Elysian Development Walkthrough by Nyseus

> **Please understand, there are feature spoilers in this page. Only read if you are fine with that.**

2025 January - May

Early Concept

The Original Elysian was much simpler and more basic compared to the idea of it today. This exposed several flaws in the core server vision and design that I (Nyseus) promised to fix based on player base feedback.

Some of these Polls, can still be found in the polls channel on our Discord server, starting at: https://discord.com/channels/1277714476635783258/1283544508465811556/1367218847517642762

During these times, I also introduced the idea of a Custom world, with fully custom features like WorldEvents. The feedback to these was overwhelmingly positive.

What did the idea of a Custom world, and “WorldEvents” imply truly?

Creating a custom, handcrafted world in a modded environment like Elysian is inherently difficult and a generally new concept within the Minecraft Community. Tools like WorldPainter do not natively support many modded elements. This results in mainly 2 outcomes: a low-quality world, or a world that takes considerable effort.

At Elysian, we decided to pick the latter to work on a World that is truly fit for the descriptions that we want, with all the elements that we originally planned properly implemented. All to provide an unforgettable experience for the players.

WorldEvents was a concept that I had planned for a long time. By default, it’s a custom server-side and client-side mod that adds fully custom elements to the world that rely on certain events. I originally intended to collect already made mods that would serve the same purpose; however, many didn’t exist, some were not compatible, or were low quality. This realistically only left the option of making it from scratch.

In May 2025, mainly due to server inactivity, I have decided to shut down the server to take a break from server management and to later begin the work on the features planned.

2025 June - September

Laying out the foundations, expanding on core concepts.

At the beginning of June, I realized that alone I am not able to develop the features that I had originally planned, nor can I properly take care of the server alone. As a result, I announced an admin recruitment program to get some help.

Out of the 7 total applications, **James, s1ice, and Gabry** were accepted as Admins. Most of the other applicants had insufficient experience, meaning they couldn't really provide any help with the server or with its development. Despite this, some of those applicants were accepted into Moderator and Helper positions.

Following this, we further discussed the concept of the World map, and how it should look and function. Here, basic concept drawings were made, among others.

Corland 2025-07-08-17-59 (1) (1).png
The Map itself would be 6000 x 6000 (or 5000 x 5000) blocks wide with mod compatibility.

What do I mean by mod compatibility?

Elysian has several mods that generate structures, trees, crops, etc. We wanted some of those to be present on the custom world as well. Not everything is doable, but we could discuss that further later. It was important that structures on the custom map should retain their functionality (achievements, potential area effects, etc. that come with the mod).
The world itself would also have custom ore/underground generation (ore diversification) where some regions are rich in certain resources, for example, lots of iron and coal, but barely any diamonds.
The world design itself would be realistic, but still exciting on a small scale. Meaning, not super realistic, where it's a plain, and in Minecraft with low render distance, it looks like you are just in a flat area for hours on end. The world should feature mountains, hills, rivers, oceans, forests, and a big variety of biomes (heavily influenced by the mods we have on the server).
At this time, it was understood that none of the members of Staff was capable of working on a Map like this. As it required intensive custom features that would require too much work. I decided that instead of letting this idea die, I would hire a Freelancer to make this map instead.

For XXX$ USD, which was financed out of my own pocket, a professional mapmaker was hired to work on the project, and he accepted and promised delivery within a month.

In the meantime, some other Staff members began working on other basic server work. Such as the complete reset of the server, the fixing / renewing of Server configuration. I also started expansion work on the Performance tool I had released in the previous Season, called "Performance BETA" (PerfB).

PerfB was a script-based (Skript) tool that automatically regulated server settings based on live load and executed entire server restarts. I made this script by hand using the Skript Spigot plugin. It was an extremely simple, yet surprisingly effective for what it was worth:

```
# Periodic TPS warning every 15 minutes if TPS is low
every 15 minutes:
  set {_tps_raw} to placeholder "server_tps_1"
  set {_tps} to {_tps_raw} parsed as number
  if {_tps} < 13:
    broadcast "§e[§6Performance BETA§e] §fServer is experiencing some server lag. §7This is normal at higher player counts. §fTo help minimize your impact on the server, avoid travelling fast or doing lag-intensive activities."

every 1 minute:
  # Get TPS and Memory
  set {_tps_raw} to placeholder "server_tps_1"
  set {_tps} to {_tps_raw} parsed as number
  set {_freeMB} to free memory
  set {_maxMB} to max memory
  set {_freePct} to ({_freeMB} / {_maxMB}) * 100

  # TPS Lag Handling
  if {_tps} < 15:
    if {performance.adjust.enabled} is true:
      set {performance.lag.detected} to true
      execute console command "schedule clear incendium:clocks/main"
      execute console command "schedule clear incendium:clocks/1m"
      execute console command "schedule clear incendium:clocks/1s"
      execute console command "schedule clear incendium:clocks/2t"
      execute console command "schedule clear incendium:clocks/7s"
      # Incendium Clocks consume a lot of power, and are not used often
      execute console command "dynview setsimulationdistance 1"
      execute console command "dynview setviewdistance 1"
      # Chunkload singlehandely contributes over half of all pressure
      execute console command "bluemap stop"
      # Bluemap has very pressure heavy chunk scanning

    # Periodic entity cleanup (every 5 minutes)
    if {performance.cleanup.enabled} is true:
      add 1 to {performance.cleanup.ticks}
      if {performance.cleanup.ticks} = 5:
        broadcast "§e[§6Performance BETA§e] §fRunning entity cleanup commands..."
      if {performance.cleanup.ticks} >= 5:
        execute console command "minecraft:kill @e[type=alexsmobs:cockroach]"
        execute console command "minecraft:kill @e[type=alexsmobs:fly]"
        execute console command "minecraft:kill @e[type=minecraft:marker]"
```

```
            # These entities are useless, but we cant control them. Monkey mode
on
            execute console command "lagg clear"
            # Clear dropped entities
            set {performance.cleanup.ticks} to 0

  # Emergency RAM Check (OS-level)
   set {_ram_used} to placeholder "server_max_ram" parsed as number
   set {_ram_max} to placeholder "server_ram_max" parsed as number
   if {_ram_used} > 25450:
      if {performance.restart.enabled} is true:
         send action bar "§c[Performance BETA] Executing Emergency restart to
avoid crash in 30 seconds!" to all players
         play sound "block.end_portal.spawn" at all players
         execute console command "save-all"
         wait 10 seconds
         send action bar "§c[Performance BETA] 20 seconds left until Restart." to
all players
         play sound "block.note_block.pling" at all players
         wait 10 seconds
         send action bar "§c[Performance BETA] 10 seconds left until Restart." to
all players
         wait 5 seconds
         play sound "block.note_block.pling" at all players
         send action bar "§c[Performance BETA] 5 seconds left until Restart." to all
players
         wait 5 seconds
         send action bar "§c[Performance BETA] Restarting now." to all players
         play sound "block.note_block.pling" at all players
         wait 1 second
         execute console command "restart"
```

