Blog for Dungeons and Crawlers

Team: Caolan William Cochrane - 22490802
Ethan Doyle - 22497082

October

Decided on what we would do for the project: a procedurally generated voxel game in the web. This was only the initial idea/stage of the project, we had not thought about any AI implementations yet. After a couple of emails to possible supervisors, we got a reply from Mark Humphry, telling us to consider using things like AI in our project, which was when we refactored the original proposal into what it is now.

November

2nd: Wrote the finished & improved project proposal. Emailed it to Mark to see if he would be our supervisor. He agreed. Set up a meeting with Mark soon after.

26-29th: Wrote the initial functional specification for our project. Talked about the system architecture, high level design and our minimum viable product which we decided on. We aimed to get: LLM integration, a physics engine, multiplayer and 3D graphics, textures & rendering. We also hoped to get a simple quest system implemented with some AI dialogue for NPCs.

We also decided the major libraries & resources we would use for our project. We decided to use Three.js for 3D graphics and rendering, LLama3.2-1B as our AI LLM model alongside Node.js, Vite and Express.js. We also decided to use noise for terrain generation - it would generate a 16*16*16 array of block IDs to render chunks with that it would pass to the frontend.

December

1st - 10th, 26th - 30th
We began deciding on even more features and components of our project. We wanted to implement registry systems for different parts of our components. Modularity arose from them that we did not expect - we built the registries in such a manner that the scope of the project changed. It was now possible to get biomes generated by the LLM - and we aimed to achieve this. We wanted biomes and blocks to be defined as JavaScript objects as the plan was for NPCs to follow suit as well. This can allow the LLM to write lambda functions that can be passed to the registry for handling. We just expose fields for the LLM to use and

hopefully down the line we would be able to make it so that the LLM can write actual functional systems for interaction and puzzles.

We eventually decided on perlin noise over simplex noise for our noise terrain generation due to the ease of implementation.

We delayed biome blending logic until bit as biomes require smoothing out at transition. We also delayed the onset of the third dimension to biome moisture, temperature map so we can add a depth parameter which will let the LLM define biomes based on depth below and above the surface as well.

Structures and NPC relationships needed more thought before implementing them, so we decided to leave them until way later.

We wanted client-side prediction to be introduced on the frontend first, for ease of implementation, then have the system be ported over to the backend for verification.

With all of these developmental decisions in mind, we started the development of the project.

<u>January</u>

24th-31st
Started development on frontend systems. We chose our physics engine - Cannon-es.js. We chose Cannon by comparing it to other physics systems. It was the easiest to implement and to use, so we believed it to be perfect for our project at the time.

We developed a player class which rendered a block onto the scene. This block took in inputs from an inputmanager class which contained a list of booleans for each direction. The block’s velocity in said directions, if the boolean was true, would increase. Eventually, we implemented this direction to work relative to the camera, ‘forward’ would be what the camera considers as forward, and so on.

Initially, using Cannon, we thought that we should have more precise collision detection, so we tried using bounding box methods to increase the accuracy. However, we eventually found that it was better to use the base engine.

We set a camera using an isometric style that could be rotated using the arrow keys, with inputs gotten from inputmanager similar to the player class.

We decided to create manager classes for the player and camera that handled their movements and rotations.

<u>February</u>

1st-21st: The final stretch of project development.

We found out that Cannon-es.js led to too much of a performance overhead. This was during the implementation of rendering the generated terrain in the frontend - having to add a physics body to each chunk was extremely taxing. Because of this, we made the developmental decision to switch to a manual physics implementation through making our own engine. The manual physics engine would follow the same logic - a broadphase which detected possible collision detections through an offset around the player object, a narrow phase which detected actual collisions through the contact point, overlap and collision normal, which then used these values to resolve said collision. It would also implement gravity. Additionally, removing Cannon from our project also meant that we had to change our player movement logic, as before we were relying on Cannon-es.js's built in .velocity value.

Even after getting rid of our plans to use Cannon-es.js there was still an overhead during terrain generation. We found out this was because of how the chunk was being rendered - creating a new mesh for every single block was not the way to go. We switched to using instancedMeshes and everything became much smoother. Since we were using instancedMeshes we had to find out a new way to apply textures from our texture atlas - which we decided on during this final stretch - to each block in the chunk instance. The difficulty with this is that all blocks within an instancedMesh contain the same material. To remedy this, we found out we could use a shader material which took in a list of block IDs and calculated which texture to use based on it. Each block ID would set the column and row variables to a different float number, in which it would then calculate the UV coordinate rectangle for the texture.

We also calculated the block normal to apply textures to different faces of the block.

We began making the final deliverables for the project using new components of our system - the technical specification, which would reflect our project as it will be in the demonstration, a user manual for how to play the game, and a video showcasing the major features of our system.