makes beautifully dense and intricate cloud masses.

The process is illustrated at left with the top-most image showing an initial collection of overlapping circles. The second and third image replace each circle with smaller circles within the bounds of the parent circle. Over time this produces the scattered cluster look familiar to sky watchers in our real world.




Generative
Design in Minecraft

Generative Design in Minecraft Competition

A participant's perspective

2018 was a crazy year for Minecraft projects. Microsoft and Mojang hit their groove, and waves of gaming platform integration and feature releases were washing through the map making and player community.

Unexpectedly, at the start of the year, New York University burst on the scene with a community challenge to create a practical village in Minecraft worlds through the use of procedural generation or artificial intelligence methods. "GDMC" stands for "Generative Design in MineCraft", in case you were wondering. You can read all about the challenge here: http://gendesignmc.engineering.nyu.edu/

My response to this was: "this is great! I want in!"

I have been an avid follower of all things computer-generative ever since I first picked up James Gleick's excellent book "Chaos!" as a teenager. The prospect for computing to create new stuff that's never been seen before is exciting, and the tools and techniques to do it are fascinating to study.

The other major influence in my interest is a quirky novel about 'computer contact with a 2D world' called "The Planiverse" by A. K. Dewdney. In this story, a species of flatworld creatures exists and we join one of them on their journey through their world and beyond.

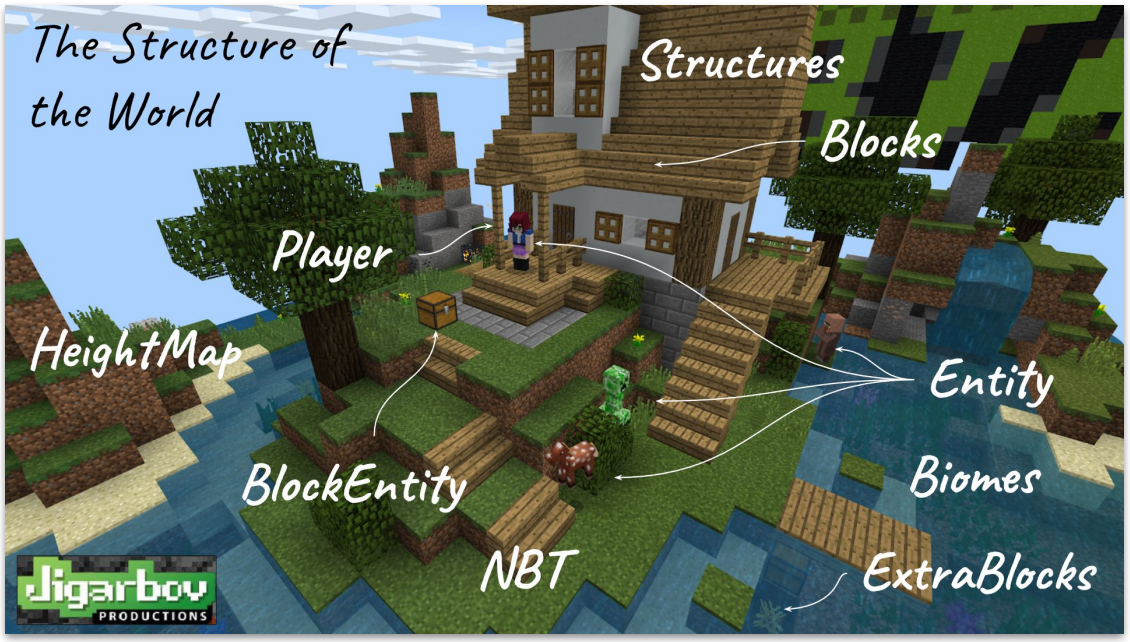
I thought about approaches for pretty much the entire competition time, then jumped in to write some code just as the deadline approached (FYI this wasn't the best approach). In my defense, there's a lot of projects underway all the time so prioritisation of the fun stuff (like this competition) suffers.

Here is a short description of what I did up until the point of submission (spoiler: I made it on time). If you're interested in procedural methods, Minecraft content creation, and Python programming then read on. If you just want to see how things worked out, check out the GDMC wrap-up presentation, and my final submission.

Thoughts on approach

From my reading of the competition, the challenge was to respond to an arbitrary section of landscape with an in-game village that has in-game playability characteristics. The framework provided was a custom version of the popular MCEdit Unified tool which I often liken to a CAD package for Minecraft worlds. It has scripting support for batch-style jobs that modify the Minecraft world. It also has an interactive interface for touch-ups and real-time ad-hoc editing.

MCEdit has a good capability for working with the 1 metre x 1 metre blocks that make up a world. It has low-level APIs for working with the hidden meta-data, called NBT, that describes chests and their contents, as well as ingame characters like Villagers. MCEdit also provides a way to work with aggregates of blocks and NBT through the use of 'schematics'. Schematics are pre-built structures that can be placed around the landscape in an editing session.



The structure of a Minecraft world, adapted from the Minecon Earth 2018 MCEdit Community Panel

The core problem in this challenge is attaining a holistic understanding of the world so that decisions can be made about designing where to place things. The supplied framework for the first challenge leaves it up to the participant to build an appropriate framework to respond to the challenge. The evaluation is based on the results, however, so whatever ideas I had as to how to apply holistic thinking to the problem were not really important to the competition ranking. Still - that's where my interest lies, so that's what I'm going to write about in this article.

Artificial Intelligence is a field that involves a lot of choice of methods to get machines acting in a way where decisions can be made. Not all methods are appropriate for all tasks. Consider Machine Learning, which is the discipline of generating statistical models that support the evaluation and production of assets by examining features of things that are 'like' what we're after while also avoiding features that are not. Choosing a machine learning approach can be complicated by a lack of available large and good training sets, for example. You can see some of the problems with machine learning against small training sets in this example using Minecraft's skin system here:

Machine learning with Minecraft skins

Traditional approaches to design problems in computing involve evaluating criteria and then responding with a procedure that takes input to create a result. Some systems are iterative and involve repeated application of 'rules' to generate an outcome. My work on this challenge was 'procedural' and not-iterative in nature, and not statistical/learning based. My approach was to evaluate the landscape and then build something in response to what was found.

The first pass: "Find places to build out"

I decided that regardless of whatever I would eventually build, it would be important to have an understanding of the characteristics of the landscape at different scales so I could make decisions about how to respond to it. If there was a lake of lava then I'd be well-placed to steer clear of it with a flammable wooden house.

To do this, I established a height map by scanning from the sky to the surface of the landscape and then used a set of grid systems across the heightmap to work out what the 'roughness' of the terrain was, among other things like what materials were present.

The height map is a 2D row/column structure of height values generated through a brute-force looping process.

Once the landscape height was profiled, a series of doubling grids from size 4x4 and up were applied to it and each cell was examined to determine the roughness and the materials present on the surface:

Once there was some landscape information to examine in the cells, placement of structures to conform to the landscape properties was (in theory) a matching exercise of generating candidate locations and selecting one for each building.

This approach supports individual structure placement, but does not deal with what the structure should be and how it should participate in the village as a whole. This approach was explored by placing 'marker tape' box outlines on the plots that would be selected to build on based on the 'flatness' of the terrain cell, as determined by the max and min heights within the cell.

