# Visualizing Player Engagement with Virtual Spaces using GIS

Frederick Chan
fredchan@uw.edu

*University of Washington*

## 1 Abstract

GIS software is meant for Earth data, but I demonstrate here that you can convert video game data and put it in GIS software anyway. This is used to reveal player trends and clean up virtual player-generated litter. I focus on getting heightmaps, locations of gameplay objects, and metadata of Minecraft maps, converting it to formats usable by QGIS, and interpreting what that data could mean so you get why this is practical.

## 2 Background

The Earth is round. You know this[1], I know this, and so do the developers of geographic information systems (GIS) software. GIS software is optimized for this fact, since its main job is to manipulate and analyze Earth data. Although it can be used for other planets and moons as well[4], these are all round-ish objects like Earth is. This is unlike most virtual spaces in video games, which are flat, but we can try putting those spaces in GIS software anyway.

It's helpful for game designers to understand where players go in these virtual spaces, called maps, to find out what map features attract players. This can even be used to weed out pesky camping spots, where players sit around and pick off unsuspecting passersby, which is annoying and unfair.[6] By looking at these maps in GIS and visualizing where people go, map design can be improved to promote a fair and more fun game environment. In this paper, I demonstrate the practicality of importing data sources from video games into GIS and provide examples of what insights this can lead to[2].

In particular, I'll be extracting data from two different maps, *EvanTest* and *Ships and Stuff*, played in *Minecraft*, a sandbox game where players collect virtual blocks and use them to build in-game structures on the procedurally generated map. Using Minecraft with GIS is not unheard of, as GIS data has been imported into Minecraft for encouraging citizen participation in urban planning.[7] While that focuses on using real data in a video game to solve real problems, I aim to use video game data in software for real data to solve video game problems. It's obviously a very important cause.

Luckily, it's easy to import arbitrary data sources into Quantum GIS (QGIS) and not specify a map projection or coordinate reference system (CRS) that ties it to a real planet. The game's world is theoretically infinite, but is divided into regions of 32x32 chunks, each containing 16x16 blocks. Each region file stores chunk data in a well documented NBT format,[1] so it's easy to convert game data into data QGIS will understand. Each chunk stores its own heightmap, analogous to a digital surface model (DSM), representing the game world's surface and all the structures on it. Chunks also store the amount of time a chunk is loaded ("InhabitedTime" which is how long a player was in its vicinity recorded in ticks (0.05 seconds). By plotting these data on top of each other, we can see some interesting trends.

We can also plot the locations of undesired objects ("litter") in a map, such as repeating command blocks, which can cause an unpleasant, laggy gameplay experience when in excess. Plotting them on the heightmap shows the capability of players for altering the virtual terrain and creates a useful geographical reference for systematically cleaning up litter.

## 3 Converting game data

All the code is available at `https://github.com/fechan/MCGS` and written in Python. To extract the heightmaps, `extract_heightmaps.py` opens up a region file and looks at every chunk. Each chunk stores various heightmaps, but `OCEAN_FLOOR` is the one we're interested in, which stores the highest solid, non-air block. Each heightmap is a 64-bit array of longs, with every 9 bits being one elevation value between 0-256 (the height limit of the game). Knowing this, we can convert the heightmap into a Python list containing the elevations as integers

[1] Unless, of course, you're a flat Earther.

[2] This is all one big excuse to show you what I've built in Minecraft with my friends.

with the following function[3].

```
def unstream(value_bits, word_size, data):
    bl = 0
    v = 0
    decoded = []
    for i in range(len(data)):
        for n in range(word_size):
            bit = (data[i] >> n) & 0x01
            v = (bit << bl) | v
            bl += 1
            if bl >= value_bits:
                decoded.append(v)
                v = 0
                bl = 0
    return decoded
```

With the elevation data in a list, we can convert it into a geographic raster for QGIS. Since our heightmap represents a fictional, flat world, there's no real-world map projection or CRS that would be appropriate to apply, so we want to create a raster in a format that doesn't require one. The ESRI ASCII grid[2] is one of these, and has the added benefit of being extremely simple. We just define the (x,y) of the raster's lower-left corner to be the (x,z) of the Minecraft chunk and set the number of rows and columns to 16. Then we join the elements of our list of elevations using spaces and put it in the last line of the file[4]. Since there are $32 \times 32$ chunks in a region (that's a lot!), we can stitch each chunk raster into one big region raster with `gdal_merge`[3].

