Minecraft2D

A 2D version of the popular java game: Minecraft. Built as part of a graded assignment.

Description

The game is a 2D open world game with realistic lighting and procedural terrain generation. The terrain is split into chunks, which are 2D arrays of blocks. The player can move around and interact with the world by placing or destroying blocks, which impacts the lighting and is saved for when the game is launched again in the same world (using the same seed). The most advanced features of the game are its procedural terrain generation and lighting system, whose development process is described in detail below.

The Lighting System

The lighting system works based on all objects within the game. Light sources present within the scene have been assigned coordinates and intensity values, which enables the game to draw shadows dynamically.

The solution relies on a technique I learned on this page, which dives relatively deeply into simple optimizations of ray casting algorithms. I learned that it is not necessary to cast rays and intersect them with every object, but it suffices in the simple 2D environment to find the edges of objects and take only them into account. This approach significantly improved the efficiency of the program while also increasing the accuracy of the shadows.

In order to be able to use the technique described above, I needed a way to, rather than looping over each edge of every block in the world, search for edges of rectangles consisting of adjacent opaque blocks. To optimize this algorithm even further, this calculation is performed only once the chunk is interacted with by the player, instead of every frame.

During the development process I used sources such as AWT docs and the Graphics2D documentation to find ways to shade the terrain. In order to handle lighting, I learned how to place a independent on top of another by

reading about JWT Panels here. This way me and my colleague were not interfering when working on independent parts of the project but utilizing the same terrain object.

My first approach of utilizing a radial gradient quickly ended up not working when more than one light source was applied and I had to find a different way to map the darkness and brightness of the environment. The second approach was far more accurate. It included a 2d array of pixels representing the screen. Every pixel was assigned a value (incremented) per light source in the frame. More factors were taken into account such as the distance from the light and its intensity. If the pixel was within the shadow of the light, its value was not incremented. In order to perform this calculation this source suggested utilizing objects available on the AWT class such as Point, Polygon and its contains() method (description in documentation).

The Procedural Terrain Generation

The terrain generation is split into several subcategories. The terrain is procedurally generated based on a seed, and can be regenerated at any time based on that very same seed.

The initial approach taken revolved around a 1D Perlin noise generator and shifting the start x value by a constant, acting as the seed. With mixed success, I then experimented with other libraries including a 2D simplex noise generator (improved Perlin), where I could use the y dimension as the seed (with some adjustments).

Generating biomes was initially done by randomly assigning a chunk to a biome, applying a modifier to the basic noise (such as smoothing it out or shifting it), and then interpolating between chunks with different biomes. This worked and could produce varied terrain, including mountain ranges, plains, and oceans, but the transition between biomes was often unrealistic. I instead opted for a combination of noise maps, where I overlay them with increasingly more detail. One acts as the biome noise map, and the other as the base terrain. This results in biomes with mountainous regions, and others with oceans, or plains and lakes, deserts, etc... This approach was suggested here was cleaner looking and more effective.

To save chunk states after being interacted with by the player, I needed an external file system to store data in. I considered csv and Json (found here). Csv had the advantage of taking slightly less space but was harder to parse and harder to modify whenever

changes were made to the chunk class. Json on the other hand could store info in a key-value format, which was easier to adapt to the increasing chunk class complexity. The chunk's offset would represent the key and the tiles the value. Each seed has its own save file.

Tiles Textures & Inventory Assets

https://piiixl.itch.io/textures

Simplex Noise

GitHub Source Link:
https://github.com/SRombauts/SimplexNoise/blob/master/references/SimplexNoise.java

JSON Simple

https://code.google.com/archive/p/json-simple/

https://github.com/fangyidong/json-simple