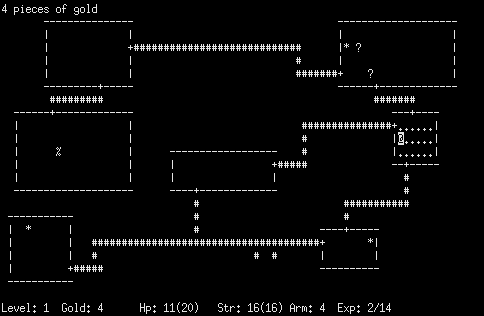
# Analysis

**Introduction**

Procedural generation is a method of algorithmically creating data. Procedural generation can be used to synthesize any type of media, but it’s most used in game development and experimental art and music.

**Users**

I’ve noticed that the field of procedural generation is difficult to access for people with less technical understanding, which is why I’d like to create a feature-rich procedural level generation tool designed for artists and game developers who want to design procedurally generated maps with ease and immediacy. There are similar tools that exist, but I feel that they lack character and don’t allow for much variety.



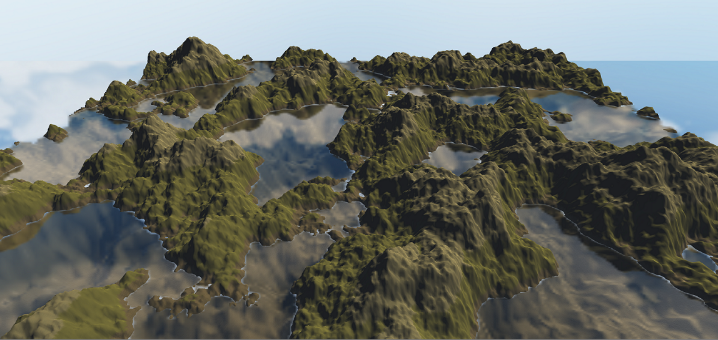
Rogue (1980)  
*Rogue is a great early example of procedural generation in a video game.*

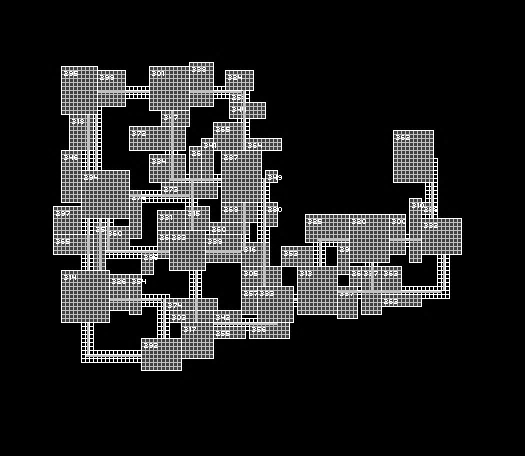
Throughout this document, I will refer to the intended users – artists and game developers – as ‘creators’.

**Concept**

I will design multiple 2D ‘top-down’ procedural level generation algorithms as part of the application. My focus will be quality of generation but also variety from a single algorithm. I love systems in games that encourage infinite re-playability, so in my development I will be conscious to write systems that create variety [while still being cohesive].

I would argue that the main two categories of procedural level generation are:

* Terrain Generation – which typically uses a noise map (but can also utilize techniques like wave function collapse) and can be infinite.
* Map

  Description automatically generatedStructure Generation (like dungeon/maze generation) – which can be done via several methods and is typically finite in size.

I intend to explore both methods of level generation, with the creation of interesting, playable levels as my focus.

I want all the algorithms to be seedable – meaning that the same seed (with the same settings) will generate the same outcome each time. This means that a save file\* won’t be too large.

*\*’save file’ within the database, not an exported version of the generated ‘world’.*

**Storing user data**

There will be user accounts, this will include password hashing and storing their worlds on a server (in a database). Users can have as many saved projects on the server as they like.

**Exporting level data**

While I intend for my algorithms to be able to create infinite levels, a file can’t be infinite! The user will specify the size of the generated level, and then export to a file in an easy to adapt format – I know that some developers have used formats like RGBA (a bitmap variation which would be quite easy to process by users), but I could also see myself using something like the ‘*Tiled Map Editor’* .tmx files, as they integrate both with LibGDX (the framework I plan to use) and Unity (a very popular game engine) or even .csv! Whatever file type I choose, my choice will be with maximum compatibility and ease of use in mind.

**Settings**

For the sake of user experience, any application settings will be associated with user accounts so that settings are saved and available wherever the user logs in.

**Similar Systems**

In the development of this project, I would like to consider other [similar] procedural generation tools as well as games that use procedural generation. I think that the variety should help me create a more effective outcome.

**Standalone procedural level generation tools**

Perilous Shores by ‘Watabou’

[***https://watabou.itch.io/perilous-shores***](https://watabou.itch.io/perilous-shores)

**Description**A web app for procedurally generating maps in styles that work for the creation of fantasy worlds and RPG games. It is heavily customizable and has the capability to be quite stylised whilst also retaining a lot of detail.

**What makes it stand out?***Perilous Shores* is easy to use whilst also producing a polished, detailed, complex outcome. In my opinion, its strengths come from its intuitively laid out interface that allows users to access and control the powerful algorithms that lie underneath.

Rather than being a tool for generating terrain, more generally, the app is designed specifically for the creation of fantasy maps. The app is highly specialised, and in turn highly powerful for its task, but this leads to it inflexible with quite a narrow scope of outputs.

