Building to 3d Minecraft Model Conversion Software

Report and Documentation

Zaniken Gurule

Contents

Capstone Report

This is a graduation project meant to demonstrate some of the technical skills that I have acquired throughout my Computer Science bachelor's degree. In addition, this project is meant to contribute to my universities ambitious plan of having virtual tours in *Minecraft*. The project consisted of creating a package of software that takes real life images of buildings and converts them into a 3d *Minecraft* buildings. The project combined concepts from a variety of computer science topics such as algorithms, file structures, and software engineering. The results of the project were successful and can be measured by the achievement of my original objectives; my objectives, software engineering documentation, literature review, and a personal reflection are detailed below.

Main Objectives

Create an algorithm that calculates the average color of a BMP image.

The need for an algorithm to find average color density was found in two areas, first to find the average color of a Minecraft block. Second, to find the average color density in small sections of images (in this case photos of walls). At first, I thought I would need to manipulate the values of individual BMP files, which would have been a burdensome endeavor, but I was able to find resources on Java's built-in image manipulation tools that saved a lot of time. I also used Java's Color classes. The resulting algorithm is a simple double for loop that iterates over each pixel in a buffered image (or section of a buffered image) taking the red, green, and blue color values and adding them to their own individual sums. The total sums are then divided by the width*height of the image and returned as their color value of the total average color.

Create an algorithm that can use average color densities of sections of images to find their Minecraft block equivalent.

After getting color averages of blocks and image sections I then needed a way to compare two colors. The answer I was looking for was utilizing the distance formula for color values. As colors are represented by their RGB values you can take the square of the intensity of red A minus red B and add that to the same formula for the green and blue values. If colors are very similar the number will be small if they are very different, they will be large. Using this algorithm, I loop through every *Minecraft* block and compare the color distances, swapping if the distance is lower. This process is used to create a 2d array of *Minecraft* block equivalents that can be used as individual walls.

Figure 1) A demonstration of an image being converted into its average color tile representation and then into its *Minecraft* block equivalent. This was done using my software.

Be able to trace the outline of a building from a google maps screenshot to get the dimensions of a building.

A goal of mine was to have the scale of buildings translate to *Minecraft* I found that a one-to-one ratio looked bad and settled on a 5 to 1 meaning one meter in real life would be 5 meters in Minecraft (1 block = 1 meter). For the sides of my generated building to be the correct length and be to scale I require the importing of a google maps image of the building. Google maps provides a meter bar that can be used to measure the building. I take a user text entry of the meter scale number and then have them click each end of the meter bar. Separate from the GUI these inputs calculate the number of pixels on the image that would equal 1 meter. With this information I can then have the user outline the building clicking on each corner in a clockwise manner. The building dimensions are then stored with their corresponding wall number. At this point it is of note that my software is only capable of converting rectangular shaped buildings due to the exponential number of variables of non-rectangular buildings and my own time and energy constraints.

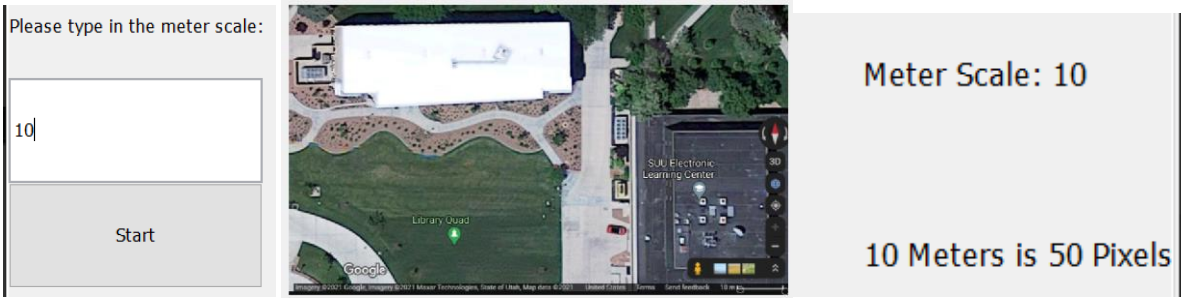

Figure 2) A demonstration of a Google maps image being used to find the size in meters of a building.

Convert the processed information into the schematic file type that is usable by 3rd party Minecraft building software.