Terrain profiles as cells at different scales

The drawback of this approach is that it doesn't offer any clues as to whether the terrain between selected plots is passable (or can be easily adjusted to be passable) for roads and pathways, making it unclear whether the resultant village would 'look' right. At a high level, my plan for generating a village was:

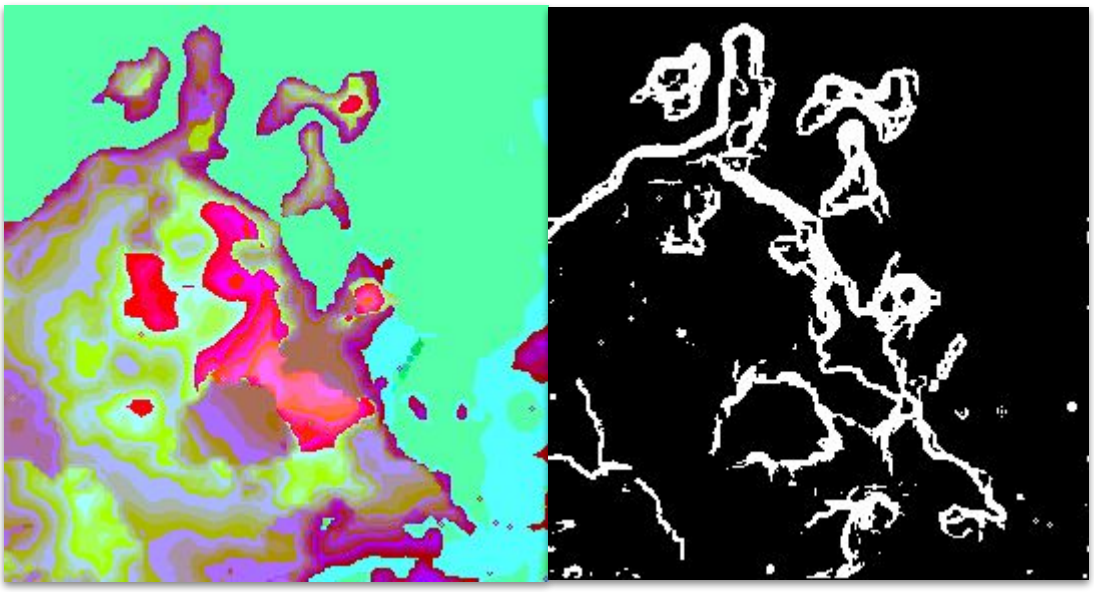
1. generate networks representing buildings in the settlement.
2. the nodes are areas/rooms, each with a local generator
3. merge the networks into a settlement network
4. position the nodes on the landscape, using rules for how close areas should be
5. Render.

Note that in thinking item 4 was the more difficult and interesting task, I focussed exclusively on understanding it and solving it first.

Ultimately I did not leave enough time to fully explore this idea, except it did drive the next revised approach.

The second pass: "Gently sloping pathways"

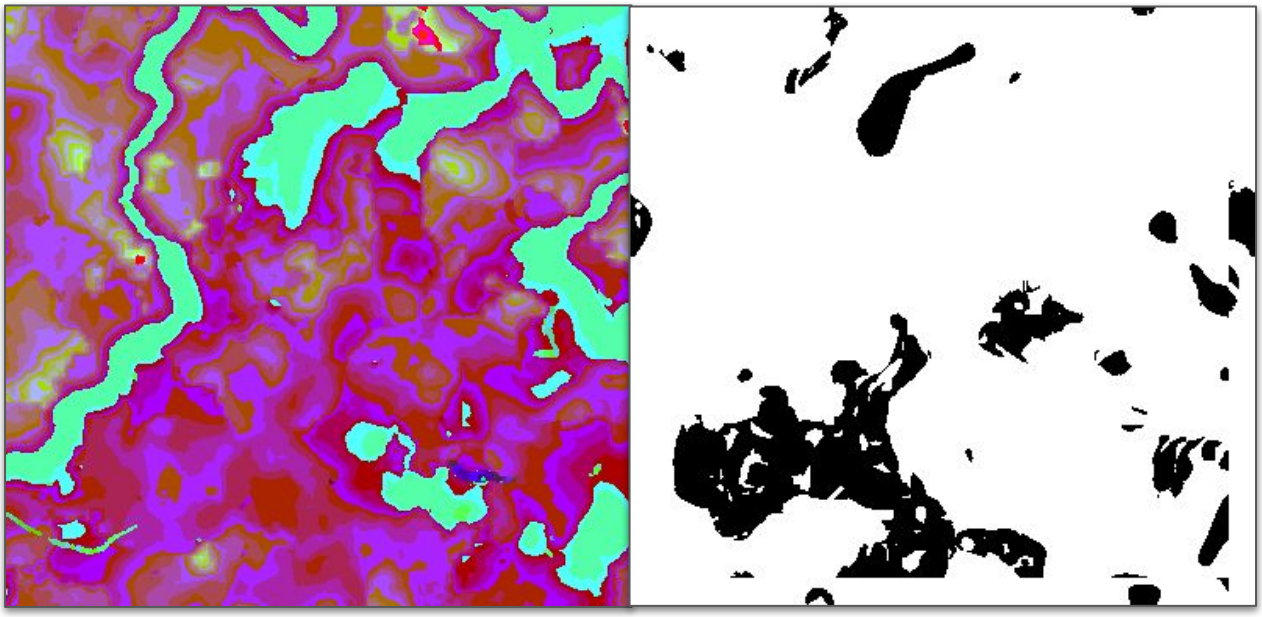
If I was going to tease out information from the landscape about how areas were connected, I figured the height map was probably a good place to start because by examining the way that block heights differ I could work out if the slope of the terrain was too aggressive to walk and build across. The other useful thing about a height map is that it is two-dimensional and so, in theory, the connectivity between points on the map can be determined using standard well-understood pathing methods. I used PyGame as the library to work with image representations of the landscape because MCEdit Unified, which is the supplied framework, bundles it in for its own use:



Height map (left) and derived edge map (right) showing connected areas of gentle sloped terrain

Edge detection was done simply by calculating whether the difference between two heights was excessive, which for a Minecraft player is the unjumpable height difference of 2 blocks, and also by projecting this difference out far enough that there is sufficient area marked as 'near the edges' that would allow a building to be safely placed anywhere there was a 'flat' pixel in the edge map:

As for buildings, finding a candidate place to build on the edge map then became a problem of looking for areas of the required size that did not have an edge marked as running through it. This was simplified by marking areas close to to edge as being non-flat so they would not be selected as the origin of a build location.



To simplify selection of building sites (black, right), the edge detection (white, right) is expanded to prevent the building dimensions falling close to an edge

The mad rush to finish

By this time I was quite please with the ideas and simplicity of implementation, particularly since image processing methods are a mainstay of 'real' AI projects. I felt this was a reasonably good way to find legal places to place buildings.

With scant hours until submission time I made the call that I'd be back for round 2, so solving this placement problem would be good enough for the first competition. I turned to a building generator I had lying around from earlier work with the Blockworks team on creating the greater London of the past and re-used my building generator by plugging it in once a suitable plot for building on was selected.

As a result my submission was limited to buildings placed according to the flatness of the terrain within the supplied landscape area. This is suggestive of a village in that there's more than one dwelling, but if I lived there I'd be looking for a bit more supporting infrastructure!

What came next

The competition has become a catalyst for developing a few of the ideas further. In particular, the theme of generation at different scales is an area I have continued to explore using a variety of techniques to generate large scale structures:

Final thoughts

I am grateful to the GDMC team for hosting and co-ordinating this competition as good governance and community management is always the hardest part of any project.

I believe that the next round of participants will benefit from being able to select from already existing frameworks that solve some of the more time-consuming problems. If a competition participant can select from a library of functions and then string them together to do the tedious grunt work then they can focus on solving the challenge in new and creative ways!