I find I want to write algorithms that allow for a much wider scope of output than *Perilous Shores* does. In my opinion, it is only inevitable that this will lead to a trade-off of less precise power for a given generative task (e.g. the app wouldn’t be able to generate fantasy cartographic-style maps as well as *Perilous Shores*), but I believe that the flexibility is worth it. (the app would be able to generate a lot more things than *Perilous Shores).*

**Menus**The app’s menus contain descriptive but not distracting labels for each parameter, and an intuitive layout where controls are sectioned out and arranged logically. I really like how the user interface allows for in-depth access to the app’s controls while also being impressively unobtrusive. Its discreet nature allows users to focus on the creation process without anything getting in their way. I would like to take inspiration from this to inform the design of the app.

Diagram

Description automatically generatedGraphical user interface

Description automatically generated with medium confidenceDiagram

Description automatically generatedScreenshots

Diagram

Description automatically generated

**Games that use procedural generation for their levels**

Dwarf Fortress by ‘Bay 12 Games’

Graphical user interface

Description automatically generated**Description***Dwarf Fortress* is a ‘construction and management’ simulation game where you manage a colony of dwarves. It’s been described as ‘the deepest, most intricate simulation of a world that's ever been created.’. This is because almost every element of gameplay in Dwarf Fortress is influenced by procedurally generated factors. Its generation is so complex and interesting that Dwarf Fortress is in the Museum of Modern Art! (<https://www.moma.org/collection/works/164920>)

**What makes it stand out?**  
Procedural generation gives the player an entirely new experience each time they generate a new level and allows for a lot of variety in gameplay. To some extent this happens in any game that uses procedural generation but, due to the complexity of the generative model and range in what it can create, Dwarf Fortress does it incredibly well.

When making a new world, the player can adjust [a lot of] values that affect the size of the world - how often certain resources appear, the length of history of the region (meaning that more has developed before the player starts).

Terrain elevation, rainfall, mineral distribution, drainage, temperature and many other factors are considered – for example, a high-rainfall and low-drainage area would create a swamp. In my opinion, Dwarf Fortress is an exemplary use of procedural generation.

Dwarf Fortress’ strength lies in its use of complex mathematic models, with concepts like ‘Markov Chains’ (a method of mathematically representing states of evolution over time – past, present and future) giving the generated worlds life and character.

It’s likely that the worlds are stored in 2D arrays, with each data about every tile of the map stored at each position – likely a 2D array of custom objects, with each object having many parameters and methods. For example, it’s likely that each tile has a method that modifies the tile based on how it develops over time - taking the tile’s type, history, surrounding tiles and more as input to that method. All the stored metadata of the world allows generation to have depth and character, creating interesting and detailed outcomes.

**Aesthetics**As a programmer, I really enjoy the ASCII game art (although this is just the default art, and community-made skins are available). The simplistic nature of the art is satisfying and reminds me greatly of a ‘real’ map (a map of a real town, for example) as there is a heavy focus on symbols. I think I will employ similarly basic and obvious tile designs as it’s effective both aesthetically and also has the ability to convey a lot of information.

Text

Description automatically generatedGraphical user interface, text

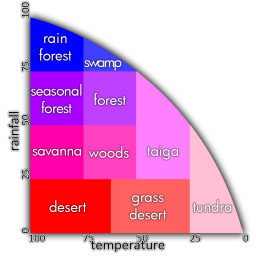
Description automatically generated

Minecraft by ‘Mojang’

**Description***Minecraft* is a sandbox open world survival exploration game.

**What makes it stand out?**The worlds generated in Minecraft have a high level of detail and variety and, in my opinion, this is what makes it so strongly replayable (and so popular!). Every experience is different, and it feels like the beginning of a new story with each new world. [Like some other systems I’ve looked at] Minecraft uses stylised pixel/voxel-based graphics. In my opinion, this is one of the best art styles for procedural generation as it lies in the centre of the goldilocks zone of being detailed enough for a wide range of variety while not being so complex that the procedural generation feels low quality.

**How does it work?**The terrain generation is procedural (generated from various modifications of noise functions), and the worlds are infinite. The worlds are made up of biomes, which have different aesthetics and contain different features.

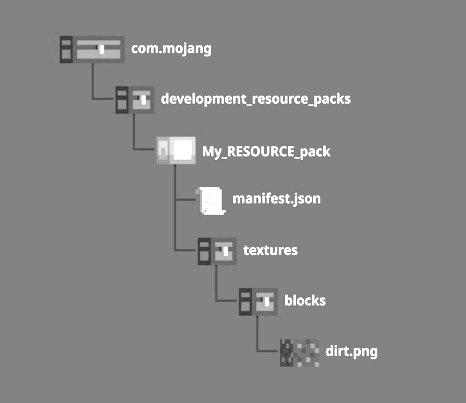
Minecraft has several layers to its terrain generation – the placement of biomes, then base terrain, then the world is populated with trees, etc. Although I will use a similar system, I won’t be able to bring as much variety into the generation of my game – Minecraft’s generation is detailed and complex, a result of the devotion of an incredible amount of time.

Minecraft uses a similar method to Dwarf Fortress for biome generation; for every ‘chunk’ of the terrain, a rainfall and a temperature value are assigned and based on these a biome is set – e.g. high temperature, high rainfall = rainforest. This is probably using a mixture of an upscaled (so that bigger areas are more similar) noise function to provide smooth in-between values and something like ‘Voronoi tessellation’, which is a method for procedurally generating interlocking irregular polygons. This added irregularity makes the biome-borders feel more real.

Once they’ve been loaded in, chunks are stored by an x and y coordinate, likely in a Dictionary or HashMap style structure, with the coordinates as the key and a 3D array (Minecraft is 3D) of blocks. As the player travels through the world, the game uses their position combined with a render distance value (how many chunks around them are loaded in) to get chunk data and render it.

**Resource Packs**Brought together by their enjoyment of the game, Minecraft has a community of artists making ‘Resource Packs’ for the game. These are partial or entire replacements of the game’s textures, sound effects and 3D models (and some other aspects too!). There are many sites dedicated to the sharing of and discussion about resource packs, and for many they’re the most exciting part of the game.