A similar thing is done by `plot_inhabitedtime.py` to get a plot of each chunk's InhabitedTime. This is just an integer, so we can just yoink the values into a list, set the raster to a size of $32 \times 32$ cells at 16 units (blocks) per cell, and join the array into the raster file. Then we have a plot of each chunk's InhabitedTime for the entire region.

Both of these rasters can be loaded into QGIS, and rasters of the same region will be automatically superimposed.

In order to plot the locations of specific blocks, a different approach is used since locations of blocks is better represented by a vector layer of points. For this, we can use PyQGIS, QGIS's Python API, to generate one. In `ore_plot_qgis.py`, I use the anvil-parser Python library to determine the location of a particular block given its coordinate. With this, I can scan every block in the region for the blocks I want to plot. Once the Python script is loaded into QGIS, I can run it and it will add a new layer with the plot. The block's precise location, including elevation, is stored in the attribute table.

[3]This is a Python rewrite of a Perl subroutine written by u/ExtraStrengthFukitol on Reddit.

[4]Y is elevation in Minecraft, but Z is elevation in QGIS. One caveat is that -Z in Minecraft is north while +Y is north in QGIS. When exporting the heightmap, you can mirror it vertically so that north is up and QGIS's Y matches up with Minecraft's Z.

## 4 Case study: Multiplayer survival

### 4.1 Background

*EvanTest* is a map generated and played on a multiplayer survival server that had 17 unique players in its playthrough. In survival, hostile mobs spawn that attack nearby players, who need food and shelter to survive. Players often take on goals such as defeating the powerful Ender Dragon, an end-game boss. To defeat it, players get resources to craft more powerful weapons and armor. Items can be made more powerful by enchanting them, sacrificing XP gained by killing mobs. In this playthrough, the players tended to divide themselves into two neighboring communities: Shack Village and Brazil. In Figure 1, the cluster of buildings on the left is Brazil and the cluster on the right is Shack Village.

### 4.2 Results

I successfully exported four regions' heightmaps and their InhabitedTime maps from Minecraft into QGIS, which is shown in Figures 1 and 2. Analyzing the statistics of the InhabitedTime raster of the main region of EvanTest where players inhabited indicates that chunks in the region were loaded for anywhere between 3,378 to 13,734,633) ticks (≈3 minutes to 191 hours), with a mean of 1,887,419 ticks (≈26 hours) and standard deviation of 3,366,901 ticks (≈47 hours). Complete maps of the whole area are in Appendix A: EvanTest Maps.

### 4.3 Discussion

To get food, players need access to farms. To craft items, they need access to mines, crafting tables, furnaces. Special resources required to craft certain items are also only available by traversing through Nether portals, special structures that players build. It is therefore unsurprising that people tend to hang around the metropolitan area (Figure 1), where all of these are available within a short distance. It's important to note that a chunk's InhabitedTime is only a measure of whether a chunk is loaded (within viewing distance of a player) and not whether a player was actually inside the chunk at the time. When a player is in a chunk, that player contributes to the InhabitedTime of all the chunks within their vision in a circle. As such, there are chunks in the metropolitan area with the highest InhabitedTime that

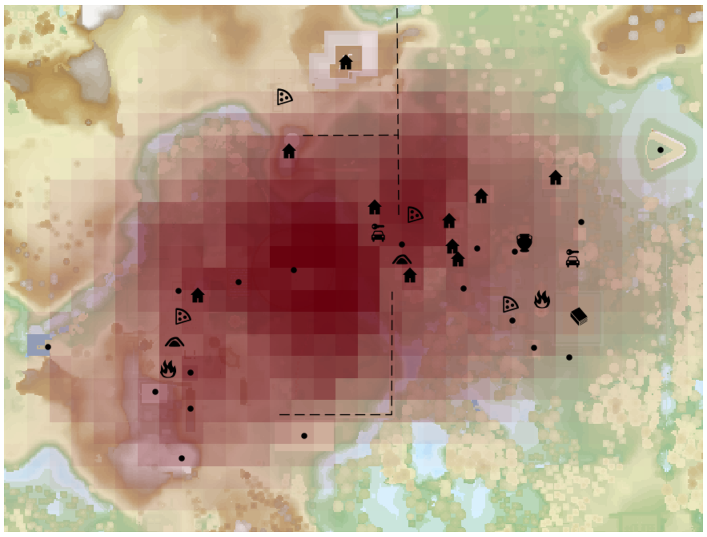


Figure 1: Pseudocolor map of the heightmap of the Shack Village-Brazil Metropolitan Area, with InhabitedTime map on top, and labels indicating the type of building underneath. House icons are player houses, vases are storage areas, cars are storage areas for horses, pizzas are food farms, caves are mines, fires are Nether portals, books are enchanting-related buildings, and dots are miscellaneous. Permanent transportation infrastructure, like minecart rails, are indicated with dotted lines.