This simple script prevented multiple server crashes and overall did a decent job of regulating server lag for what it was worth. This script managed to improve overall TPS during high load by around 2-3 TPS. This is really valuable, especially when under load the server drops to around 12-13 TPS, and even the smallest things matter. For reference, perfect server performance (Ticks Per Second, TPS) is 20. A frozen server is 0. At half, so around 10 TPS, your game is effectively running at half speed.

This basic setup is relatively common amongst most modded servers, so this wasn’t some sort of revolutionary invention.

While designing this script, it was super important to figure out the exact lag causes. This meant pressure testing and running profilers and hoping to spot what stands out.

Example of a Spark Flamegraph that we use to identify lag problems:

Screenshot 2025-11-26 002213.png

> In this particular example from November 2025, the source of the issue is that the Majrusz's Library mod is performing heavy checks to spawn Cursed Armor chests, causing intense lag during chunk loading. We later fixed this particular issue by building a mod that injects into the culprit mod and throttles/limits these checks without killing its functionality.

During this same period, we also started work on the ElysianKD mod. This mod was inspired by an Idea to make combat itself more realistic and let it expand more into roleplay politics. The core idea is simple: Players who reach very low health can be "knocked down", which is semi-realistic, as someone who's heavily injured is typically not able to jump around and run away.

Expanding further on this idea, we came up with the ability to arrest players who are knocked down. As sometimes people prefer to arrest someone rather than have them respawn. This, by itself, opens up a whole new layer of Roleplay. Allowing the arrested individual to be trialed and jailed.

This was the first custom mod we started to develop, but we were mainly still inexperienced and needed help; because of this, I hired yet another freelancer to help make this mod. For XXX$ USD, the developer offered to make this mod. After some trial and error, this mod was completed around November 2025.

Screenshot 2025-11-05 182402.png

Around this time, in August 2025. The first-ever problems started to show.
The Freelancer I originally hired to make the Map for Elysian missed their deadline. They claimed to be very busy with other orders, which I personally found highly disrespectful, as I hired them when they were still not busy and they accepted more orders knowing it would affect mine.

I gave this person 1-2 more weeks to have the map ready; however, they declined, explaining they won't be able to do that. As a result, this order was cancelled, and I had to find another Freelancer to complete the work. This already pushed development away by 1-2 months.

Roughly in September, another Freelancer was hired who looked more capable. Promising delivery around October 2025.

Around this same time, I decided to move away from dedicated Minecraft hosting. Our development and performance orianted works were limited by such hosts who do not give root terminal access. I understand why; we just outgrew such limitations. As in the next phases of Performance orianted works, we needed to directly modify JVM flags, exact RAM allocation/diversification, and more.

We decided to fully rent a server with basically limitless access. This move today allows us to host more than just a Minecraft server. With this server, we host our own API, websites, and more. However, back then, it just meant more RAM and a better CPU for roughly the same price, with more possibilities in the future. After migrating the entire server to the new location and properly configuring it, we were back on track.

2025 October - December

Fine, we will do it oursevles.

October was the time when we realized we were alone and there was nobody able to help us anymore. The questions we were asking were new and hypothetical. Sometimes our concepts were so complex and difficult that experienced devs offered no help or advice.

The Freelancer I previously hired the month before also gave up on our Map project as simply too complex, with him underestimating the amount of work required. This was the last time we decided to hire a Freelancer, as we have realized they are too ambitious, yet unreliable.

Our savior in this issue ended up being James, whom I recruited a few months prior as an Admin. He offered to struggle through the complexities of the map and build it up from scratch, as he had some experience with the tools we use to make custom maps. Unlike the freelancers, James actually made progress on the map, and together we were navigating the issues that the idea of a modded map provided. Let's go through the issues

> “
>
> - **WorldPainter is not really compatible with modded elements.**
>
> If we want to use brushes and other elements such as the modded trees, custom modded blocks in terrain, and other things like custom rivers and structures. It’s a whole shebang.
>
> - **Structures cannot be used with WorldPainter.**
>
> Simply based on how Minecraft works, structures are generated in a phase before the terrain is even made. Trying to reverse this process, aka placing the structures after the world is already done, is basically impossible.
>
> - **WorldPainter itself is an old and non-user-friendly software.**
>
> As a software made in 2011, its UI and some of its methods are unconventional and sometimes require repeated monotone actions. Absolutely no hate to the makers of WorldPainter; I don’t blame them; it’s a non-profit software maintained by 1 person.
>
> - **Most of us lack any artistic ability.**
>
> Not directly related to modding, but I personally have no artistic talent whatsoever; sl1ice and Gabry are largely in the same boat as me. This basically means that the part where someone has to draw the world by hand falls on 1 person. That one person being James.