I really like the idea of being able to import your own tilesets to the app, and as my project is aimed towards artists and developers – who are the exact group that most enjoys Minecraft’s resource packs – I think I’d like to create a similar system in my project that allows for the import of custom tile textures.



Structure of Minecraft’s resource pack system, showing where the texture for a dirt block is saved.

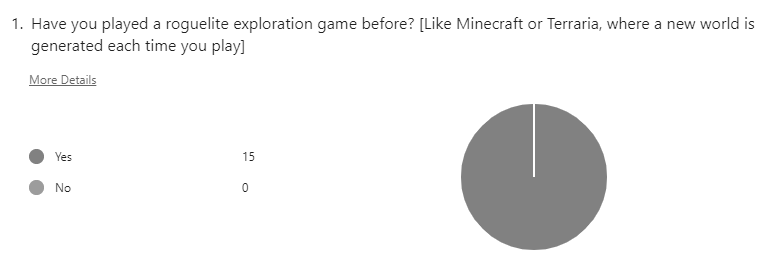
Minecraft uses a custom file type as part of its resource packs’ structures – ‘.mcmeta’ – but the actual data in the file is written in JSON, this makes it quite easily legible as well as easy to import and handle within the game’s code. It looks like Minecraft uses Gson - Google’s JSON focused [Java] library – to handle the JSON. (<https://www.minecraft.net/pt-br/attribution>) Another factor which improves usability is that each resource pack is in its own folder, meaning they can be easily shared (in a .zip).

**Survey**

I wasn’t able to get responses from game developers so I aimed my survey at individuals that play games that use systems like mine, to get a feeling for what makes level design interesting to players – and therefore what creators would benefit from including.

For each question, I only presented the most interesting responses (essentially because some answers were redundant). The quoted text in Consolas is people’s responses, the Century Gothic text is my analysis and thoughts.

1. *Have you played a roguelite exploration game before? [Like Minecraft or Terraria, where a new world is generated each time you play]*



The answers to this question show that every respondent to my survey has some understanding of the games that use similar techniques.

*2. If so, which ones and what in particular did you like or dislike about them?*

* “big open world is good”, “Minecraft - the open ended nature of the game”, “I like the freedom you get in Minecraft”, “I liked how much freedom the player has”, “I liked the ability to control the direction of my story, the pacing was controlled by me”  
    
  A lot of people mentioned freedom and open-endedness. This bodes well, as procedural generation allows for infinite level generation. In my tool, users will be able to create infinite worlds (with a limit of export size, where they choose a specific size of region to export).

* “there is a variety of things you can do”  
    
  I would like to include a variety of generative algorithms, with parameters that intentionally create a wide variety of outcomes – this would come from intentional design of the algorithms with this in mind, but also clamping the parameters at ranges that are wide but still very usable at any place within the range.
* “don’t make too repetitive”, “I disliked how repetitive they can get at times”  
    
  I completely agree with this opinion and (although it doesn’t directly apply) I plan to counter repetitiveness with my design of the algorithms. If I have time, I really like the idea of adding ‘biomes’ so that generated levels can have different areas with different settings.
* “great ability to customize your experience”  
    
  Again, I plan to design this tool with variety and customization at its core.

*3.* *In roguelite exploration games, what additional features of the levels do you enjoy?*

* “fishing” [x3], “all games need fishing”   
    
  I enjoyed seeing this response! Even though it doesn’t directly work for my project, bodies of water are fairly easy to generate. If I have time, I’d like to add the option for a layer to be a body of water – when users check that box, the option of changing the way that tiles connect, a lot like Unity’s ‘Rule Tile’ or the way that Godot handles TileMaps.
* “procedural generation of map/level”  
    
  Glad to hear it!

*4. What features of the levels would you really not like to see? Something that you hate in games like this?*

* “repetitiveness - make diverse biomes and make gameplay refreshing”, “repetitive gameplay with little differentiation”  
    
  I’ve addressed these already in my analysis of responses to question 2, but I wanted to show that they came up again.
* “tutorials which you can't skip”  
    
  I think that basic tutorial pointers are a useful feature, but I agree that they shouldn’t stop the user from actually using the tool. If I have time to add them, I will make sure that any pointers don’t hinder actually using the tool.

*5. Is there anything you really like/dislike about the user interface of games you've played in the past?*

note: I wanted to take influence from [good] video game UI and aesthetics as the intended users are likely familiar with games

* “Interfaces like on Terraria can have too many items on the screen at once.”, “non intrusive interfaces”, “[dislike] intrusive design”  
    
  A cluttered interface is irritating and makes the games/software harder to play/use and harder to understand. For this reason, and influenced by these responses, I will keep the layout and design of the interface carefully in mind.
* “consistent with game's aesthetic [implied as good]”, “contrast to the gameplay [implied as bad]”  
    
  These two responses talk about how the aesthetics of the interface should be cohesive with everything else, and I completely agree.
* “not particularly”, “no”, “nope”, “no”, “no”  
    
  To me these responses show that UI, while important, is less significant than the rest of the actual focus of an app. Based on these responses, I think that a decent proportion of gamers care more about if the game is enjoyable than its menus (which is understandable!). Influenced by this, I will prioritise effective algorithms over the UI in development.
* “I like having a coordinate system so that I know where i am”  
    
  I like this idea, and I will implement it if I have time (which I likely will, coordinates are data that will exist based on the game works anyway). I don’t want it to be intrusive on the screen though, so I plan to add the ability to turn it on and off in settings (and this links back to respondents preferring non-intrusive interface design too).

*6. Suggestions for the UI?*

* “possibly a minimap depending on scale of exploration”   
    
  I think that features that allow users to change scale of view would be great – like a zoom in/out feature.
* “maybe a crosshair so aiming is easier”  
    
  This doesn’t apply to my project, but it did inspire me to think about the different ways a user can control their character and interact with the world.  
    