I now had the problem of knowing what blocks each of my building walls were going to be made of in addition to the size of each wall, but I had no way of making that in *Minecraft*. I ended up researching and reverse engineering the file types used by *Minecraft* and its third-party software. NBT is a custom file type created for *Minecraft* to store block data and location data, WorldEdit (a 3rd party world editing software for *Minecraft*, the tool I used to paste my processed buildings into Minecraft), uses their own version of an NBT file called ".schematics"; there is very little documentation on WorldEdit's schematic file type and to make things more confusing other *Minecraft* editing software have their own incompatible versions of schematics. To replicate the files made by WorldEdit in my own software I used file analysis techniques including creating files using their software and analyzing them with a hex editor gaining an understanding of the general file structure and deconstructing the WorldEdit source code and isolating the relevant code. I also took advantage of JNBT which is a Java library that handles the actual writing of the NBT file, but not the file structure. I was able to compare the files I was making with JNBT with WorldEdit's own schematic files until they eventually matched and were compatible.

Figure 3) 2d Images being converted into their 3d Minecraft equivalents.

Create a user-friendly GUI that allows for quick and easy use of the software.

As this project was programmed in Java I wanted to use libraries that were native. At first, I tried to use JavaFX but found that Java has begun to phase out JavaFX from the default JDK, so I ended up using Swing. Swing worked and I got the features that I needed out of it, but it was not ideal. “user friendly and easy to use” are subjective terms, but overall, I believe that the GUI is usable and useful, this project would be near impossible to execute using only command line.

Secondary Objectives

Be able to use a meter stick in a photo as a size reference.

In addition to the measurements taken from Google maps images which provided the buildings width and length I needed a height measurement. My first thought was that I would need to place a meter stick in each picture to be able to get the scale. However, I realized that most doors are almost exactly 2 meters tall so I could get a close enough estimate using them. Furthermore, working under the assumption that each wall of a rectangular building is the same height I was able to require only one height measurement from a singular wall.

Create functional requirements document.

This document can be found in the same folder as this report.

Create class diagram.

The class diagram I created is a high-level overview of how each of my classes interact with each other. I worked hard to keep the GUI front end separate from the information data processing backend. The document can be found in the Software Engineering section of this report.

Create a Model Justification.

This document can be found in the software Engineering section of this report.

Integration testing with the SUU server (applying generated buildings onto their map).

The SUU *Minecraft* server is being continuously worked on so I have yet to implement my designs. The schematic files I made can be used on any *Minecraft* world and can quickly be saved onto the SUU server at any point. While my software is inept at converting some of the abstract buildings on campus such as our library, it is well suited for converting many of the buildings located along Cedar City’s Main Street which are more rectangular.

Figure 4) The bizarre results of converting a non-rectangular building using my software. Image on the left is SUU’s library and the image on the right is my *Minecraft* conversion which created the illusion of depth while being completely flat.

Tertiary Objectives

Begin the process of converting campus buildings into their 3d Minecraft equivalents using the developed software.

I tested this software on several of the buildings around campus with good results. Most of the images of buildings in this document are campus buildings.

Expand the software to allow for less “rectangular” buildings such as the library.

I was unable to achieve this objective as many of the techniques I used simply do not convert to non-rectangular buildings. I do believe with the knowledge I have gained over the course of this project I could do this, but it would take a lot more time.

Allow for non-BMP type images to be used.

This objective is a remnant of my ignorance to Java’s built-in image tools, as I first thought I would need to manually manipulate BMP file data. By using Java’s image libraries, I believe my software can handle any image format, though in practice I have only tested BMP, PNG, and JPG file formats.

Allow for editing the algorithms block choices in case there are errors.

If I had more time this would be a top priority, but in the current state of the software it is unimplemented. Implementing block selection would allow for fixing many of the issues my software currently faces such as: shadows, window reflections, odd block choices, etc.

Software Engineering Documentation

Model Justification

I used the waterfall model of software development to create this project. I chose this model for a variety of reasons including: I started with clear milestones that I needed to complete that could be modeled using a timeline, I had limited resources/time and needed a model to push development quickly, and lastly because the waterfall model encourages documentation which is one of my outlined objectives with this project.

It is possible that other models would be better suited for similar software development, but given my circumstances such as timeframe and experience, I believe that the waterfall model allowed for a clear and easy to follow development process.