Despite all of these issues, we began work on the map, solving the issues as we went.

While James was working on the map, I began looking back at the performance improvements. I have come to the conclusion that the expected server load cannot be sustained by a performance regulator as simple as PerfB. We needed something more advanced, something more dedicated, and powerful.

With this, I began theorizing on how to expand on performance. Elysian is a Hybrid Arclight server, which means it runs on Forge with an implementation for Bukkit/Spigot plugins to run on the same server. This is something that's normally not possible, and also relatively unpopular/underrated. This means whatever performance implementations we make have to be compatible not just with Forge but also with Spigot. This is already highly difficult and new.

I dived deep into server performance research, and came to the conclusion that experimental multithreading is the single most effective way to eliminate such issues, even if it’s unstable. Furthermore, I wanted to expand on the base features of PerfB, with more targeted, less vague, and less unreliable metrics, etc.

As a result of my research, I began working on Elysian’s custom performance mod, which I decided to call Haste.

The core idea behind Haste was to take some of the work normally done by Minecraft’s main server thread and safely distribute it across multiple threads. Minecraft servers are mostly single-threaded, meaning that the majority of important work has to be completed one task after another. If one mod, entity, or loaded chunk takes too long to process, everything else has to wait for it. This is one of the main reasons why simply giving a Minecraft server more CPU cores usually does not improve its performance by much.

Obviously, multithreading Minecraft is not as simple as telling the server to use more threads. Minecraft, Forge mods, and Spigot plugins generally expect most actions to happen on the main server thread. Moving the wrong thing to another thread can result in crashes, duplicated or missing entities, broken commands, world corruption, or issues that only happen under very specific circumstances.

This problem is even more complicated on Elysian. Haste has to understand not only how Minecraft and Forge handle an entity or event, but also whether Arclight passes that same action through Bukkit. A change that is safe for a normal Forge server may still break a Spigot plugin, while a common Spigot optimization may be completely incompatible with a Forge mod.

Because of this, I decided that Haste should not blindly attempt to multithread everything. Instead, it would identify specific entities and workloads that could potentially be processed asynchronously, while keeping unsafe or incompatible ones on the main thread. The system would also need different performance modes, automatic fallbacks, and detailed information about what was actually being processed.

I also wanted Haste to replace the vague measurements used by PerfB. TPS alone tells us that the server is lagging, but it does not explain why. A server can technically remain at 20 TPS while individual ticks, entities, or chunks are already causing major pressure. Haste therefore needed to track more exact information, such as the amount of loaded entities, asynchronously processed

entities, loaded chunks, tick times, and the reasons why certain workloads were excluded from its optimizations.

This quickly turned Haste into more than a basic performance mod. It became an experimental attempt to find out how far Minecraft server performance can actually be pushed, while still remaining compatible with the unusually complicated Forge, Arclight, and Spigot environment that Elysian uses.

> Haste's actual development lasted well into December 2025; to this day in July 2026, Haste is still being constantly worked on roughly on a daily basis to work out minor bugs and improve the mod itself.

To explain Haste properly, it is useful to look at some of its most important code. These examples are slightly shortened to make them easier to understand.

Measuring server performance

```
public void onServerTickStart() {
    tickStartNanos = System.nanoTime();
}

public void onServerTickEnd() {
    long elapsed = System.nanoTime() - tickStartNanos;
    addSample(elapsed / 1_000_000.0d);
    updateSmoothedTps();
}
```

Every Minecraft tick, Haste records when the tick started and when it ended. The difference is converted into MSPT, meaning Milliseconds Per Tick.
A perfect 20 TPS server has around 50 MSPT or less. Unlike simply checking TPS every few minutes, this allows Haste to continuously see how expensive individual ticks are. It also stores short-term and long-term averages, so one random lag spike does not immediately cause the entire server configuration to change.

Deciding which entities can be multithreaded

```
if (entity instanceof Projectile
        || entity instanceof AbstractMinecart
        || entity instanceof ServerPlayer
        || isBlockedEntityClass(entity.getClass())
        || synchronizedEntities.contains(entityTypeId)) {
    return true;
```

```
}
```

Not every entity can safely run on another thread. Players, projectiles, minecarts, boats, falling blocks, and several known problematic modded entities are kept on the main thread.

This is one of the most important parts of Haste. Blindly multithreading every entity would probably improve performance for a few minutes before breaking the server. Haste instead checks each entity and decides whether it should be processed asynchronously or normally.

Sending safe entities to the thread pool

```
if (shouldTickSynchronously(entity)) {
    return false;
}

CompletableFuture<Void> future = CompletableFuture.runAsync(
        () -> performAsyncEntityTick(world, entity),
        tickPool
);

taskQueue.add(future);
return true;
```

If an entity passes the safety checks, Haste sends its tick into a separate worker pool. This allows multiple entities to be processed at the same time instead of waiting for every entity before them.

If Haste returns false, Minecraft runs the entity normally on the main thread. If an asynchronous entity causes an error, Haste does not immediately retry the same tick, as it may have already changed part of the world. Instead, that entity type is automatically moved back to synchronous processing for future ticks.

Waiting for the work to finish safely

```
CompletableFuture<?> allTasks =
        CompletableFuture.allOf(futuresList.toArray(new CompletableFuture[0]));

while (!allTasks.isDone()) {
    pumpMainThreadTasks(128);
    ServerCompat.pollTask(server);
}
```

Haste does not simply launch entity tasks and forget about them. At the end of the entity phase, it waits for the submitted work to finish before the server continues.