  I think that movement will probably be controlled with WASD, but I really like the idea of using mouse input in ways that feel intuitive (clicking on things to interact with them etc.).

**Diagram

Description automatically generatedBasic System Flowchart**I have created this flowchart with what I believe to be an appropriate level of abstraction – though some parts aren’t fully defined, I am confident that the system can be understood without additional detail.

**Diagram

Description automatically generated**

**Proposed DFD of system**I’ve identified what I think are the tables I will need in my database, and the processes that write and read to/from those tables.

Diagram

Description automatically generated

|  |  |  |
| --- | --- | --- |
| **Storage** |  | **Process** |
| S1: User table – for storing all solely user related data | P1: Register – adding a new user to the user table |
| S2: World table – for storing global world-related information | P2: Login – enter details of pre-existing user and be checked against table |
| S3: Layer table – for storing layer specific information | P3: Change settings – get current settings and make modifications to them |
| S4: Tile mapping table – for storing how generated values are mapped to tile images | P4: New world – add new world to relevant tables |
|  | P5: Load world – get all details of world from tables |
|  | P6: Save world – update relevant tables |

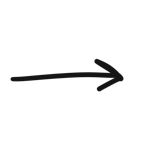
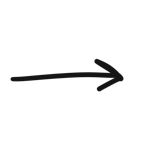
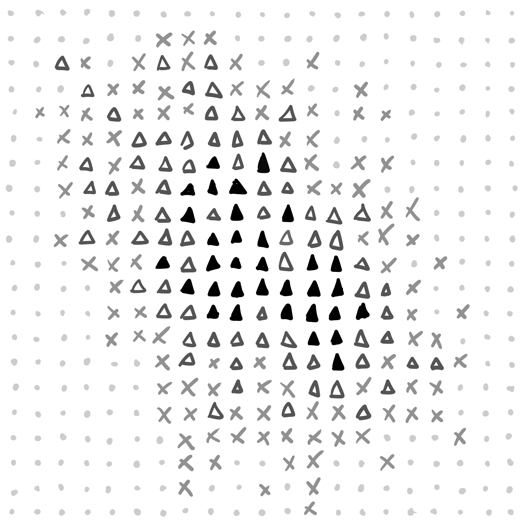
**Specific algorithm ideas**

I think I’ll probably write a noise-map based terrain generation algorithm. I have an idea for an added feature that would make terrain more interesting:

1. Terrain is generated with a noise map, essentially ‘as normal’. This results in a 2D array of float values ranging from 0 to 1, in typical noise function fashion.
2. This data, which can be scaled in and out of anyway within the noise function (which would lead to the noise function generating interpolating values), is then be spaced out – e.g. points that are normally next to each other can have multiple blank spaces inserted between; this would be regular and just controlled by a ‘spacing value’, stored in a 2D array of dimensions double the size of the previous.
3. Values are then converted into discrete tiles, also ‘as normal’, using thresholds – e.g. ‘if the value > 0.8, place this certain tile here, else if value > 0.6, place a different tile’.
4. In the ‘blank’ space, a wave function collapse algorithm could be run, creating variation on a smaller scale while still retaining the larger overarching forms generated by the noise map function.   
   The WFC would work here by randomly choosing coordinates where there is a blank space, and then assessing the 3x3 grid that that coordinate is at the centre of. Based on the surrounding tiles and set rules for how tiles interpolate, a new tile is chosen for that position. For example, only sand tiles can be next to water tiles, or only grass tiles can be next to tree tiles. These rules lead to the generation of realistic gradients that are similar to real life terrain. See this video by Martin Donald - <https://youtu.be/2SuvO4Gi7uY> - to learn more.

The wave function collapse interpolation mixed with more basic noise generation allows for the original overall shape of the noise generated terrain to be retained, while being more detailed with more variety (due to the wave function collapse algorithm).

I’ve drawn a visual mockup to help explain:



I like the way that depth-first dungeon generation algorithms work, I think I’d also like to use a depth-first method for that, though I’m not sure about the specific type of structure I want to generate.

**Data Volumes**

The system should be able to store at least 50 user records initially, with at least 10 worlds per user and 5 layers per world. Each layer should be able to have at least 6 tiles associated with it. 5 layers per world and 6 tiles per layer allows for a wide scope of variety and complexity.

I’ve designed an Entity Relationship Diagram to represent this, but I know that this definitely won’t be my final design – I’ll need design a database with separate tables for separate layer types (and therefore separate tile data types), and also write a system that has more concise key declarations.