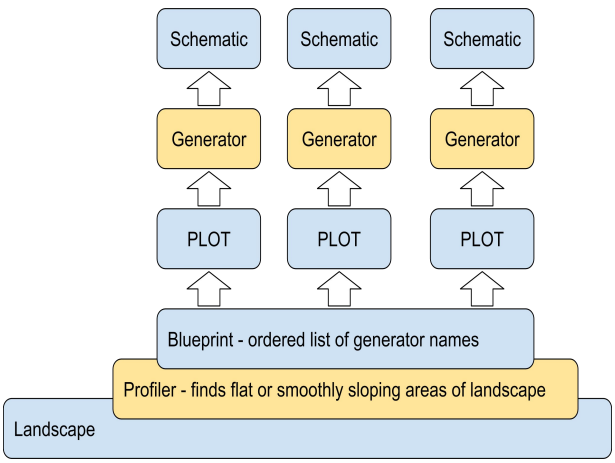
Procedural Settlement Generation in Minecraft: GDMC 2019

The Challenging Second Year

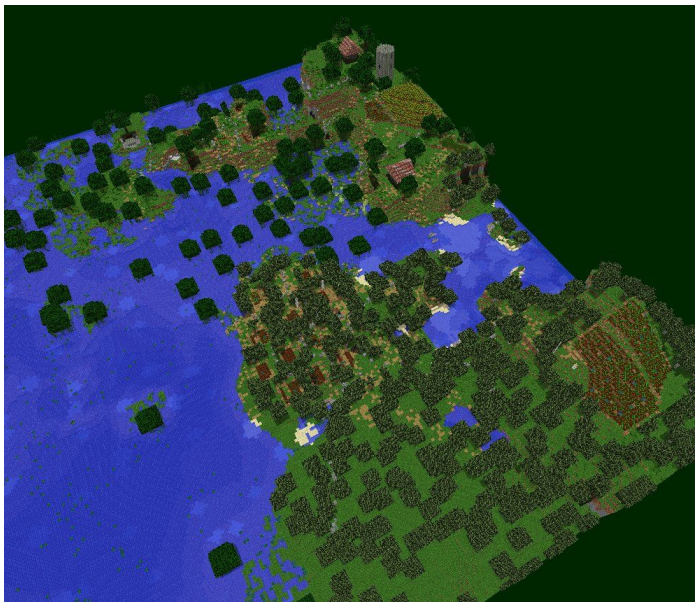
GDMC returned in 2019 with the settlement generation challenge. I chose to expand and extend on the work from the previous year to make an extensible procedural framework for many projects to use.

Framework

The approach I have taken to settlement generation involved identifying plots of existing landscape that can accept a building and then placing clusters of rooms on them. I populate the buildings with contents and then connect the buildings together with paths. The areas between the building plots are strung with low wall sections, and some farm land with various crop types are placed where possible. Building types include houses, towers, and a village square with a fountain.



Placement of generated schematics on the landscape requires some minor adjustments to parts of the landscape. Sometimes it is necessary to create sections of building foundations to prevent parts of the flooring floating over air.



Buildings

Buildings are a collection of rooms packed close to each other using a simple packing algorithm. The contents of the rooms are determined from a simple blueprint that associates the room type with a generation procedure. Orientation of blocks to line up with walls and avoid windows is done through the generator inspecting the area and making appropriate layout decisions.

Windows, lighting, and doors are punched through the assembly at intervals to provide passage for the inhabitants.

Carpet, crafting tables, furnaces, chests, beds and chairs are placed according to the room type to provide simple functional capabilities that an inhabitant might require.

Building placement

The algorithm profiles the landscape using a heightmapping approach. It then attempts to isolate areas of severe terrain height change, and then searches the remaining gently sloping areas for square sections that can be used to place buildings.

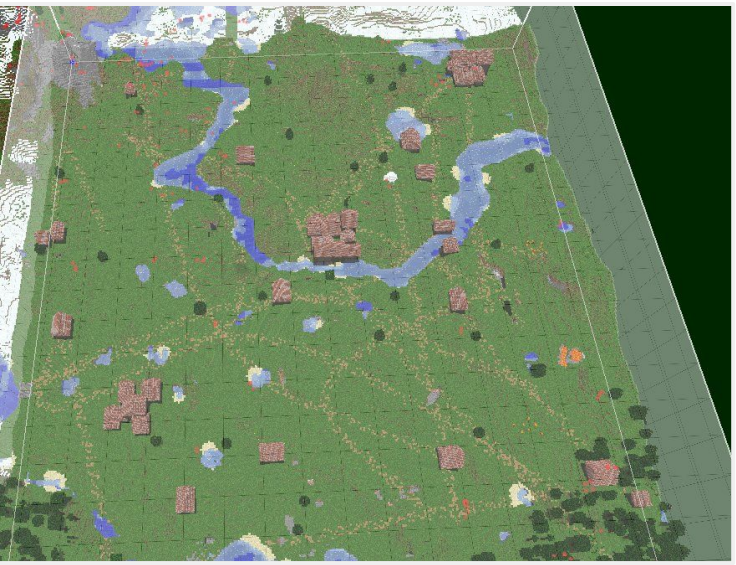
Towers

The tower variant of building has been introduced to provide a watchtower to the settlement.

It was primarily created to show how generators can be created using python and the features of the pymclevel library.

A worked example is in this thread:

https://twitter.com/TheWorldFoundry/status/1087616035992170496



Paths between plots

Each building plot centre is an anchor point that can be used to mark paths between the plots. By adjusting the number and weights of path plotting the settlement can get a ‘lived in’ feel.

Walls

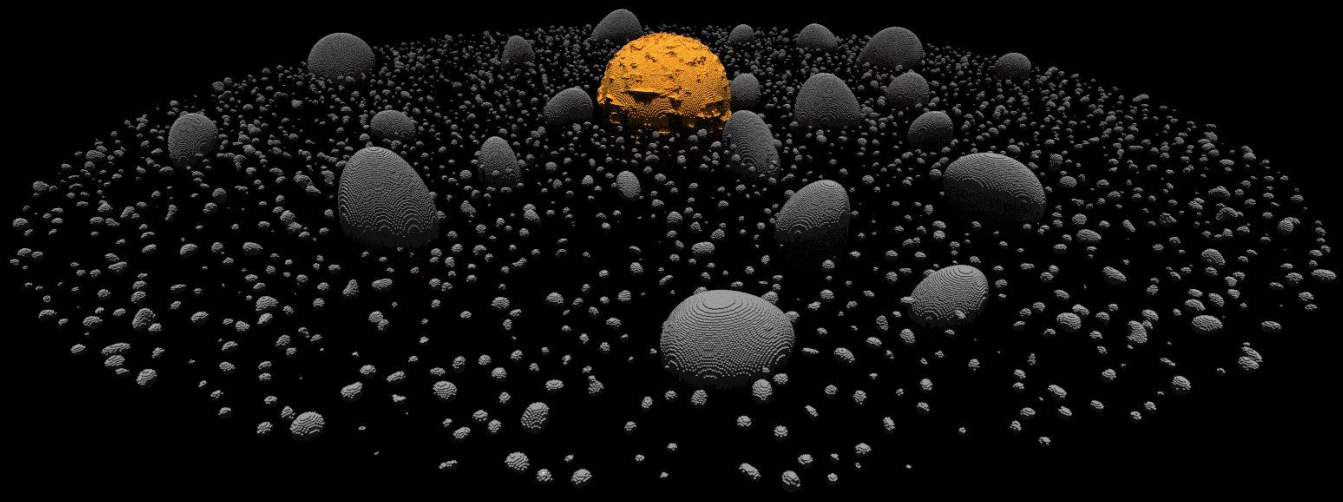
Establishing a voronoi graph for the plots in the settlement creates sections of landscape where walls can be placed.