While waiting, it processes tasks that need to return to the main thread. This keeps the different worker threads within the same overall Minecraft tick and reduces the chance of one tick continuing while entities from the previous tick are still modifying the world.

Handling Forge and Spigot compatibility

> ```
> if (ParallelProcessor.shouldBridgeThreadAffinityCallbacks()) {
>     return ParallelProcessor.callOnMainThread(
>             () -> original.call(event, wrapper)
>     );
> }
> ```

Some Forge events and Bukkit actions are not allowed to run asynchronously. If an entity triggers one of these actions from a Haste worker thread, Haste places the action into a queue and executes it on the main server thread.
This is especially important for Arclight. Without this bridge, a Forge event could indirectly call a Bukkit plugin from the wrong thread. Arclight would either block it with its AsyncCatcher or allow unsafe server state to be modified.
If an entity repeatedly requires this bridge, Haste assumes that its behavior is too dependent on the main thread and moves that entity type back to synchronous ticking.

Automatically reacting to server pressure

> ```
> if (memHeadroom < 0.10d) {
>     return PerformanceMode.VERY_HARD;
> }
>
> PerformanceMode desired = evaluateDesiredMode(smoothedTps, general);
>
> if (isTightening(desired, current)) {
>     return desired;
> }
> ```

Haste combines its performance measurements with the amount of remaining memory to select a performance mode. These modes range from *NORMAL* to *VERY_HARD*.
Restrictions can become stronger immediately when performance drops. However, Haste requires longer-term stability before relaxing them again. This prevents the server from constantly switching between modes whenever TPS moves slightly up or down.
This is the main difference between Haste and PerfB. PerfB reacted to a few basic thresholds by running commands. Haste directly measures the server, separates different types of pressure, and adjusts its internal systems while the server is running.
While explained like this, Haste may look relatively simple. Check whether an entity is safe, send it to another thread, and wait for it to finish. In reality, developing these systems took months of research, testing, crashes, and trial and error.

Minecraft was never designed for this type of multithreading. Many of its systems silently assume that they are always running on the main thread. A piece of code could appear to work perfectly during testing, only to cause a rare crash several days later when a specific modded entity enters a portal, loads a chunk, triggers a Forge event, or indirectly calls a Spigot plugin.
Finding these problems was also not straightforward. Some issues only appeared with thousands of entities, tens of thousands of loaded chunks, or multiple players travelling at the same time. Others were caused by combinations of mods rather than Haste itself. This meant repeatedly studying crash reports and Spark profiles, examining Minecraft's internal code, creating compatibility fixes, and moving unsafe systems back to synchronous processing.
Even the safety systems required their own trial and error. The main-thread bridge, automatic entity fallback, chunk quarantine, thread-safe collections, synchronization locks, and detailed statistics were added because simpler implementations eventually failed in real server conditions.
Therefore, the code shown above is not the first version of Haste. It is the result of many earlier versions that crashed, froze, failed to improve performance, or revealed another limitation in Minecraft, Forge, Bukkit, or Arclight. What now looks like a few relatively understandable checks represents months of identifying exactly where multithreading works, where it does not, and how Haste can safely recover when an assumption turns out to be wrong.

Today Haste is a server-side mod that contains:

- 154 production Java files
- 24,400 total Java source lines
- 21,380 non-empty Java lines
- 10 test files with 277 additional lines
- Approximately 1 MB of tracked source code
- A compiled mod size of 584,613 bytes, or roughly 571 KB

The compiled mod itself may seem surprisingly small. This is because Haste does not include Minecraft, Forge, or Arclight inside its own file. It only contains the code needed to modify and control those existing systems.
Despite being less than 1 MB when compiled, Haste currently contains over 24,000 lines of production Java code. Much of this code is not the multithreading itself, but the configuration, measurements, compatibility layers, safety checks, recovery systems, commands, chunk management, entity cleanup, and Mixins required to make the main optimizations function on a real hybrid server.

> Using Haste, we were also able to properly calculate the perfect JVM flags for the server. We learned a lot, such as how Elysian is actually perfect for using Generational ZGC (uncommon among Minecraft servers) and that running the server requires way less RAM than we originally expected.

Haste, due to its nature, being fully developed by us, and since it's a server-side-only mod. Is not open-source to protect and copyright our hard work.

While I was working on all of this, James was busy working on the map and its related issues.
The first problem of WorldPainter not liking modded blocks was solved relatively quickly. However, some issues became more intense than we expected.

Around December 2025, WorldPainter and structures became one of the largest problems we were facing. I'd like to take you through how we tried solving this problem.

Attempt no. 1
Manually place the structures.
We have decided that, for once, instead of trying to reinvent Minecraft, we will just build a mod that smoothly allows us to manually place the structures. We called the mod very creatively "structureplacepreview".

The mod itself is similar to Schematica or Create Schematics. We select a structure, position it like a schematic, and then place it via commands. A very Monkey solution to a super complex problem. In terms of functionality, it worked; however, we soon ran into the Problem that the map itself would be huge, and even with 4 staff members, it would take many hours of manual labour and trial and error to place them by hand. This precious time we didn't have while working on several other projects. Regardless, as we saw no other solutions, we tried anyway, hoping that the damage was not too bad.
image.png
image.png
Obviously, despite trying, we realized that manually placing the structures is a desperate last resort rather than the preferred solution. Unfortunately for us, we would be returning to this mod again.

Attempt no. 2
Generate the structures on existing terrain. Aka reverse-engineer Minecraft structure generation.

This was the original idea; however, realizing its sheer complexity made us try manually placing structures first. That's how bad it is.