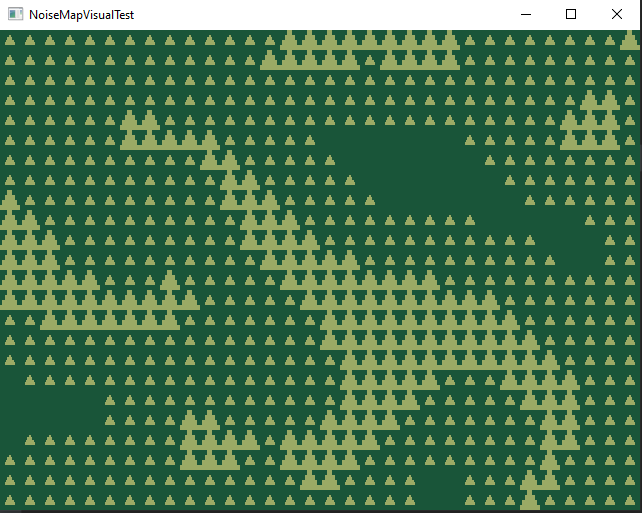
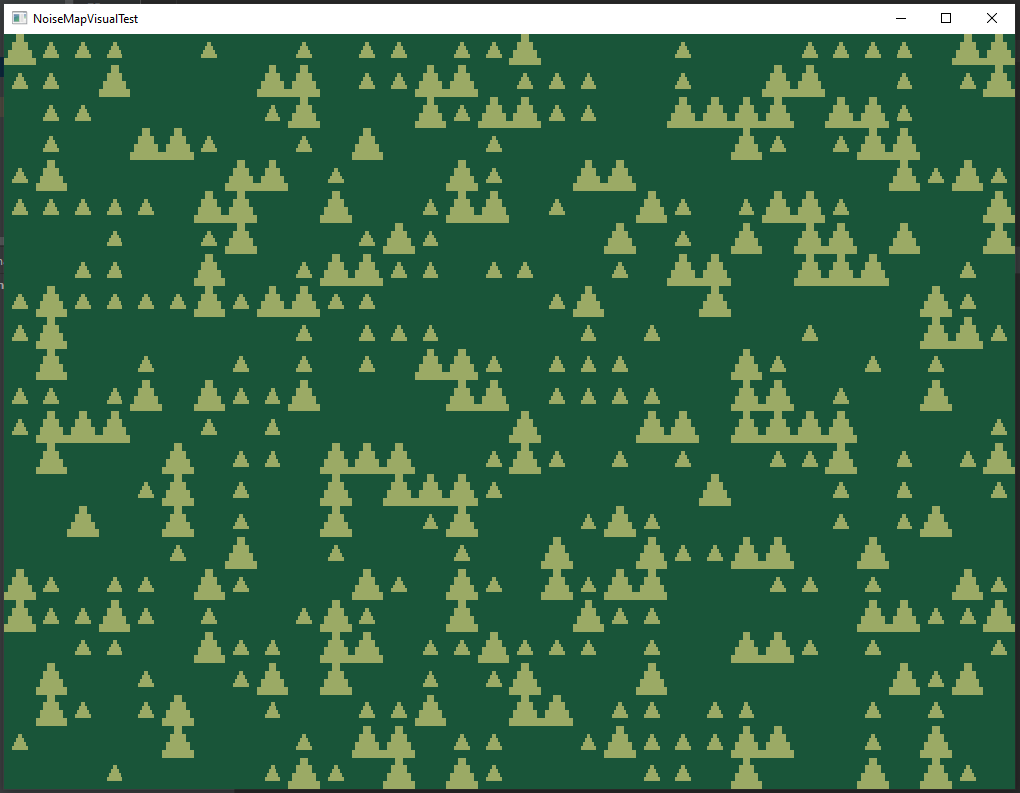
Diagram

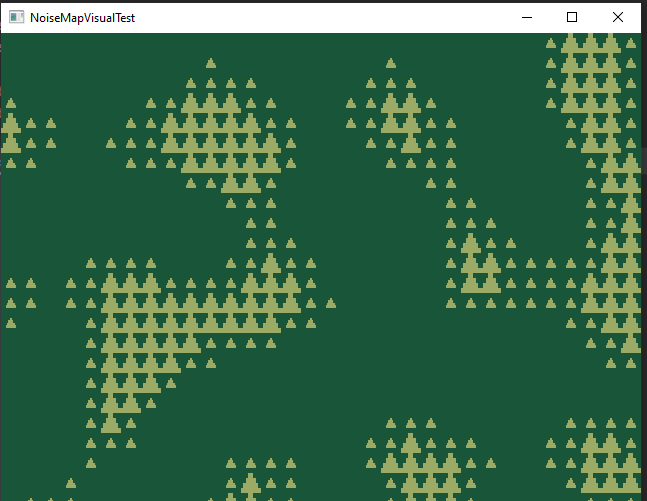
Description automatically generated

**Initial Interface Ideas**

Using some screenshots of a prototype I built to explore procedural generation, I’ve created some UI concepts for the main app screen.

The interface should be unobtrusive and intuitive. To achieve this, I’ve kept the UI from obscuring the generated world and separated out the controls into different panels grouped by function.





Generation properties

Layers

Tile details

**Proposed Complex Algorithms**

* For terrain generation, I’ll probably use a noise map generation algorithm from a noise value, and **pattern matching** for drawing certain tiles at certain thresholds.
* For dungeon generation, algorithms like a depth-first search maze generation algorithm exist, which uses a **stack**. This could end up also being **recursive**.
* **Hashing** password
* **Regular expressions** for things like checking that username and password meet requirements
* **Cross-table parameterized SQL** for getting all save data, user accounts etc.
* **Aggregate SQL functions** to find metadata about the users’ saves and history within the app [for quality of life and app usability].
* I could do **graph traversal** using a **stack** for the navigation of different screens? I’m not sure that my app will have a complex enough page structure to need that though…
* **OOP** for modelling the layers, layer types, etc.
* I’ll probably use **JSON** or **XML** for tileset data, they’re known standards that are good for representing things like that – e.g. Minecraft uses JSON for resource packs (which are a very similar concept)
* **Optimization** during the render process

**Other relevant details**

As I have a small amount of experience with it already, I think I’ll use the LibGDX framework as it handles drawing tiles and user input quite well – and I think it also has quite good pre-existing GUI support, which should allow me to focus more on the ‘meat’ of the program.

**Objectives**

1. New users can register an account to be able to save worlds to the server.

* When creating an account, the user must complete a form with a username and a password.
* The username must be unique within the system.
* The password must be at least 8 characters long and contain at least one letter of the alphabet, and at least one special character and at least one number.
* Passwords must be stored in a secure manner – e.g. via a method that uses hashing, or a method that uses encryption.