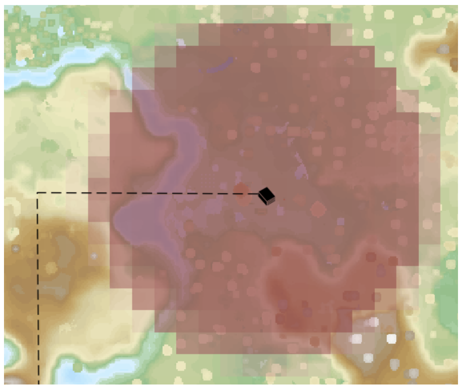


Figure 2: Map of the Mob spawner and AFK fishing hole. The dotted line is connected with the dotted line going up Figure 1 by extra rail in between. (See Appendix A for more detail).

have paradoxically few buildings. This is probably because it's the intersection of circles (imagine the middle of a Venn Diagram) around the urban cores of Shack Village and Brazil respectively, and not because people spend a lot of time hanging around the Colosseum between them, which is decorative and serves no gameplay purpose.

The second hot zone, completely separate from the metropolitan area, is the Mob spawner and AFK fishing area. Being one of only two available enchanting buildings, it is important for being the only convenient and readily available source of XP required for enchanting. This is due to the mob spawner block within, generated upon map creation, that cannot be moved. Players sit around the mob spawner and kill mobs for XP. Inside the building are other facilities dedicated to enchanting, such as the enchanting table and AFK fishing area. The AFK fishing area in particular is a tremendous contributor to the surrounding InhabitedTime; it allows players to fish while being away from their keyboard (AFK). Players would leave themselves logged in and fishing there overnight hoping for enchantments available only though fishing or trading with non-player Villagers. Contrast this with the Villager Tenements and Trading Grounds in the bottom right of the metropolitan area, which was established much later and relies on trading rather than fishing to acquire these enchantments. Trading requires significantly less time commitment, and therefore has comparatively low InhabitedTime around it.

This information can be used to inform and evaluate the placement of transport infrastructure. For example, the transport rail in the top of Figure 1 and in Figure 2 is effective since it connects the middle of the metropolitan area and the mob spawner. Meanwhile, the ice-boat bridge on the bottom of Figure 1 may not be so useful, connecting the urban center to a building without much function. Players making new buildings could, however, be encouraged to build near it and give it more purpose.