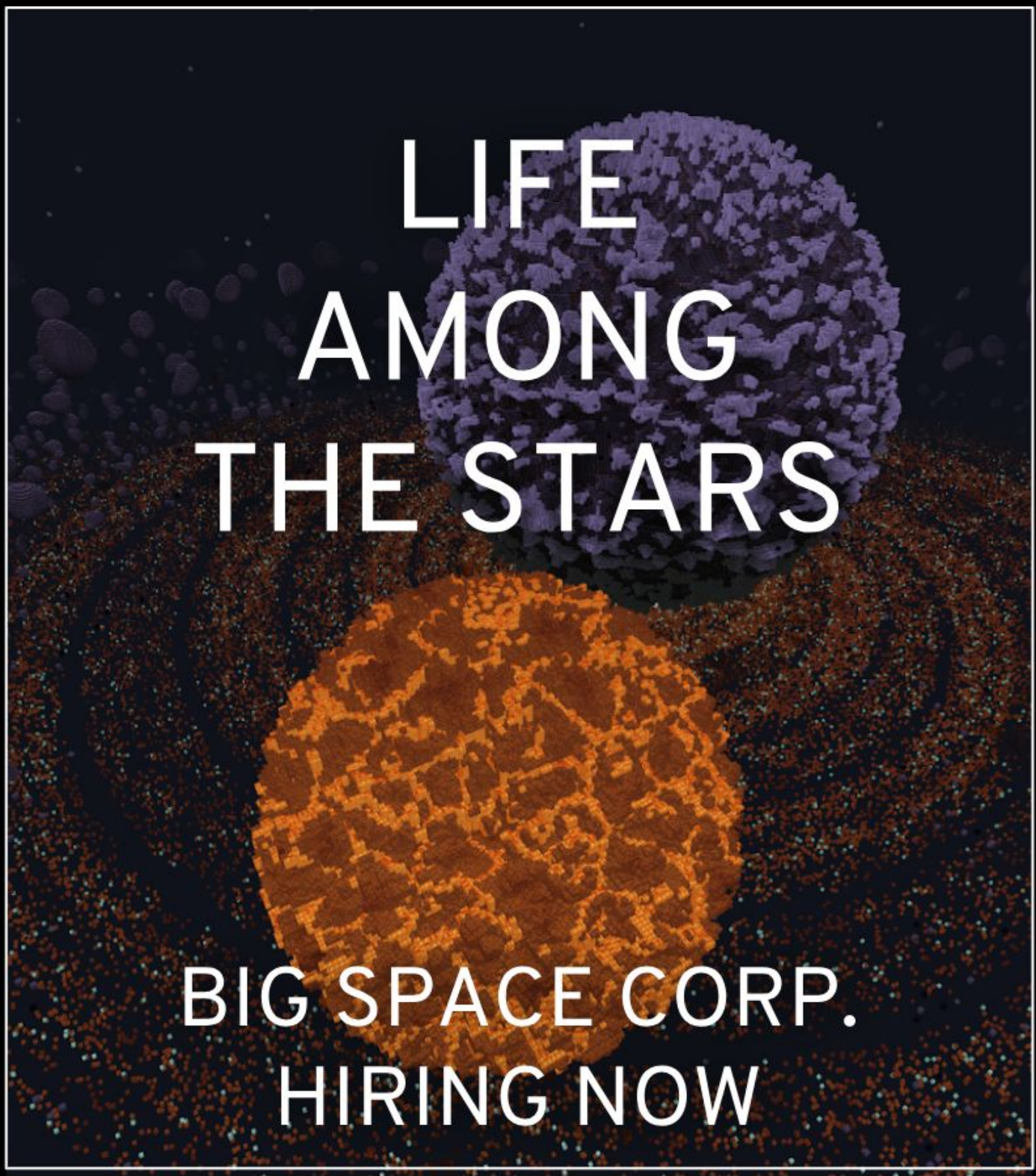
Farmland

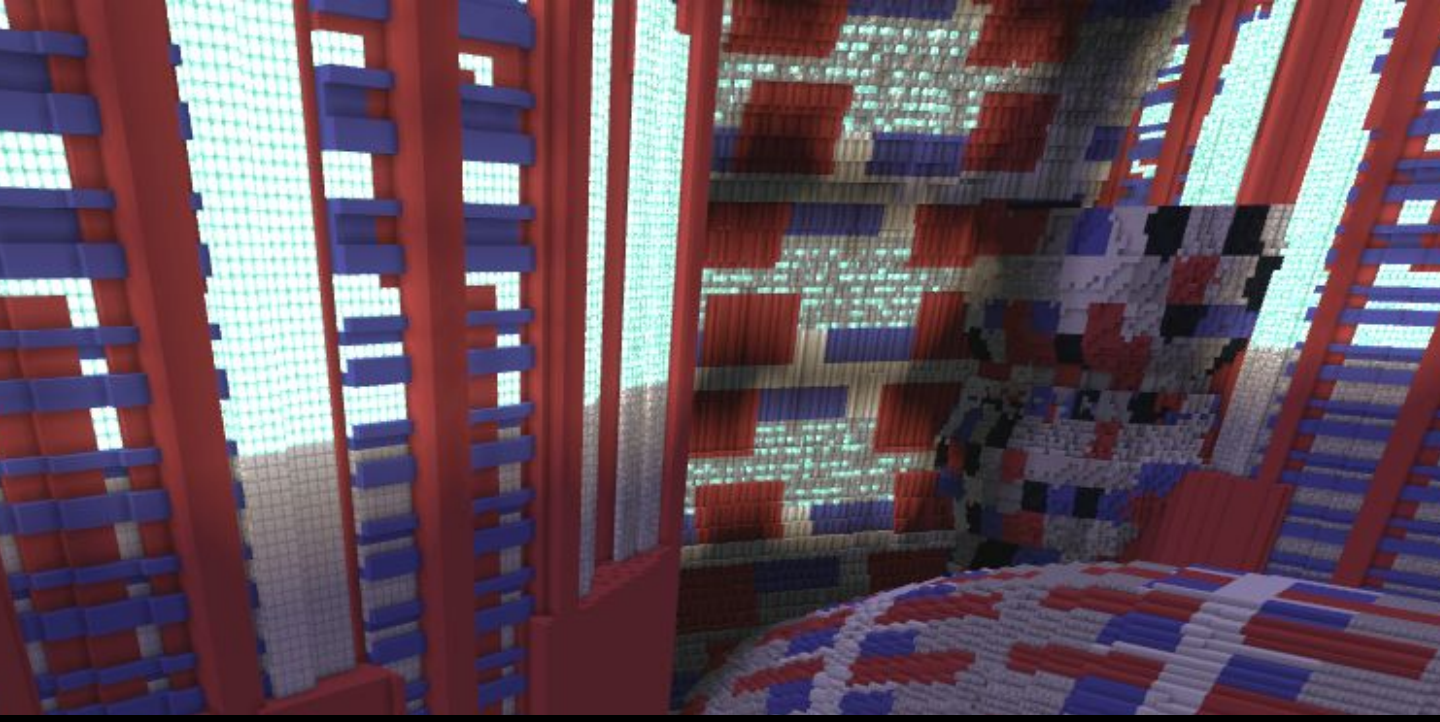
Adding a farmland generator was then as simple as creating a blueprint entry and writing the corresponding generator to create farmland, water source, fences and crops.

The Code

Check out the framework here: https://github.com/abrightmoore/ProceduralSettlementsInMinecraft