In theory, it's relatively simple: scan the terrain for places where a structure can be placed; when an area like this is found, place the structure and slightly smooth the terrain. Of course, there are several "small" details that people don't think about until they are actually making something.

In terms of issues, it's hard to say where to begin. So I'll just list them from most memorable to least memorable.

1. The mod needed to know what structures can be placed where. So, for example, we do not try to place an ocean temple in a plains biome above ground. This was fixed relatively simply with vanilla structures; with modded ones, not so much, as sometimes modded structures are not coded the same as vanilla ones, which basically means they do not provide simple information such as "can spawn in X biome," "can spawn at X height," "can spawn at X elevation". This means for the vast majority of structures we had to manually verify where they can spawn and under what conditions.

image.png
image.pngimage.png

2. Terrain smoothing was not nearly as easy as we originally expected. We tried different approaches, even getting as desperate as to code a direct implementation for the WorldEdit

//smooth command. It worked. Kind of. It was easily the best way to make it work, and even then it has issues. Eventually James found a solution to the smoothing problem, and we could move on.

image.png

3. Villages are so much more complex than anyone could have expected. Nobody thinks about them; however, Mojang put so much complexity into their generation that no wonder most of the time they generate oddly. Like most structures, Villages are generated before the terrain in vanilla Minecraft. They are made up of multiple pieces that all connect. To put it simply, villages are divided into 2 parts: the houses and the roads. The roads depend on the terrain; the houses depend on the road and the terrain, and all together they depend on each other. Normally, without fixed terrain, in the generation process, the terrain just morphs and bends to the needs of the village; however, unfortunately, we do not have such luxury.

Attempt no. 3?
There wasn't really a third attempt. I'd call it Misc. rather.

We often go back and fourth between the manually placing strategy and the generator. Depending on how hard the doomerism goes. We often encounter issues regarding structures so often that one day it feels like you made a break through only for it to get shattered the next day.

The manual placing strategy is fix, it is guaranteed to work. However, who on Gods green Earth would like to do that? So we experiment as long as we have the time. As long as the Map itself isn't done we are not under pressure to invent a magic fix for structures. However, the latest builds seem to work as intended with the generator with only minor flaws.

2026 January - July

Polishing of the work.

In January 2026 James completed the first map version that was testable. Many of you might remember this time as the BETA testing. The timing was convenient, as I also finished some builds of Haste and other features.

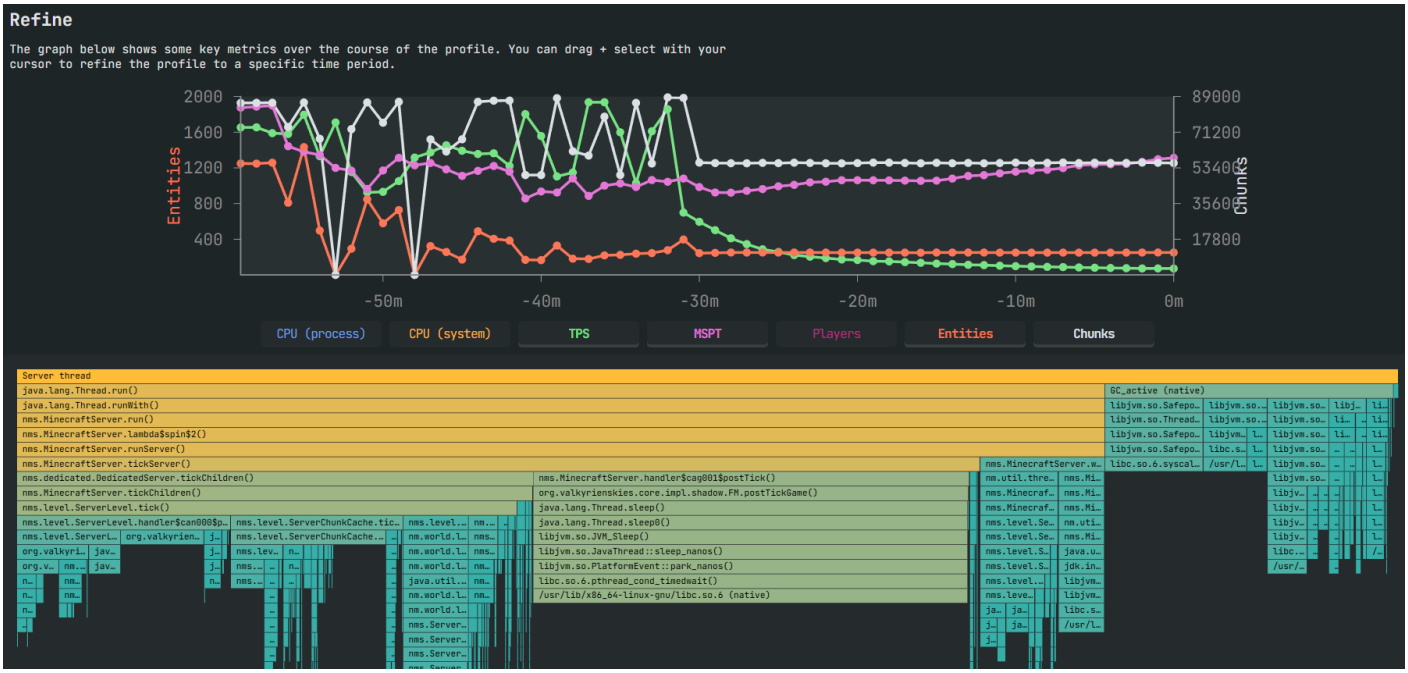
During the BETA testing there were several issues that arose, this was also the point of the BETA testing. We knew the server wasn't done, we wanted to know what was wrong.

