



VOXEL TERRAIN

Project

ABSTRACT

This document aims to describe the game component along with the surrounding project. It begins with a description of the intended game, followed by a closer look at the project structure and code architecture. The document concludes with an overview of the game component and its relationship to the rest of the project.

Niklas Johansson
a24nikjo

The Game

Due to me being the sole developer on the project, the game is more of a limited prototype. It can however be useful to discuss the intended game that the prototype would become if considerably more time had been put into developing it.

The intended game is like Minecraft, a sandbox survival game where the world consists of blocks. The player must explore a vast and diverse world to gather resources and build new and better things. Enemies try to hinder the player and make certain areas more dangerous. Better gear needs to be crafted to fight off these enemies and to dig deeper and discover rare minerals. There is no real end to the game and only the imagination of the player sets the limits.

This description could just as well have been used to describe Minecraft. The purpose of this prototype is not to make a commercially viable game but rather to explore the various challenges that come with voxel terrain. It provides knowledge for future projects that use voxels.

Project Structure

Since the project is quite small, a simple structure has been used. The project is a Gradle project which is version controlled with Git. The Git repository is hosted on GitHub. The Gradle project contains only one project and no sub-projects. The single project then contains a single Java module that contains all the code.

The code is split into two main packages. The engine package contains an abstraction layer over low-level functionality and the game package contains the code for the actual game. If the project were to grow, I would probably separate these two packages into two different Java modules and Gradle projects. This would make the separation more distinct and allow for greater control over what parts of the engine are accessible to games.

Code Architecture

The architecture of the code is built around the game side and the engine side. The engine side employs a small central skeleton which can then be extended with modules. These modules are not Java modules but rather a subclass of an abstract class. Each module is handled automatically by the engine. The base module class contains abstract methods that get called when the module loads, updates and unloads. The modules are placed as public static final variables in the engine class. This makes each module accessible to the game through the static access of the engine class. Examples of modules include modules for rendering, assets, input and display.

I find this modular approach with a lean main engine both scalable and easy to work with. The main engine can be kept simple while modules can add complex features. The modules can also be developed in separation and can be switched out, turned on or turned off. This results in a very flexible game engine.

The game is then a subclass of a base game class. It is this class that the engine uses to tell the game when to start, update, render and stop. The engine skeleton correctly loads modules, starts the game, updates, renders, stops the game and unloads modules. To build a game, one simply creates a subclass of the game class and takes advantage of the abstract methods that get called by the engine. There is no need to create a complex game loop as the engine takes care of that.

The Component

The component that was chosen for this project is the voxel-like terrain. It is a fundamental part of the intended game since it makes up the entire world and all other mechanics revolve around it. The terrain also takes up a lot of the development time due to it being required and technical. It would be hard for me as a solo developer to choose a game component other than the terrain. I would then need to develop both the component and the terrain which would not be feasible due to time constraints. Besides, the game does not need much more than the terrain to be playable. It is for these reasons that I believe that the voxel-like terrain makes a good candidate for the game component of this project. I also wanted to explore games built around voxel-like terrain which adds to those reasons.

The component can then be divided into three main parts: The rendering of the terrain, the generation of the terrain and the interaction between the player and the terrain. Most of the focus has been on the rendering part but this does not mean the others have been ignored. They are all key to achieving a usable terrain for the game. The component is also very scalable since it consists of a basic implementation that can then be extended with any number of features. These features are relatively separate and can be implemented in series which allows for a more dynamic approach to planning. If some features take less time than planned to implement, more features can be added to the to-do list. In the planning phase of the project, multiple such features were listed. Some of them were labelled as extra and would be implemented only if time permits. Unfortunately, no extra features were implemented due to the limited amount of time.