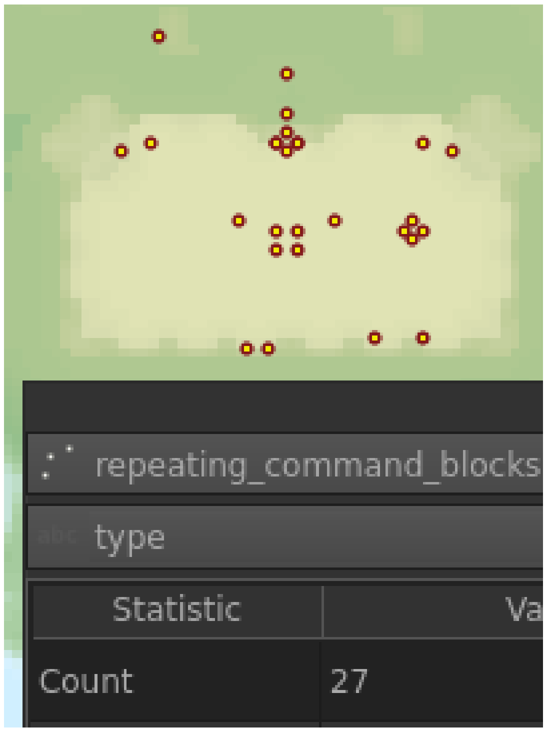
## 5 Case study: Multiplayer creative

### 5.1 Background

*Ships and Stuff* is a map generated and continuously played on for over 5 years. Unlike EvanTest, this world is a creative mode map, meaning that players are invincible and are given unlimited blocks to build with. The only goal is to channel your creativity (hence *creative* mode) and build to your heart's content. On this map, players mostly built starships from the popular sci-fi franchise *Star Trek* (hence "Ships") as well as other miscellany (hence "and Stuff"). Over the years, there has been a gradual build-up of litter by players. Repeating command blocks, which issue commands that check and modify the game state every tick that they're loaded, run simultaneously and create lag when there are a lot of them. One trend that surfaced while playing on this map was creating traps out of these blocks, which checked for nearby players and annoyed the living daylights out of them. In order to be effective as traps, they were hidden from view. Years later, when people realized they were causing lag, they were horrendously hard to find because they were buried in places nobody could see. The extent to which this player trend has changed the landscape was unknown, but with the power of GIS, we can find all these pesky blocks and put and end to them once and for all.

### 5.2 Results

Scanning the four main regions of the map turned up an impressive 295 repeating command blocks littered around the world. See Appendix B, Figure 5 for a plot of all the litter that showed up.



repeating_command_blocks

type

| Statistic | Va |
|---|---|
| Count | 27 |

Figure 3: 27 repeating command blocks in this guy's house alone! Naughty, naughty!

### 5.3 Discussion

Years and years of litter accumulation occurred on the map, despite surface-level cleanups. Hundreds of command blocks were just sitting around constantly doing things, creating lag for players for years. Knowing the precise locations of all the litter in the area certainly makes it much easier to clean up, which hopefully reduces the lag significantly. If you use GRASS GIS's `v.net.salesman`[5], you can try to make an optimal route visiting all the repeating command blocks in the world to get rid of them. I dub this the "Traveling Minecraft player problem."

Repeating command blocks aren't the only thing you can scan the world for, either. You can scan the world for ores and other resources, if so inclined. Useful for people who don't want to spend time looking for ore. For diamond ore especially, you can skip scanning a ton of blocks by taking advantage of the fact that they spawn at elevations below 16 and in veins that appear only once per chunk. This would save a considerable amount of processing time.

## 6 Conclusion and future work

All in all, extracting data from Minecraft and into QGIS is a relatively simple and practical procedure. I leveraged existing libraries and bridged the gap between video games and software meant for modeling the real world. It can reveal interesting patterns in where players go and what players do, and aids in creating actionable plans for increasing building visibility and use. It can show the effect of years of play on a map on accumulation of litter and be a tool in cleaning it up a at the same time. Plus, it just makes some pretty cool looking maps. I mean, just look at them.

What can be done with GIS software isn't limited to what can be seen here. All the compatible tools that GIS provides is at your disposal. The methods outlined here can also be generalized for other video games. If you can extract the layout of the map, it can be a basemap that provides geographical context for the other data you want to plot.

## 7 Acknowledgments

A word of thanks to Brian, Evan Grilley (Embry Riddle Aeronautical University), James Akina (Central Washington University), Jack Doughty, Logan Lemieux (Western Washington University), Oliver Low (Georgia Institute of Technology), Tom Connolly (Carnegie Mellon University), and others for playing and building up *EvanTest*. A word of thanks also goes out to Cole Ellis (Oregon State University), James Gale (University of Washington), and Oliver Low (again), and others for playing and building up *Ships and Stuff*. James Gale also requested that I note that he "made all the good ships."

## References

[1] Chunk format — Minecraft wiki. https://minecraft.gamepedia.com/index.php?title=Chunk_format&oldid=1497793. [Online; accessed 14-February-2020].

[2] Esri Grid format — ArcGIS resource center. http://help.arcgis.com/en/arcgisdesktop/10.0/help/index.html#//009t0000000w000000. [Online; accessed 14-February-2020].

[3] gdal_merge — GDAL documentation. https://gdal.org/programs/gdal_merge.html. [Online; accessed 14-February-2020].

[4] USGS astrogeology mapping, remote-sensing, cartography, technology, and research (MRCTR) gis lab. "https://www.usgs.gov/centers/astrogeology-science-center/science/mrctr-gis-lab". [Online; accessed 14-February-2020].