The testing revealed that Haste was still not ideal. It revealed that the Server doesn't feel immersive. It revealed that the map was too simple. 3 Separate issues, with severe underlying causes. Let's understand them.

Haste

Originally the symptoms from the BETA testing didn't reveal any conclusive reason as to why Haste was ineffective. It pointed to ticking among others. Which basically means it can be anything and good luck.

More digging revealed a Heap issue which was visible after some time.



This is normally due to a memory leak. The reason was that Haste's cache kept collecting Asynchronous entities. This resulted in a build-up of cache memory, which eventually ran out. This isn't like normal out-of-memory errors. This one doesn't crash the server, it silently kills it and all activity on it. This can be seen on the Green TPS graph on the above picture.

Unfortunately, to diagnose this issue took weeks of trial and error.

Lack of Immersion

Watching the players on the server and being with them made me realize that while the server is built on realism it's missing one crucial aspect of it.

The ability to see far.

Due to Server performance, actual chunk render distance can get low to ensure the server stays playable. In very severe situations it can get as low as just 1 single chunk. However, in those cases it's only to free up memory for an emergency restart. The average actual render distance depends on the activity and concurrent server pressure. Without sufficient data I am unable to say an average render distance, but if I'd need to guess I'd say 2-3 chunks would be normal. However, that is nowhere near enough to properly see and immerse yourself in the game.

Around this time, mods like Distant Horizons and Voxy were starting to take off. However, with Elysian, it's never just a simple "add the mod to the modpack". Voxy for starters, is a fabric mod, we are on Forge. Distant Horizons is known to have performance issues. In our situation where we are already struggling to maintain client side performance that was an immediate no.

Other reason's for the lack of immersion was to the predictable nature of the Server. There are no sudden terrors or others that players would need to suspect. This time I remembered that I originally designed the concept of a mod called WorldEvents that does exactly that.

Furthermore, a seemingly random appearance in the world also seemed to disconnect everyone from the realistic focus of the world.

I began working on mods to enhance the QOL and overall feeling of the world.

Let's start with the first-join sequence.

https://www.youtube.com/embed/4aVveLK3XbE

On top of the First-join sequence, we have also made unique one-time ticket replicas.

https://www.youtube.com/embed/IC3W5NvSb-o

These ticket replicas have a unique ID number which is visible in its render, each player gets their own number.

Furthermore, we have a tab for age achievements.

https://www.youtube.com/embed/zBd3fnIMW-4

To solve the problem of render distance, after many failed attempts at quite literally recreating Voxy, we opted to just make a hook mod to make Voxy function on Forge. This is the result. **Please keep in mind, that due to me recording my screen the FPS is lower than normal.**

https://www.youtube.com/embed/nm0NyBnW8FY

Voxy can go up to as many chunks as you like, it's limited by only your Computers specs. In the video, I have my Voxy view distance set to 80 chunks. These chunks are sent from the server directly to your client, so no download's needed. However, if your wifi is unstable, it is possible to locally store these Voxy chunks.

I have added custom modes that instead of relying on fixed settings automatically adjusts your chunk digestion rate and view distance based on your current FPS compared to a "goal FPS". So for example, if your FPS allows it your client isn't capped at 80 chunks and can keep going as long as your FPS is stable. This is also helpful when entering laggy areas, as when that occurs your chunk digestion effectively stops to allow your client to deal with the lag.

Our custom Voxy implementation also has dedicated modded block processing, something that Voxy by default, doesn't do. What this means in essence is default Voxy either doesn't render, or renders modded blocks incorrectly, this is a known limitation. Our Voxy on the other hand has been improved to allow some of the modded blocks to be rendered correctly. For example: Domum Ornamentum, Framed Blocks, Dynamic Trees.

Bonus:
I am currently working on an addon that would allow people to edit a global map using Xaero's World Map. Nation Leaders would be able to draw their borders, roads, among others. This would show as a map layer that players can toggle as they like. Everyone gets this map and it gets globally synced. Since this mod is only in the early stages of development I am unable to show any screenshots.

The Map's simplicity

Playing on the map didn't feel special or unique. Unless you were told it's a custom map, you would never realize that it is. Large seemingly endless biomes made it feel monotone and the overall lack of detail made the individual biomes feel empty.

The map being finished was a major milestone. However, in it's current state, it didn't live up to the expectations.

Unfortunately, due to this I had to ask James to resume work on the Map and to make it more immersive. He agreed and started on the refining process.

The above picture is of the old map, before it's revision.

James overall has done a great job at the map. He is currently working on the Map which is nearing it's final polishing touches. The map is our final main blocker to release.

The Launcher

We created the launcher as a side project over the course of about 3 months. Mainly I worked on the launcher, but recently it was completed.

The launcher itself is a custom fork of another launcher called Helios Launcher. As they allow it, we have forked their launcher and greatly modified it to ensure it fits Elysian. This was very time efficient compared to going through the trouble of building one from scratch.

The Launcher itself uses official Microsoft Authentication and official Minecraft API. Both of which we needed to apply for and got approved.