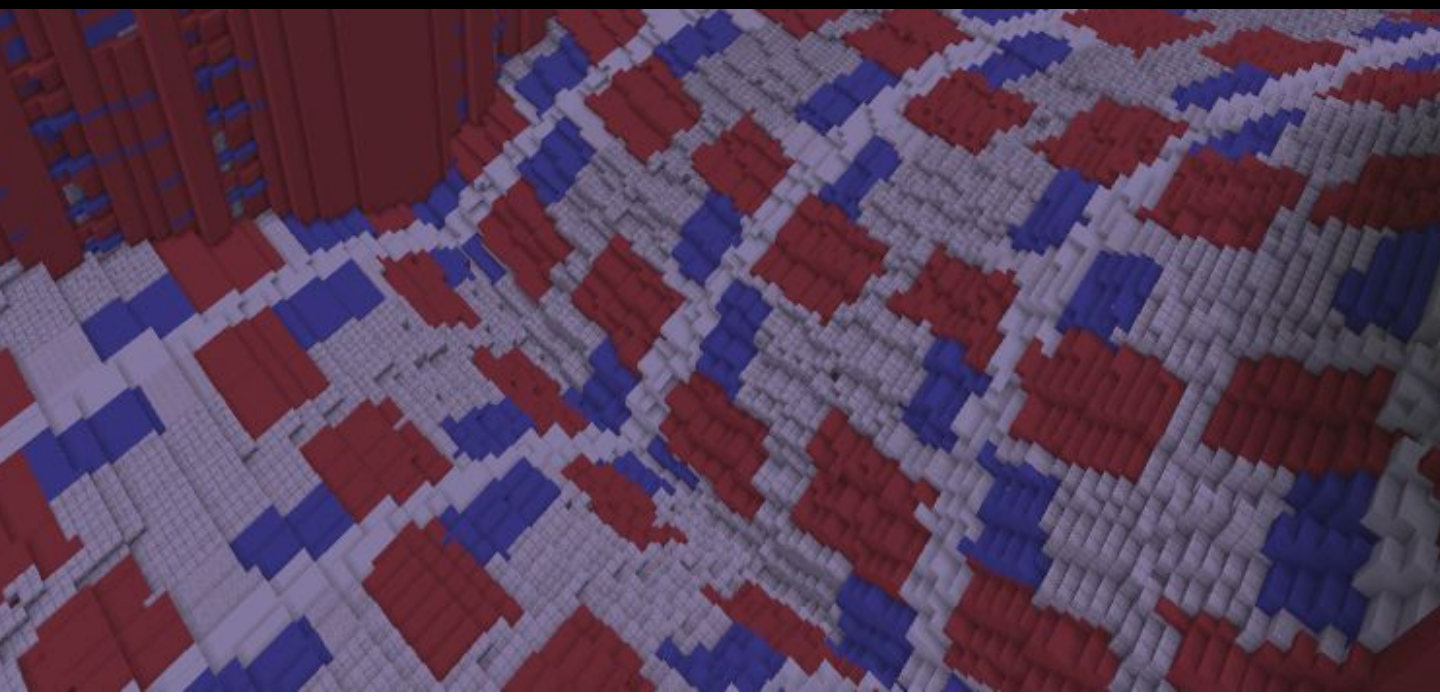



LIFE
AMONG
THE STARS
BIG SPACE CORP.
HIRING NOW



Space Station Z

Making a Marketplace Map



Disclaimer

Space Station Z is a Minecraft Marketplace world that is offered for sale on the Bedrock edition of the game. It is built by the Map Making Magazine editor's gaming content company called "The World Foundry". The World Foundry is an official Microsoft Partner and is a commercial entity. This article is intended as an informative review of the map design and build process and is not intended as a product promotion. You are not required to purchase Space Station Z to benefit from this article

The Scenario (Spoilers!)

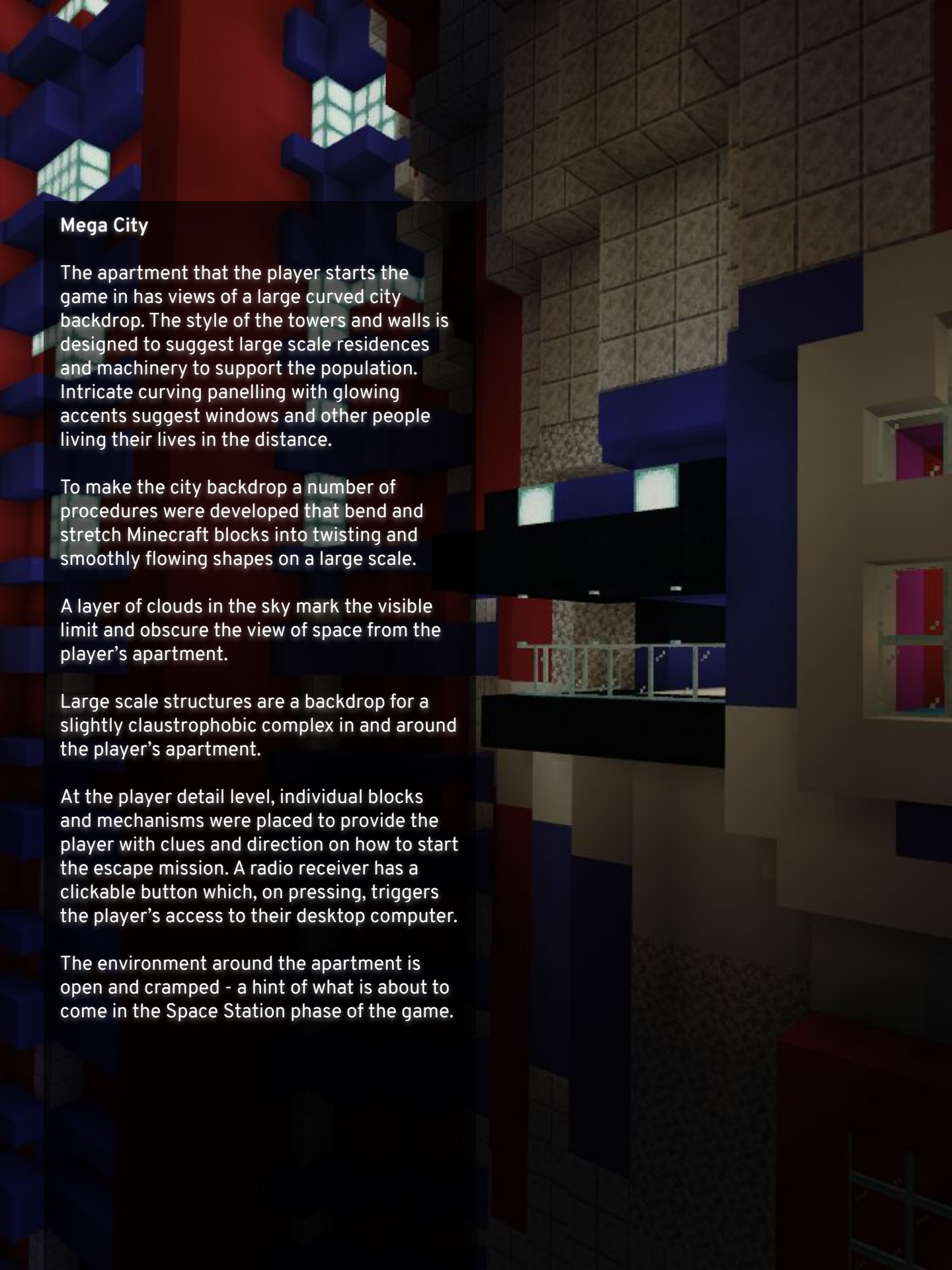
In Space Station Z, the player inhabits a part of a futuristic universe where life in the mega-city is repetitive and dull.

Seeking adventure, the player applies to join Big Space Corporation, a massive galaxy-spanning enterprise. Once accepted, the player is stationed on a Space Station in a strange solar system. The player is surprised one day while on duty by an emergency broadcast.

Through custom audio cues and environmental alerts the player discovers that people are becoming aggressive zombies. All seems lost and so the space station self-destruct is started, apparently to contain the contagion.

From this point the player has one main goal: escape the space station before it blows up.

To do this, they must navigate the station corridors, rooms, and service tunnels to reach the hangar. From there, they will equip an elytra flight suit and try to find a safe place to await rescue.



Mega City

The apartment that the player starts the game in has views of a large curved city backdrop. The style of the towers and walls is designed to suggest large scale residences and machinery to support the population. Intricate curving panelling with glowing accents suggest windows and other people living their lives in the distance.