[5] v.net.salesman — GRASS GIS 7.6.2dev reference manual. https://grass.osgeo.org/grass76/manuals/v.net.salesman.html. [Online; accessed 18-February-2020].

[6] Simon Egenfeldt-Nielsen, Jonas Heide Smith, and Susana Pajares Tosca. *Understanding Video Games: The Essential Introduction*. Routledge, 2 edition, 2012.

[7] Fanny von Heland, Pontus Westerberg, and Marcus Nyberg. Using Minecraft as a citizen participation tool in urban design and decision making. *Future of Places, Stockholm*, 2015.

## A EvanTest Maps

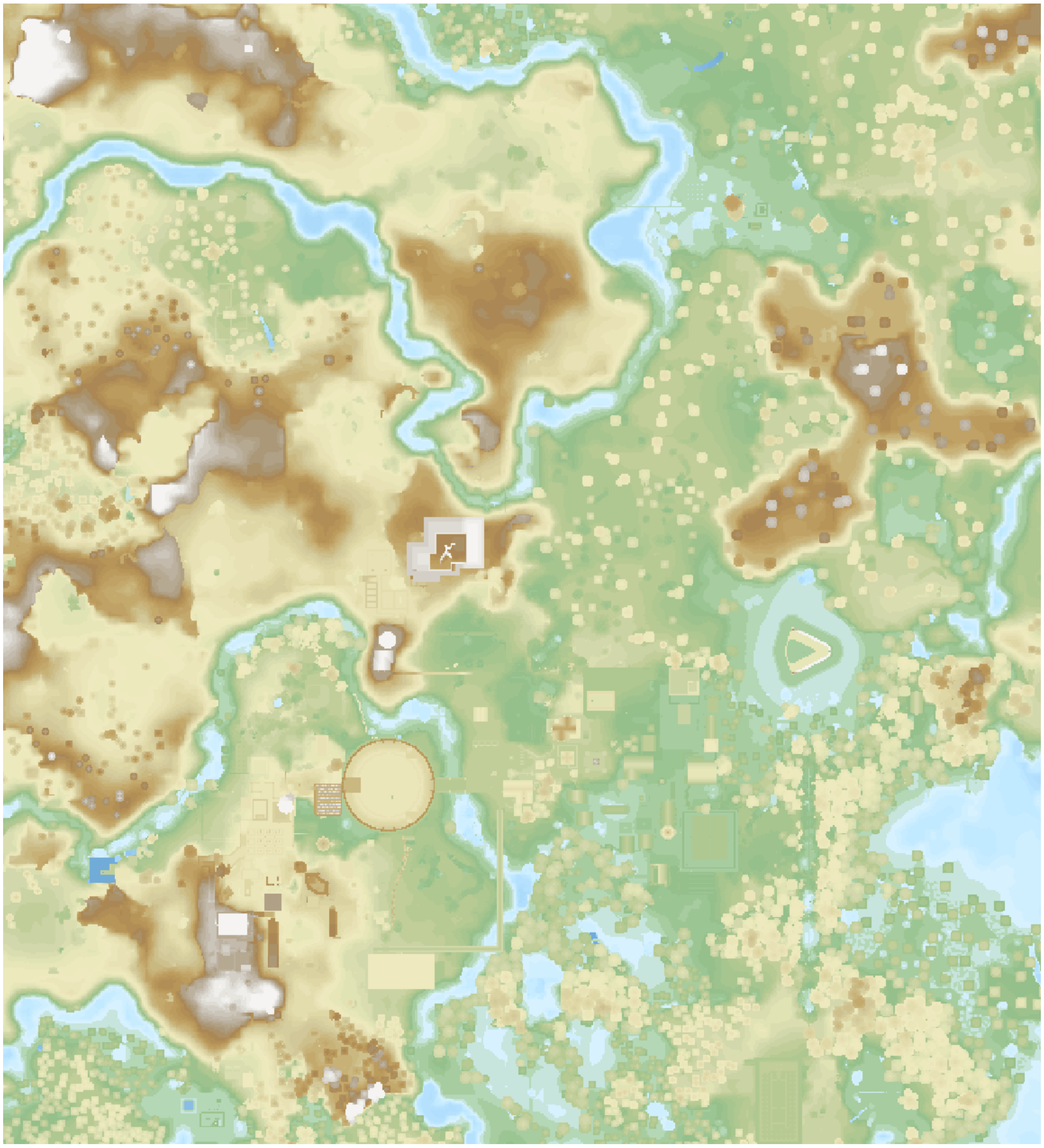


Figure 4: A part of EvanTest's main region's heightmap with a pseudocolor gradient (cpt-city wiki-2.0) applied.