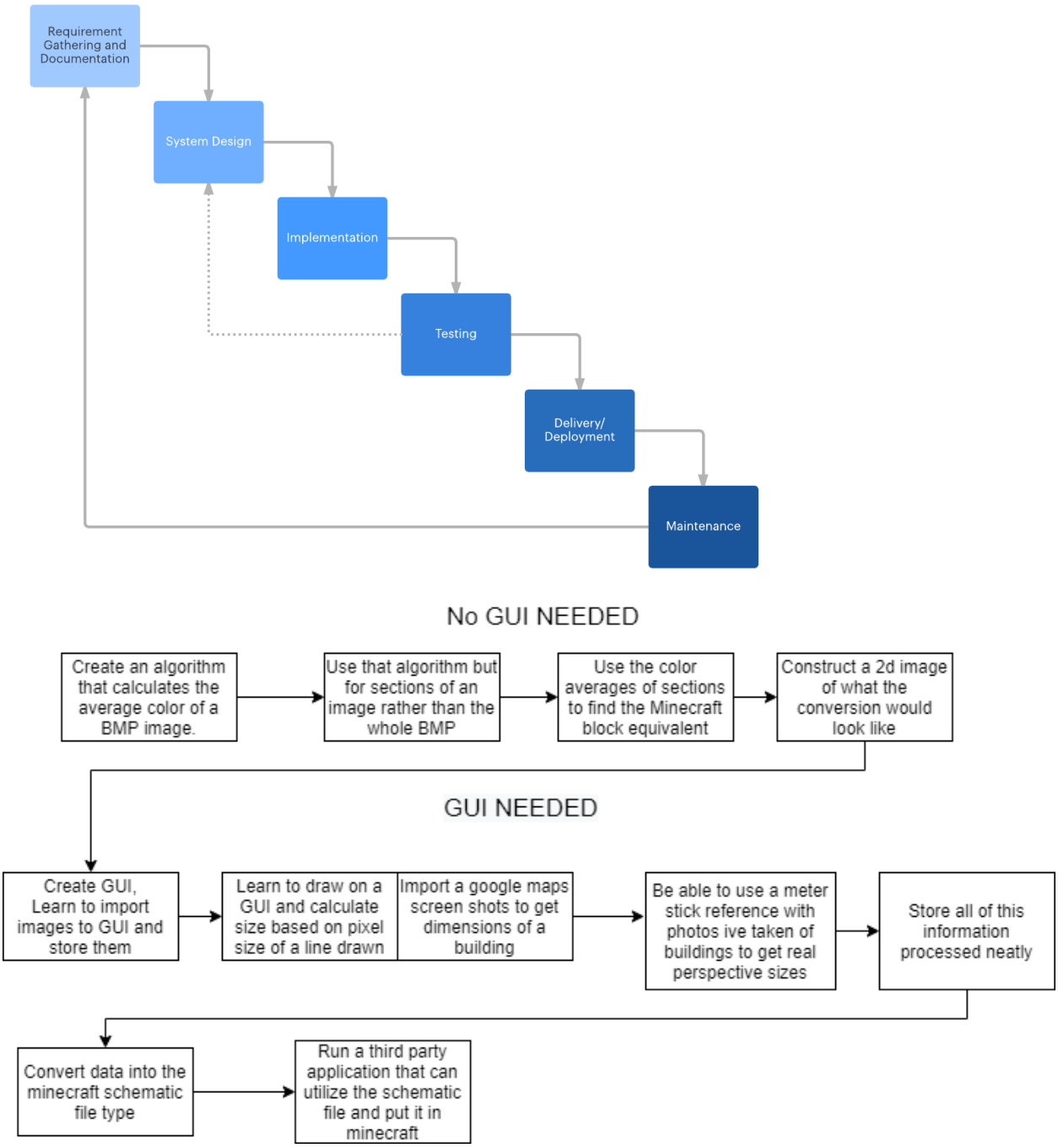

Figure 5) A generic waterfall chart and my specific timeline for this project (the timeline was followed fairly strictly).

Functional Requirements

The SRS document can be found in the same folder as this report.