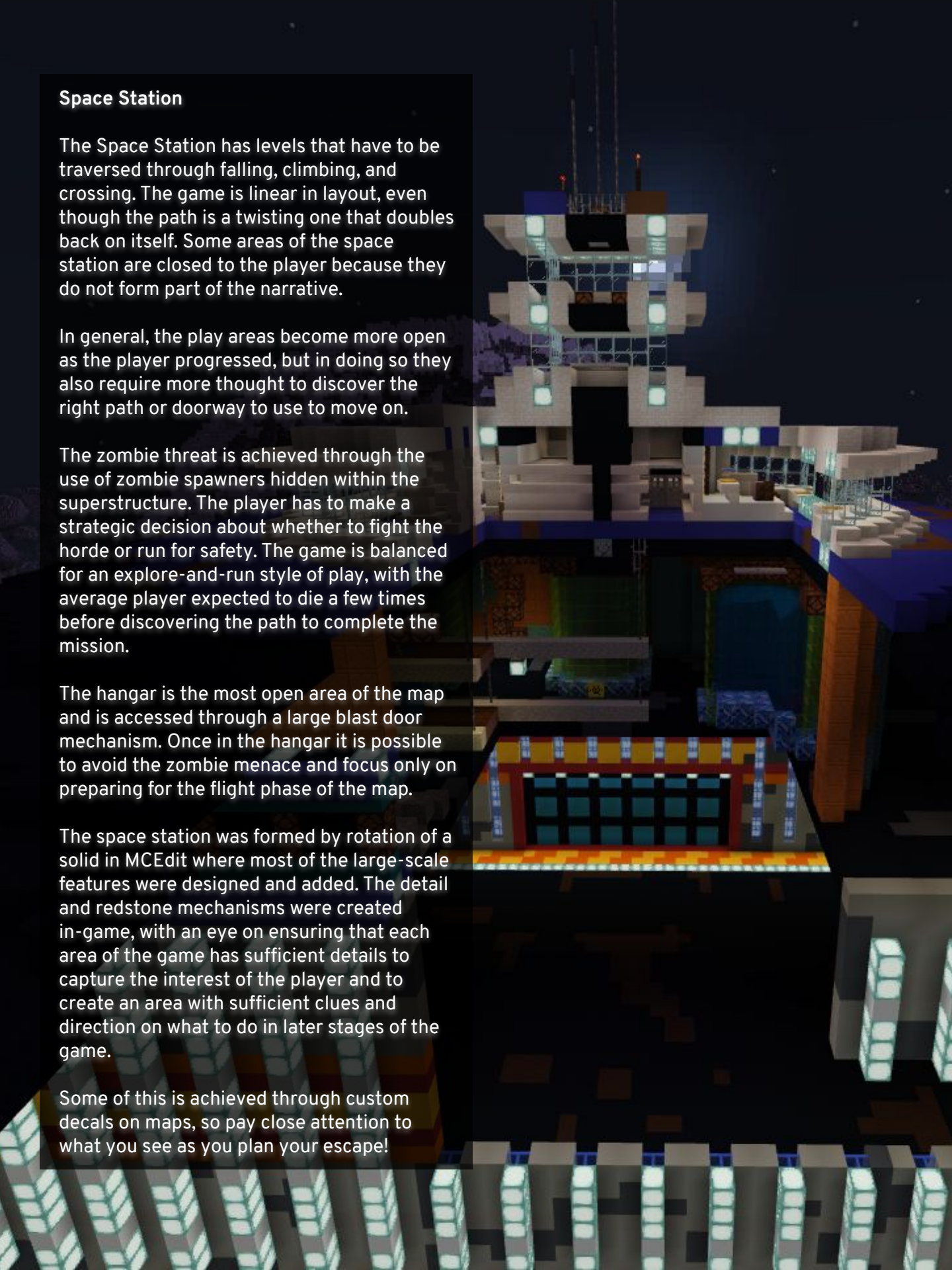
To make the city backdrop a number of procedures were developed that bend and stretch Minecraft blocks into twisting and smoothly flowing shapes on a large scale.

A layer of clouds in the sky mark the visible limit and obscure the view of space from the player's apartment.

Large scale structures are a backdrop for a slightly claustrophobic complex in and around the player's apartment.

At the player detail level, individual blocks and mechanisms were placed to provide the player with clues and direction on how to start the escape mission. A radio receiver has a clickable button which, on pressing, triggers the player's access to their desktop computer.

The environment around the apartment is open and cramped - a hint of what is about to come in the Space Station phase of the game.



Space Station

The Space Station has levels that have to be traversed through falling, climbing, and crossing. The game is linear in layout, even though the path is a twisting one that doubles back on itself. Some areas of the space station are closed to the player because they do not form part of the narrative.

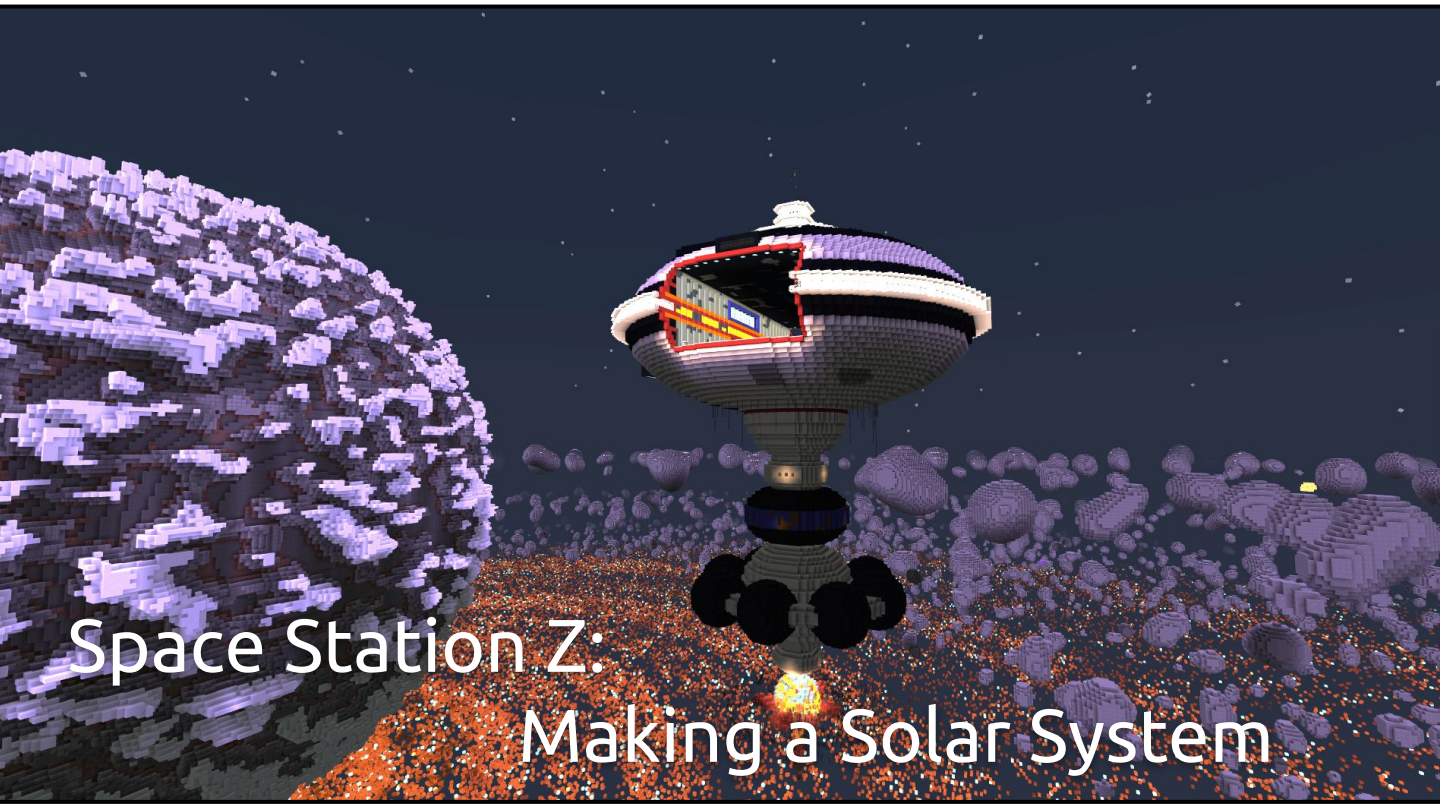
In general, the play areas become more open as the player progressed, but in doing so they also require more thought to discover the right path or doorway to use to move on.