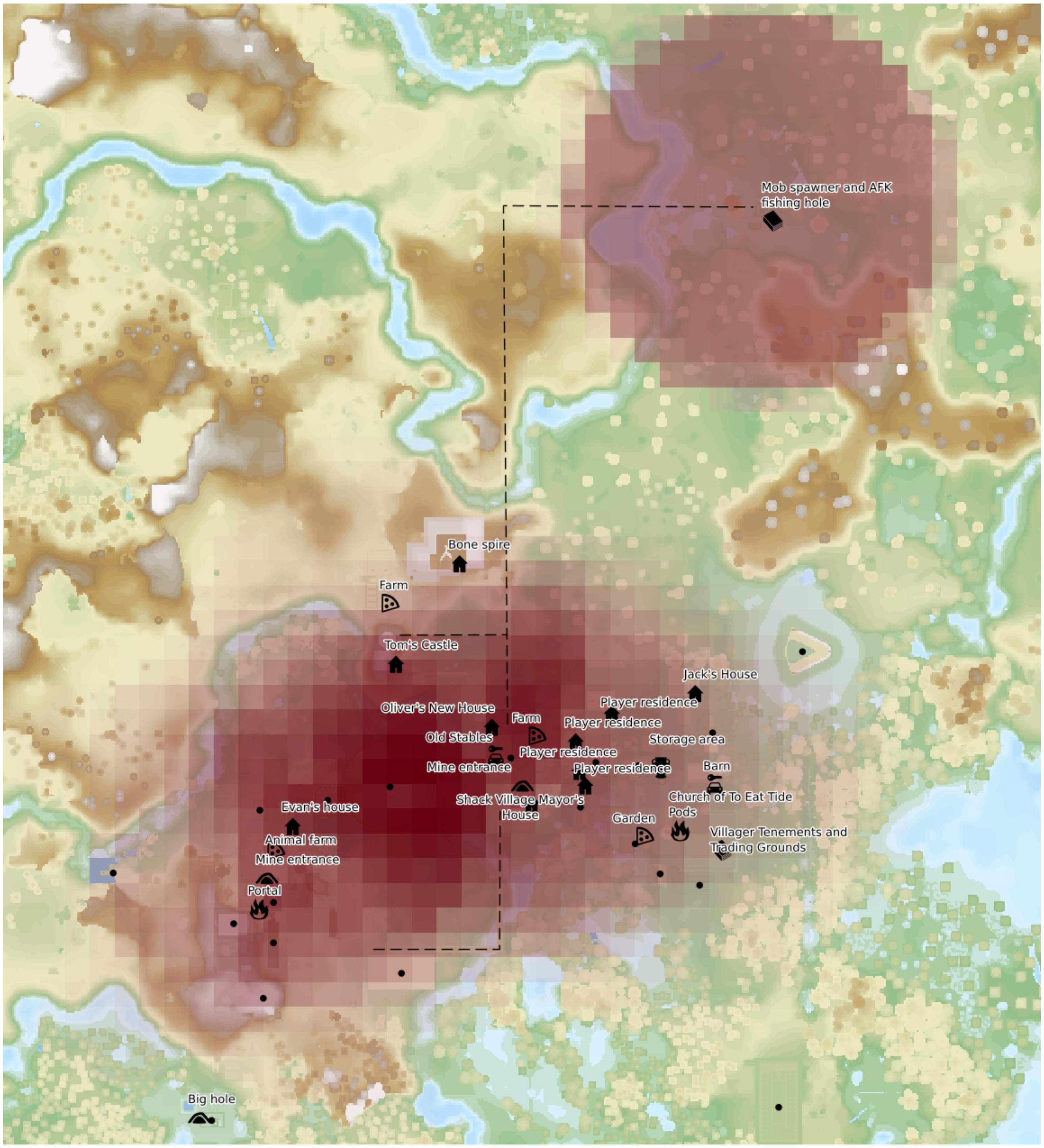


Figure 5: Above map with InhabitedTime map superimposed. The redder, the more inhabited. Labels are also added showing the purpose of buildings below. House icons are player houses, vases are storage areas, cars are storage areas for horses, pizzas are food farms, caves are mines, fires are Nether portals, books are enchanting-related buildings, and dots are miscellaneous. Permanent transportation, like minecart rails, are indicated with dotted lines.