1. To log in, the user must enter their username and password.
   * + - The username will be used to find the associated secure password, which will be used to verify that the entered password is correct.
       - If the login details are correct, the user is logged in.
       - If the login details are incorrect, the user is notified that some part of the login was incorrect but is not told what specifically was wrong (as a small way of increasing security).­
2. The user must be able to save and load their created worlds to and from a server.
   * + - Metadata about worlds should be displayed to the user to help them select the one they’re thinking of.
       - When selecting worlds, the worlds should be displayed in a non-random order based on a piece of metadata.
       - There must be no discrepancy between the world the user creates, what’s saved, and how it gets loaded back – i.e. saving and loading should work in their entireties, without causing problems for the user.
3. The app must have the capability to display multiple generated layers ‘on top of each other’, in a specific hierarchy.
   * + - They must be able to be ‘moved’ up and down within the hierarchy.
       - Individual layers must be able to be toggled between shown/hidden.
       - Layers must be able to be edited separately.
       - The user must be able to add and delete layers, allowing for a variable number of layers.
4. There must be a way for a user to import their own tileset.
   * + - As this will mean needing to map out the individual tiles within an overall tilemap, a suitable way of modelling this must be defined.
       - This data should integrate smoothly with the app.
       - The import process doesn’t have to be fully within the UI but should be completable by users.
5. Users should have the ability to modify the way that tiles interact with the data generated by the algorithm from within the app.
   * + - This must be on a per-layer basis.
       - This data must be considered part of layer data, as it is integral to the world that the user designs and therefore must be stored as part of a world save.
6. Algorithms must be ‘seedable’ – in part so layer data is stored efficiently in database.
   * + - There must be a wide range of values for seeds, allowing greater customizability.
       - There must be an overarching world-seed as well as layers having the option to have a different seed.
7. Algorithms must be designed in a way that allows for multiple noticeably different outcomes from them.
   * + - To achieve this, parameters that control the output should have ranges of values that produce a wide range of outputs.
       - Algorithms must be able to produce at least three highly distinct outcomes – e.g. three different, characterful outcomes with their own ‘personalities’ rather than three quite plain shapes that might as well just be the same one.
8. Users should be able to use what they’ve created outside of just my software.
   * + - Users must be able to export a portion of their world to a pre-existing file format for use elsewhere.
9. UI should be designed in a way enables easier use of the software.
   * + - Tips and guidance for controls etc. should be displayed.
       - Coordinates for the user’s current ‘position’ should be displayed.
       - Icons should either use easy to understand symbols or appropriate similarly easy to understand text.
       - The UI, or most of it, should be able to be hidden - for easier software use and so that higher quality screenshots can be taken.

**Design**

System Overview

**Libraries and APIs, etc.**

This project uses ***LibGDX*** – a cross-platform Java game development framework, built on LWJGL. It’s open source, which I like, and powerful while not being overcomplicated; and it has a great wiki. It’s been really useful for this project.

For example:

* SpriteBatch – for drawing tiles
* ShapeRenderer – for drawing shape based UI elements, e.g. outlines
* Vector2D – a data type for 2D coordinates
* User input features, like InputListeners – for keyboard and mouse input
* Scene2D.ui – provides UI widgets
* The Screens system – for different screens, e.g. login screen, main app screen, etc.

For my UI, I modified the openly available ***VisUI skin*** for Scene2D, which can be found here: <https://github.com/czyzby/gdx-skins/tree/master/vis>

The use of LibGDX has also required me to learn how to use ***Gradle***. At first, this was a bit of a strange learning process, partially due to having to also learn Groovy, but I’ve used Gradle where needed – to import Gson and PostgreSQL into the project and manage the LibGDX libraries.

This project also uses Google’s ***Gson*** library – a Java library for converting between Java and JSON objects. This was useful for the Tileset import process.

This project also uses ***PostgreSQL***. I chose PostgreSQL over other implementations and variations of SQL due to its extensive documentation, the fact that it is open source, has great data integrity focused features, has JSON support (which I ended up not using within the database but likely would if I extended this project), is the industry standard (for these reasons and many others), and is supported by Java via JDBC.

This project uses ***JDBC*** to connect to and interact with the PostgreSQL server from my Java code.

For noise generation, I used K.jpg’s ***OpenSimplex2S*** algorithm. I imported this into my project just by copying the java file available on GitHub into my project, in a package outside of the ‘com.joelallison’ under the name ‘tools.OpenSimplex2S’. While Perlin noise is the typical algorithm type used for noise-based terrain generation, Simplex noise seemed like a better option as it has less artifacts.

Link: <https://github.com/KdotJPG/OpenSimplex2/blob/master/java/OpenSimplex2S.java>,

I also used the ***Java.util*** library quite a lot, which is to be expected! This includes use of Java.util.ArrayList, Java.util.HashMap, Java.util.Comparator and many others within the util namespace.

**Classes**

Diagram

Description automatically generatedIn the creation of this project, I’ve identified the need for certain classes to model procedurally generated worlds – some of these classes being subclasses of others, some being inner classes of others. I’ve created a UML class diagram to show these relationships. Due to Java being object oriented, there are many other connections between the classes in my technical solution not shown in this diagram – public variables and methods, defined in one class and used in others. This diagram depicts the non-static classes; classes which are designed to have multiple instances of their objects.

I’ve kept just the defining features of the classes and the methods of the classes that interact with each other, focusing on what I feel the diagram should be highlighting.

**Graphical user interface

Description automatically generated with medium confidenceClass layout**

**Class definitions**

This is an outline of every ‘concrete’ class (non-abstract, has objects) and its key features. Most have more methods and attributes than what are written about here, but I’ve identified these to focus on.

*LibGDX has an object type ‘Screen’. Most of the abstract classes (not written about in this section) extend Screen, are UI related or are to do with File Handling & SQL.*

**Name:** generation.Layer

**Description / Use:**

An abstract object that all other layer types extend from. The subclasses of Layer inherit all of its attributes, and polymorphism is used so that methods are type specific.