The zombie threat is achieved through the use of zombie spawners hidden within the superstructure. The player has to make a strategic decision about whether to fight the horde or run for safety. The game is balanced for an explore-and-run style of play, with the average player expected to die a few times before discovering the path to complete the mission.

The hangar is the most open area of the map and is accessed through a large blast door mechanism. Once in the hangar it is possible to avoid the zombie menace and focus only on preparing for the flight phase of the map.

The space station was formed by rotation of a solid in MCEdit where most of the large-scale features were designed and added. The detail and redstone mechanisms were created in-game, with an eye on ensuring that each area of the game has sufficient details to capture the interest of the player and to create an area with sufficient clues and direction on what to do in later stages of the game.

Some of this is achieved through custom decals on maps, so pay close attention to what you see as you plan your escape!



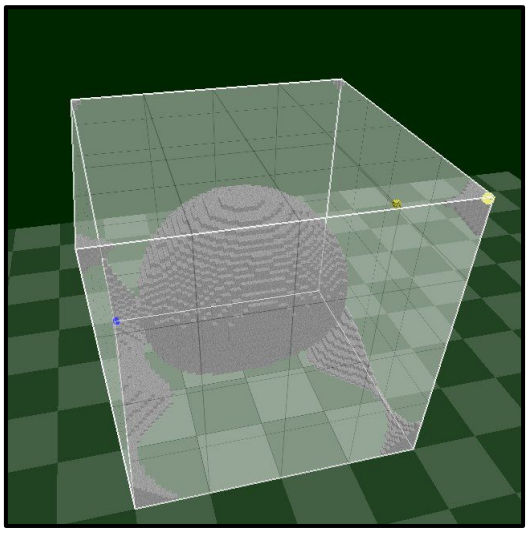
Space Station Z: Making a Solar System

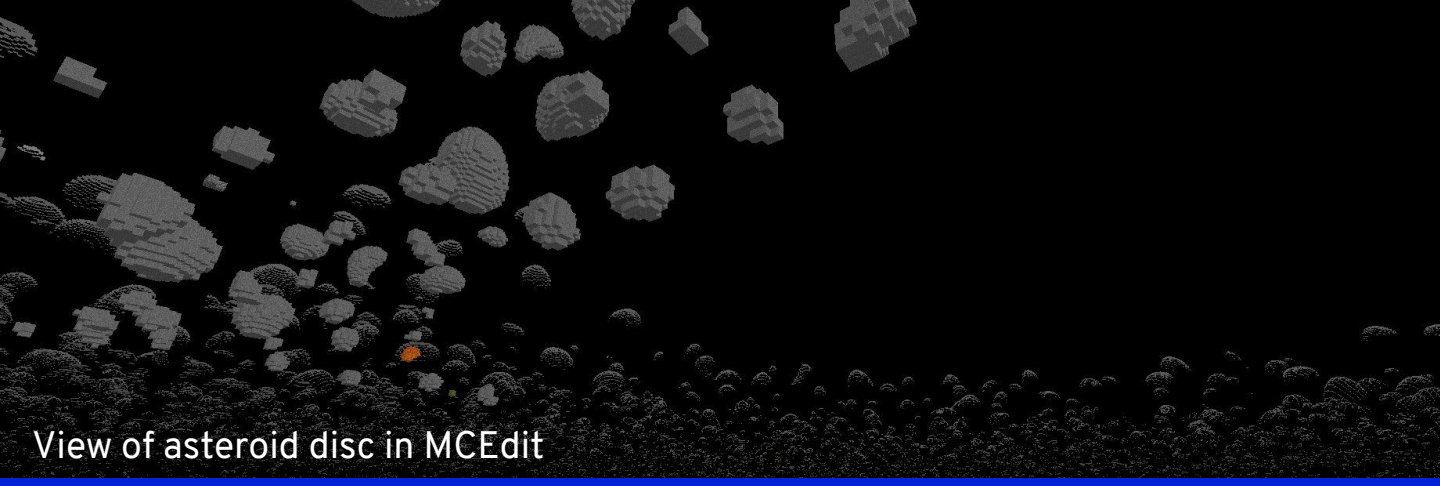
In the Marketplace escape map "Space Station Z" the player has to escape from the confines of a zombie-infected self-destructing space station using an elytra. They then must explore a vast asteroid-filled solar system to locate a damage space ship that can be used to survive until rescued.

A ticking clock measures how long each escape attempt takes, so you can try to beat your best score once you work out the right way to survive to the end of the mission.

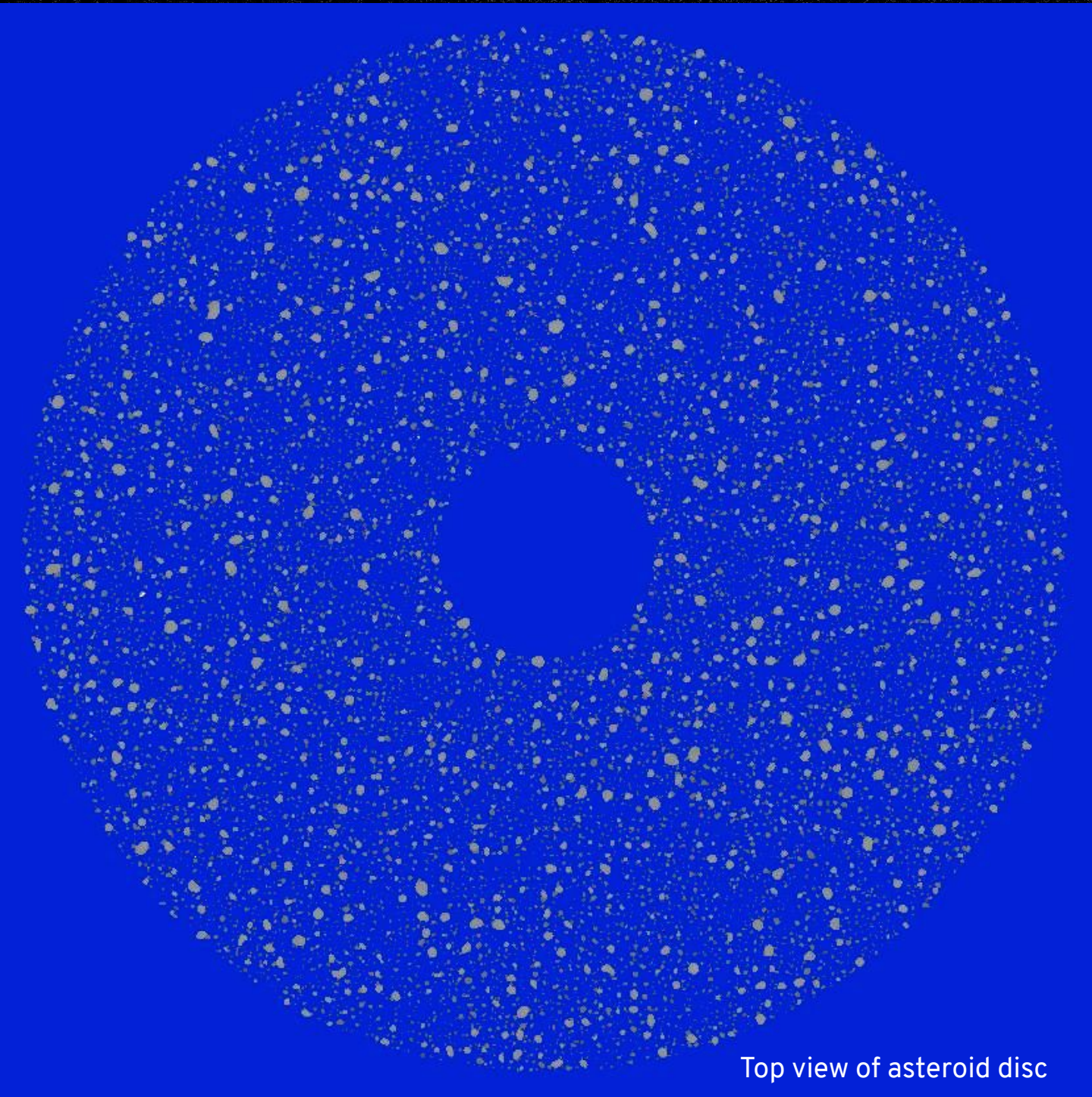
If all that sounds a bit crazy, you're probably right. A key part of the magic is a 3km x 3km asteroid field that hugs a planet and its hot star. The field contains around 20,000 spheroidal objects with various materials. All of these were procedurally created because it would have been massively impractical to create them all by hand. Let's take a look at how that was done.