## B Ships and Stuff Map

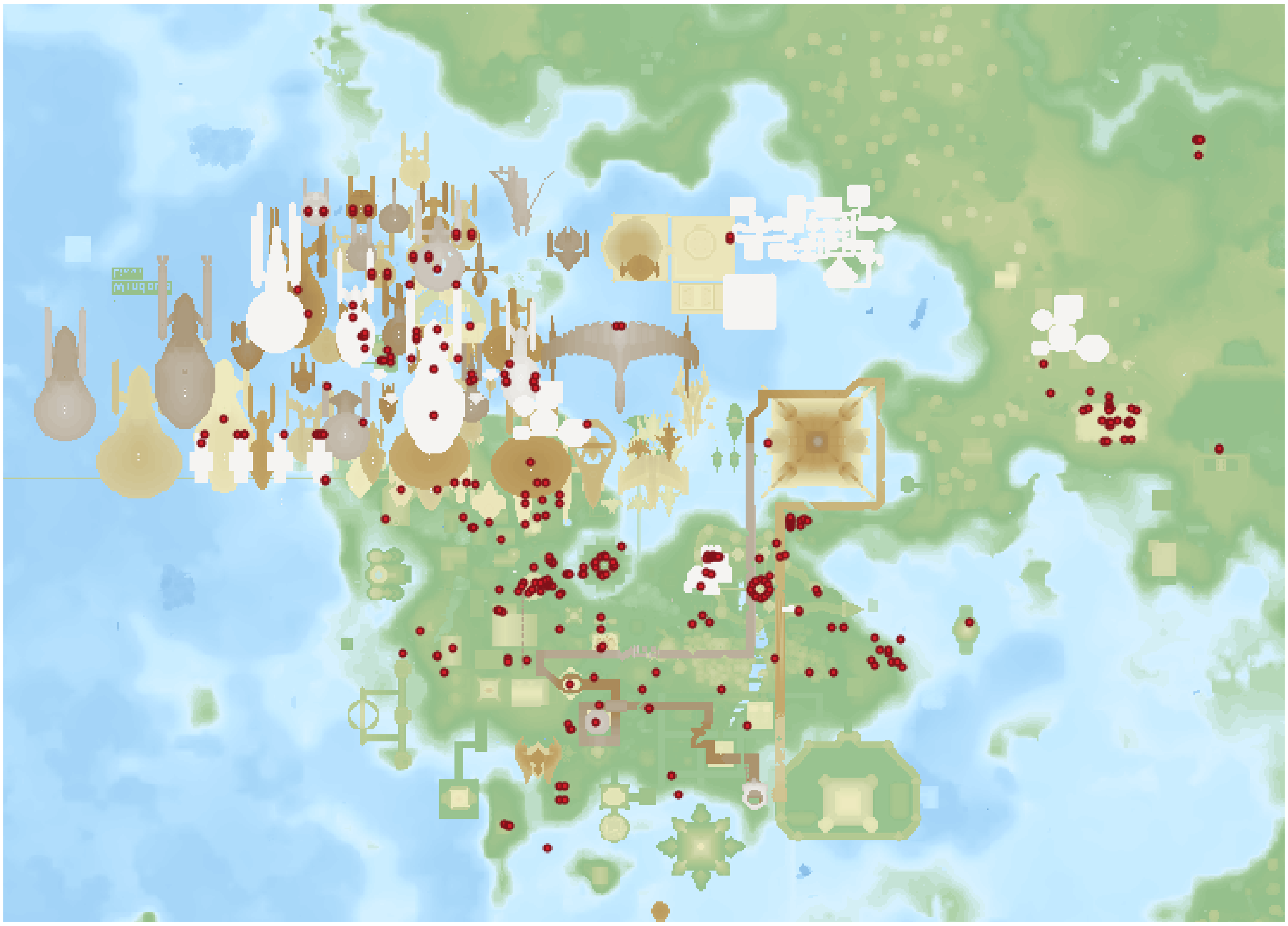


Figure 6: Heightmap of Ships and Stuff with locations of repeating command blocks plotted on top as red dots.