Class Diagram

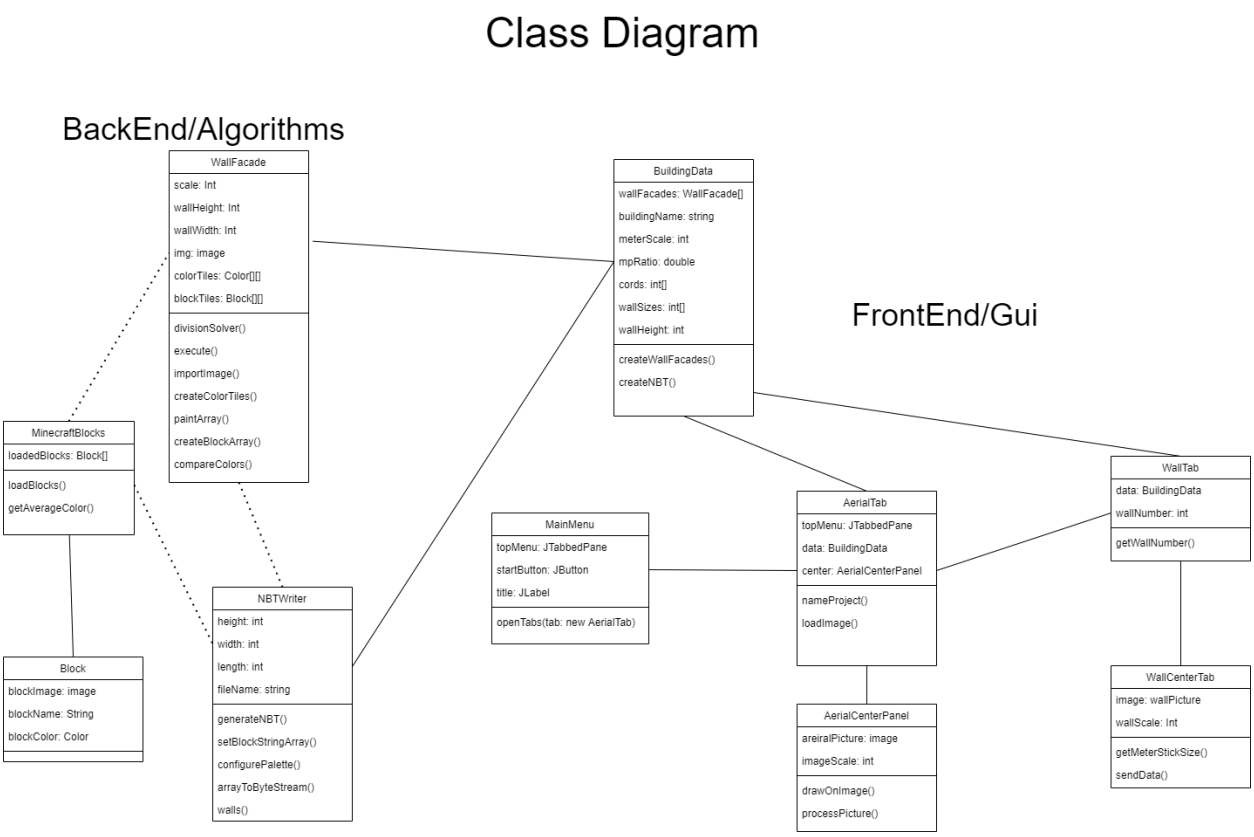

Figure X) A class diagram of my software application.

Lessons Learned

This project was a very big learning experience. This was the largest project that I completed during my course work. Over the course of the project, I ran into many problems, but with research and a lot of time testing out different things I was able to get a product that I am happy with. If I did this project again there are some things I would do different: First I would probably not use Java again at least not for the GUI. Java Swing was a nightmare to get to work and I am still not fully satisfied with my user interface. It is very difficult to have a good-looking Swing app and app portability is too limited with Java despite the “write once run anywhere” motto. I think that webapps are a much more portable form of software, so if I did this again I would probably make it in JavaScript and HTML/CSS. Second, I would spend more time outlining my class organization; I ran into a lot of problems trying to keep back-end and front-end logic separate, and I think that was preventable. Third, I ended up with a few too many “monolithic” classes that contained way more functions than is probably appropriate, this could have been reduced using inheritance.

Conclusion

I learned a lot from this project and I am happy with the results that I achieved. I achieved my primary and secondary objectives and many of my tertiary objectives. I created a piece of software that I believe to be unique; while many people have created *Minecraft* pixel art generators, I have been unable to find anyone who has tried to create 3d building generation software for *Minecraft*. I believe overall I followed good software design principles and was able to create something big from many little parts. Despite my own successes, I am skeptical of how useful my project will be to the SUU *Minecraft* team, I think they will have trouble replicating many of the buildings on campus due to their large sizes and odd shapes. However, if they also plan on making other parts of Cedar City software could save them hundreds of hours of work.

Literature Review

A tutorial on Java Swing which is the GUI library that I used:

> https://zetcode.com/javaswing/

An explanation of how bitmaps work and their file structure. This is important because I will be doing pixel analysis:

> https://medium.com/sysf/bits-to-bitmaps-a-simple-walkthrough-of-bmp-image-format-765dc6857393

The only documentation on schematic files:

> https://github.com/SpongePowered/Schematic-Specification/blob/master/versions/schematic-2.md

File IO for Bitmap:

> https://stackoverflow.com/questions/16475482/how-can-i-load-a-bitmap-image-and-manipulate-individual-pixels

I used this and a few other ideas to get my own color checking algorithm to work:

> https://stackoverflow.com/questions/28162488/get-average-color-on-bufferedimage-and-bufferedimage-portion-as-fast-aspossible/28162725

Comparing colors:

> https://stackoverflow.com/questions/15262258/how-could-i-compare-colors-in-java

Painting overlapping buffered images:

> https://stackoverflow.com/questions/20826216/copy-two-bufferedimages-into-one-image-side-by-side

Action listener passing

> https://stackoverflow.com/questions/33799800/java-local-variable-mi-defined-in-an-enclosing-scope-must-be-final-oreffective