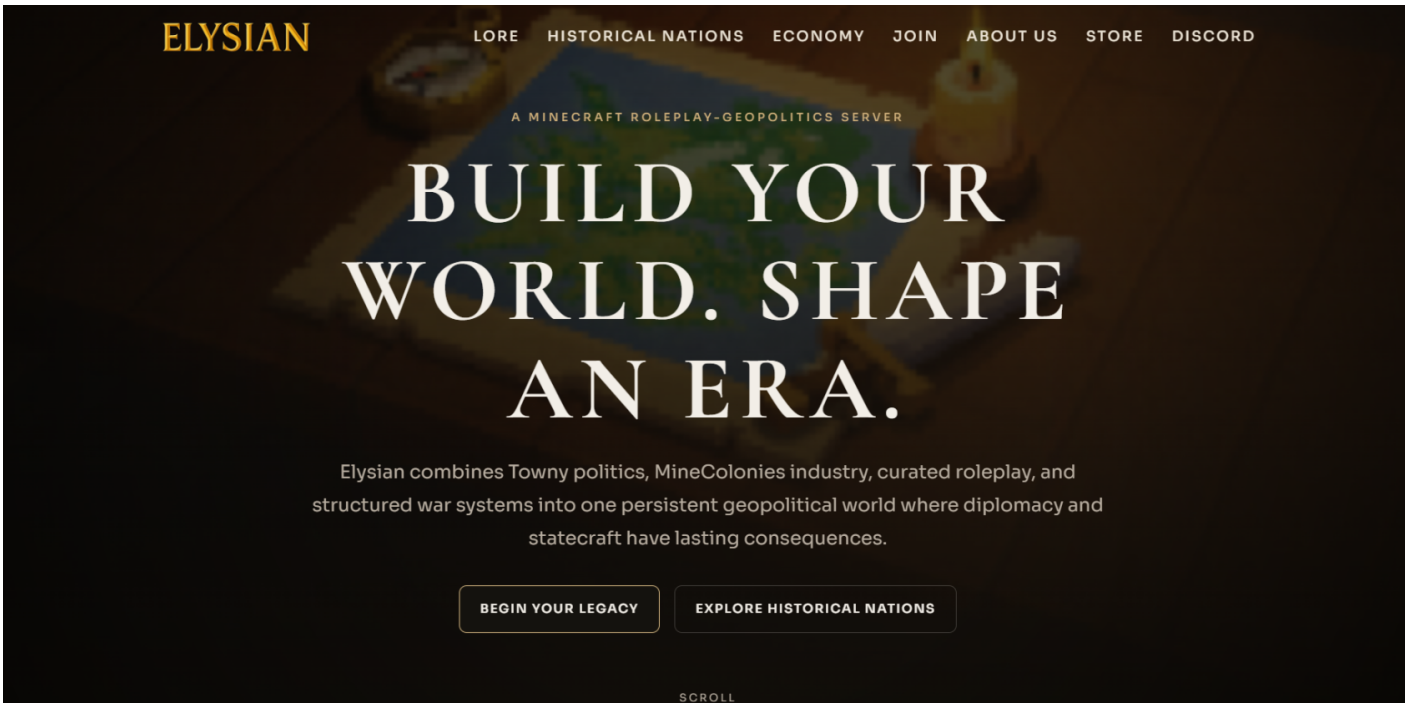
The Launcher as of right now is fully functional, as we have completed it's initial development phase. Only legalities, such as the creation of a Terms of Use, Privacy Policy, and EULA remain for which we need to hire a professional.

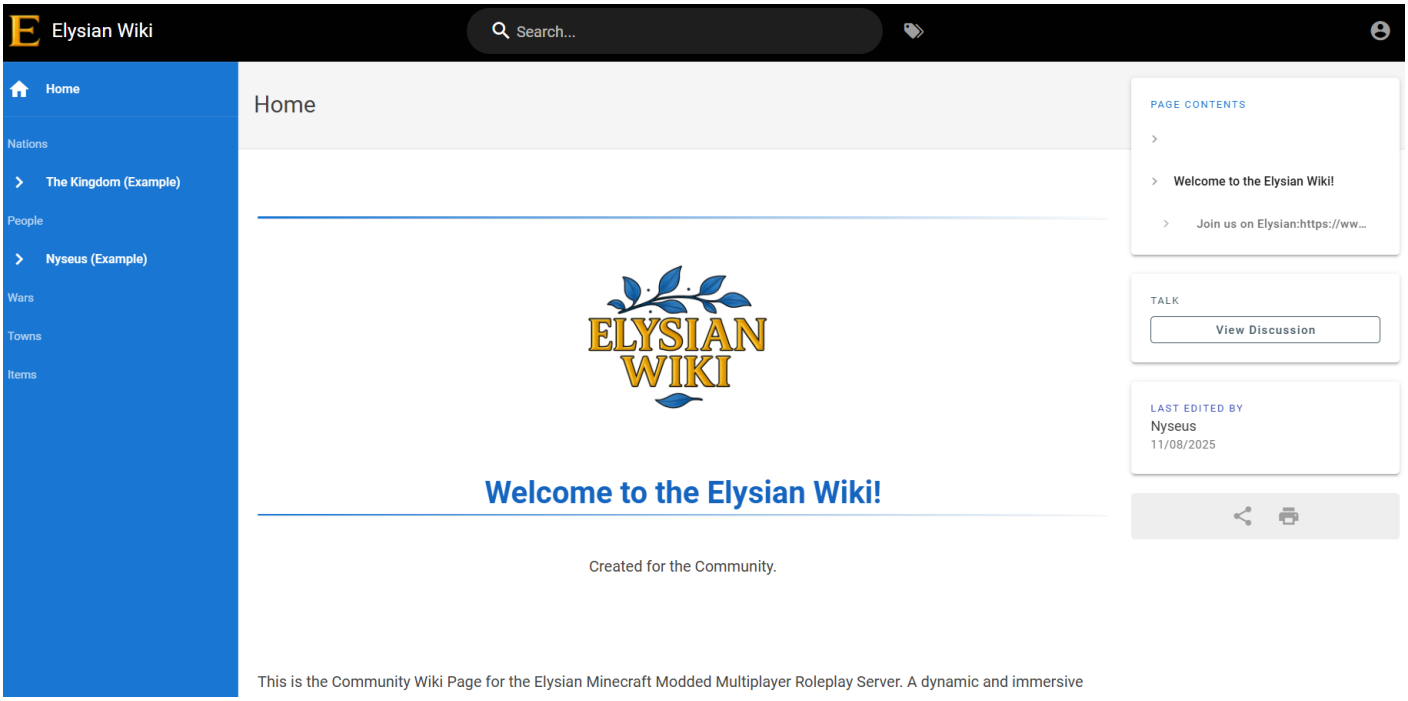
The Elysian Website(s) and API

During this same period, I have made a vast collection of websites:

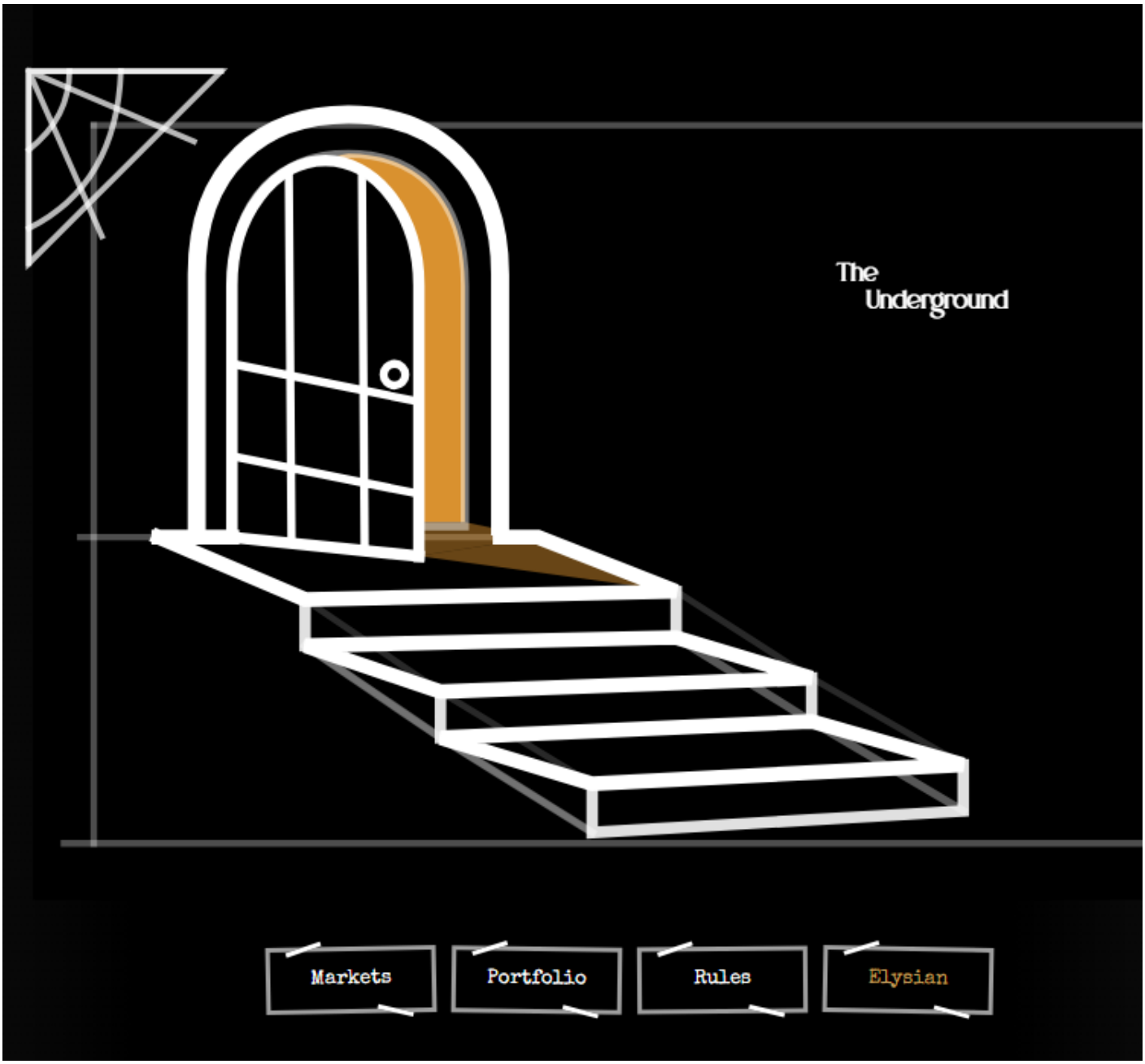
- The Main Elysian website: https://www.elysianmc.world/
- The Elysian Wiki website: https://wiki.elysianmc.world/
- The Elysian Info website: https://info.elysianmc.world/
- The Elysian Staff website (not public): https://console.elysianmc.world/

Among other domains such as: wings.elysianmc.world, launcher.elysianmc.world, and api.elysianmc.world.





Currently, as a side project I'm also working on another website called the Underground.



I didn't want to share many details about the Underground, but think of it as a betting website regarding events of Elysian. Kind of like Polymarket, but strictly only with in-game currency. We do not condone actual gambling. This website will be available once it's done as a hidden easter egg located at the bottom of the Main Elysian website page, and will also be reachable via: underground.elysianmc.world.

The Underground works with your Username only. You provide a Username, the website talks to the Server, if you are online on the server you will receive a 2FA code that you will need to provide in the Chat that only you can see. You can only obtain this code if you are able to login to the Server itself with your official Minecraft account. If you are not online, no code is sent. The code itself is hidden, only revealed if copy pasted from the Minecraft chat.

Thank you for reading the development summary. I probably didn't include everything, as it's too much to write down.

-Nyseus

Revision #2
Created 21 July 2026 12:22:40 by Nyseus (Owner)
Updated 21 July 2026 17:28:27 by Nyseus (Owner)