The existence of Layer allows there to be a ‘layers’ ArrayList within a World, in which any subclass of Layer can be stored, allowing different layer types to coexist in the same world.

**Key methods:**

* sortTileSpecs() – Is overridden by Layer subclasses. Is used to sort the tileSpecs list – for example, TerrainLayer sorts tileSpecs by the lowerBound value.
* defaultTileValues() – Is overridden by Layer subclasses. Sets up the tileSpecs list with default values.
* There are two constructors for the subclasses to inherit, one with name and seed and one with just seed. The former is used by subclasses for loading layers from the database, the latter is for the creation of new layers within the app.
  + public Layer(String name, Long seed)
  + public Layer(Long seed)

**Name:** generation.TerrainLayer

**Description / Use:**

Extends from Layer. It has the attributes needed for its noise map based terrain generation algorithm, including a 2D float array (float[][]) to store the generated values in.

**Key methods:**

* sortTileSpecs() – Overrides the Layer method. Sorts it the TerrainTileSpecs by lowerBound, using a Comparator. [Java.util.Comparator]
* defaultTileValues() – Overrides the Layer method. Sets up the tileSpecs list with default values. For TerrainLayer, this is using the ‘Trees & Rocks’ tileset which would be packaged with the app if the app was distributed.
* genTerrain() – Terrain generation algorithm. Takes all the terrain generation parameters as input, along with the dimensions of the view being displayed. The terrain is generated and then processed, normalising the values and also affecting them further with the wrapValue method. It returns a 2D float array with values to be used between 0 and 1 - wrapping sometimes puts the values above 1, but those just aren't used; as if they've been clipped off.  
  The algorithm is based on the algorithm by Sebastian Lague in his ‘Procedural Landmass Generation’ series, just up to episode three - <https://youtu.be/MRNFcywkUSA> - but his algorithm was written in C# (for Unity), so my changes include: porting it to Java, making it more powerful with the wrapValue method and the invert boolean, and implementing the tile threshold levels in the drawing process.
* genValueMap() – loads the generated terrain into the layer’s valueMap variable.

Diagram, engineering drawing

Description automatically generated

Diagram, engineering drawing

Description automatically generated

**Name:** generation.MazeLayer

**Description / Use:**

Extends from Layer. It has the attributes needed for its depth-first recursive maze generation algorithm, including a 2D int array (int[][]) to store the generated values in.

**Key methods:**

* sortTileSpecs() – Overrides the Layer method. Actually does nothing in the case of MazeLayer; it’s not vital to user experience that the tileSpecs are sorted.
* defaultTileValues() – Overrides the Layer method. Sets up the tileSpecs list with default values. For MazeLayer, this is using the ‘Walls’ tileset which would be packaged with the app if the app was distributed.
* genMaze() – Depth-first recursive maze generation algorithm uses maze width, height and a seed to generate a maze. Width and height are processed before genMaze() to be the nearest odd number to their input 🡪 number = number + (number+1)%2. The base workings of this algorithm are from here: <http://www.migapro.com/depth-first-search/>*,* but I’ve modified it to be seedable.

count = 0

genMaze():

maze = array[height][width] where every element is set to wall

rand = new Random(seed)

row = rand.oddInt(height)

col = rand.oddInt(width)

maze[row][col] = path

recursion(row, col)

recursion (row, col):

randomDirections = generateRandomDirections()

count++

for direction in randomDirections:

if moving in that direction doesn’t go out of the bounds of the maze:

if the cell two cells in that direction is a wall:

set it, and the cell one cell in that direction, to be path

row, col are set to be that cell’s position

recursion (row, col)

generateRandomDirections():

return a list of [up, down, left, right] put in a random order where the randomisation is seeded by (seed + count)

This pseudocode is [suitably] abstracted and simplified, but accurately describes how the algorithm works. It’s not the most efficient version of a depth-first recursive maze generation algorithm; it essentially [eventually] works its way round to every cell that can be path, not wall, based on pre-existing locations of path. This means that there is a way to get from every path cell to every other path cell (and so entrance and exit could actually be placed anywhere).

**Name:** generation.World

**Description / Use:**

An object that holds all the data about the created world in it – its name, the date it was created, the date it was accessed, its layers, and the overarching world seed – and has methods relating to the world.

It is the object that data from the database is loaded into – and then used in the main app screen – and it is the object that has the data loaded out of it and back into the database when saving.

**Key methods:**

* makeLayerNamesUnique() – [For the sake of the user] makes every layer name unique, by adding an incrementing number to the end of duplicate layers. This is to help the user identify which layer is which, which layer they’ve selected.

makeLayerNamesUnique():

layerCounts = HashMap<String, Integer> with size equivalent to number of layers

for layer in layersList:

rawLayerName = layer.name.strip(regex for trailing numbers)

if (!layerCounts.containsKey(rawLayerName)):

*#if layerCounts doesn’t contain the layer name,*

*#add it to layerCounts with a count of 1*

layerCounts.put(rawLayerName, 1)

else:

*#stores layer name without any numbers at the end, and appropriate count*

layerCounts.put(rawLayerName, layerCounts.get(rawLayerName)+1

*#if a layer with the same name (and no number) also exists,*

*#then this layer’s name is displayed with the appropriate number on the end*

if (layer.name == rawLayerName):

layer.setName(rawLayerName + " (" + layerCounts.get(rawLayerName) + ")")

*Pseudocode*

Layers specifically aren’t modified if they already have a number, and specifically aren’t updated if there are no others with that name.

* swapLayers(int layerOne, int layerTwo) – Swaps two layers around in the list of layers. This is used for moving a layer up and down (e.g. when one moves up, the one above it has to move down.)

tempLayer = layers.get(layerOne)

layers.set(layerOne, layers.get(layerTwo))

layers.set(layerTwo, tempLayer)

* getClearColor() – Returns the baseColor of the lowest layer’s tileset.