Each asteroid in the solar system is formed by creating a density map within a randomly sized selection box. The density is based on randomly placed points which have a 'strength'. Then each block position within the box has a 'strength' calculated by summing all the contributions from every point in space, which falls away with distance from the point. This creates a unique potato-like shape for each asteroid.





View of asteroid disc in MCEdit



Top view of asteroid disc

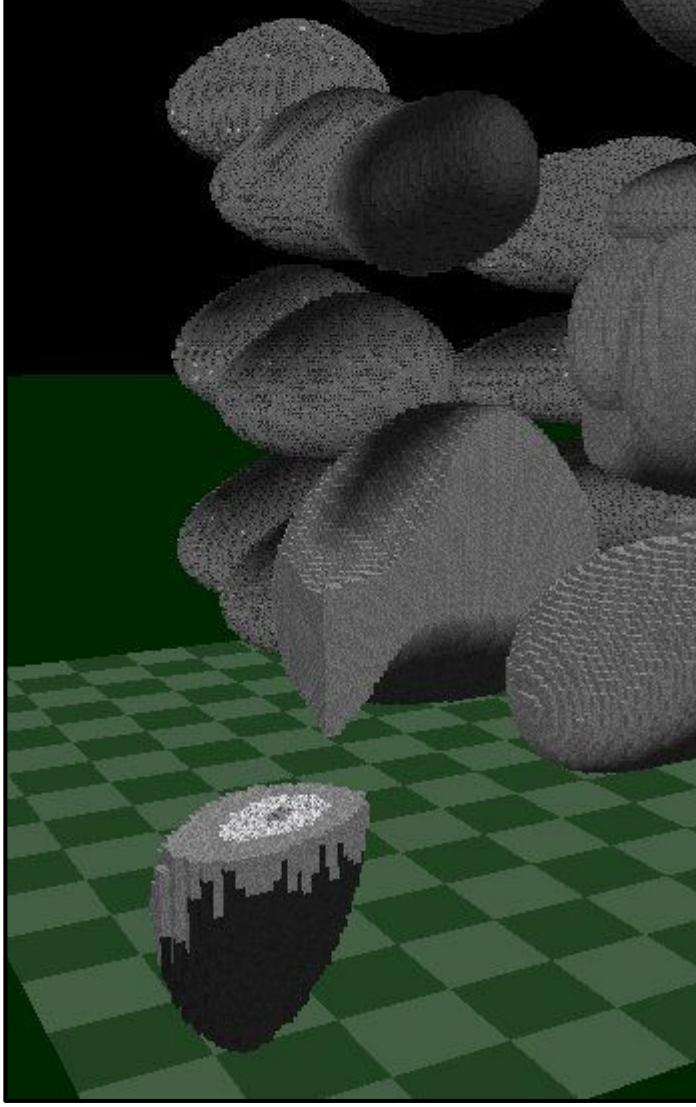
The disc of the solar system is scattered with each asteroid using an algorithm that randomly tries to place each one without overlapping a previous one.

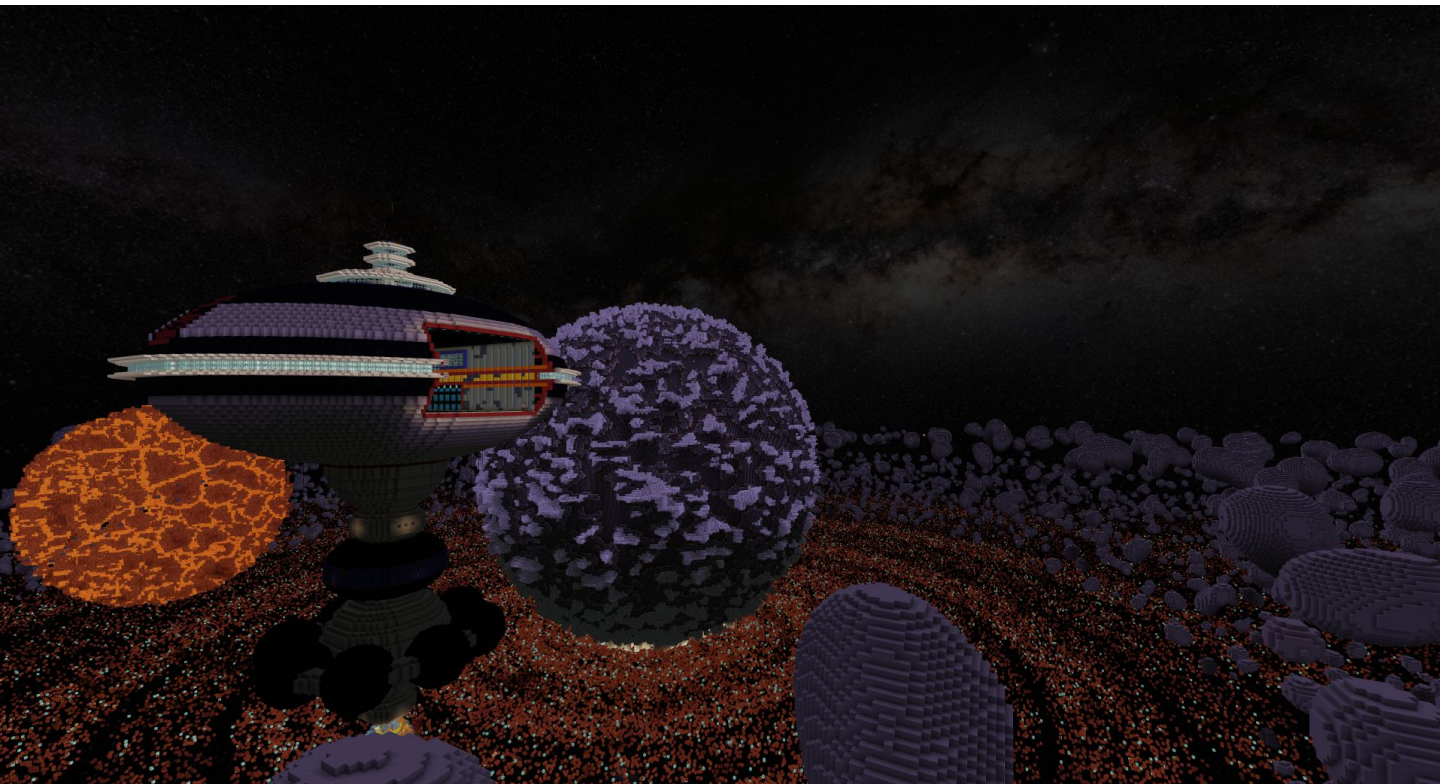
For the gameplay reasons it was important that the final challenge of finding the space ship was difficult enough to make the player balance the number of rocket boosters they have in their inventory against the problem of effectively searching among the rocks for the ship.

The number of large rocks that obscure the search needed to be balanced by smaller rocks, and these were packed into three different layers of rings around the central worlds to provide some clear site-lines to help the player be successful.

You can explore Space Station Z further on the Minecraft Marketplace here:

https://marketplace.minecraft.net/en-us/pdp?id=3d27cae6-1728-430a-be0c-c809a805745e





@MapMakingMag | MapMakingMag@gmail.com