**Name:** graphics.Tileset

**Description / Use:**

An object that represents the full mapping of a tileset that’s been loaded into the app (see io.FileHandling and io.JsonHandling), and methods for handling the tilesets.

This class also contains inner classes that are used in the object modelling of the tileset. (see pPAGE\_NUM for more details on this.)

**Key methods:**

* getTileTextureFromName(String name) – returns the TextureRegion (essentially the image for the SpriteBatch to draw, with necessary metadata) associated with a specific tilename, as defined within the Tileset.
* initTileset() – loads the spriteSheet as an actual texture, based on the inputted spriteSheetLocation string.

**Name:** graphics.Tileset.TileCorner [is an inner class of Tileset]

**Description / Use:**

Stores the bottom left corner of a tile. Combined with the tileSize int (an attribute of Tileset), the tile’s specific texture can be parsed from the Tileset spritesheet.

**Attributes:**

* cornerX: int
* cornerY: int

**Name:** graphics.Tileset.TileSpec [is an inner class of Tileset]

**Description / Use:**

An abstract class, to be the superclass of the classes that represent how the user chooses to map tiles to a layer’s generative algorithm.

**Attributes:**

* Name: String [inherited by subclasses]

**Name:** graphics.Tileset.TerrainTileSpec [is an inner class of Tileset]

**Description / Use:**

TerrainTileSpec is a subclass of TileSpec, inheriting the name attribute.

Represents how the user chooses to map tiles to TerrainLayer’s generative algorithm.

Because TerrainLayer generates a map of values from 0 to 1, the defining attribute of TerrainTileSpec is the threshold below which the tile isn’t drawn – called ‘lowerBound’. e.g. if a tile has a threshold of 0.6, a value of 0.55 would mean there’s no tile drawn there, but a value of 0.65 would result in a drawn tile.

**Attributes:**

* lowerBound: int

**Name:** graphics.Tileset.MazeTileSpec [is an inner class of Tileset]

**Description / Use:**

MazeTileSpec is a subclass of TileSpec, inheriting the name attribute.

Represents how the user chooses to map tiles to MazeLayer’s generative algorithm.

Because TerrainLayer generates a map of path or cell tiles, the important part of the MazeLayer tile mapping is choosing the right tiles based on the surrounding walls – e.g. a tile with adjacent walls left and right, but no others would be different to a tile with walls above, left, and right.

To represent this, I’ve chosen to use a 2D boolean array – 3x3.

**Attributes:**

* neighbourMap: boolean[][]

**Methods:**

* neighbourMapParseString() – converts inputted 9 character string of 1s and 0s in to a 3x3 boolean 2D array.

Actual code:

public static boolean[][] neighbourMapParseString(String neighbourMapString) {

boolean[][] oMap = new boolean[3][3];

for (int i = 0; i < neighbourMapString.length(); i++) {

//converts char to int to boolean

oMap[i / 3][i % 3] = !(Character.getNumericValue(neighbourMapString.charAt(i)) == 0);

}

return oMap;

}

* neighbourMapToString() – converts the stored neighbourMap into a 9 character string of 1s and 0s.  
    
  Actual code:

public String neighbourMapToString() {

StringBuilder sb = new StringBuilder();

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

if (neighbourMap[i][j]) {

sb.append("1");

} else {

sb.append("0");

}

}

}

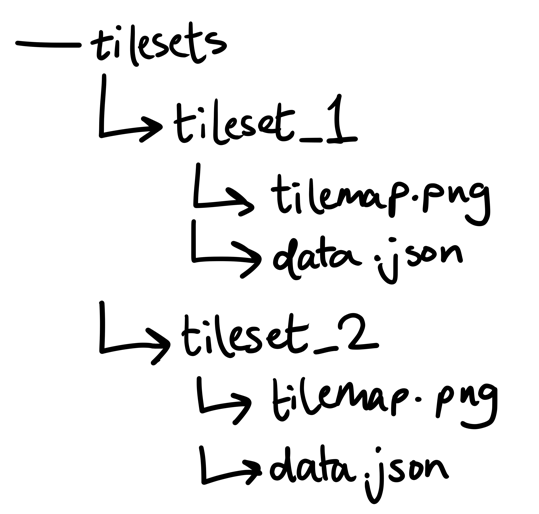
return sb.toString();

}

**JSON - Tilesets**

For the modelling of tileset data, I felt that JSON was the best thing to use. It can be used to directly model the structure of objects and is easily legible.

LibGDX allows ‘TextureRegion’s to be created from a Texture (a LibGDX object which is essentially just an imported image) by declaring the bottom left of the region with coordinates and a width and height of the TextureRegion. I chose TextureRegions because I feel that a single tilemap image is better than many individual tile images, and (aside from their construction) they function analogously to Textures.



File structure of the tilesets folder in abstract terms.

Within the app, because I want all tilesets in the tilesets folder to be loaded and accessible to the user, I think it is best that they’re listed by their names, so I chose to store the tilesets in a HashMap – with their name (string) as the key, and a Tileset object as the value.

As the names need to be written in the JSON, yet shouldn’t be a part of the tileset object (to avoid redundancy, as they’re already stored as the HashMap entry key), I separated the name from the rest of the data in the loading process and then removed it from the string that the JSON was being held in.

After that, I used Gson’s fromJson() method to parse a Tileset object from the JSON.

**Structure of Tileset object and the consequent JSON**For the function of the app, I need to represent the following in the Tileset object:

|  |  |  |
| --- | --- | --- |
| **Property** | **Java data type** | **Details** |
| Spritesheet | String for filename, Texture for actual use | String is to direct the app to the location of the spritesheet image |
| Creator of tileset | String | To encourage community, sharing of tilesets etc. |
| Tile size | Int | The size, in pixels, of the tiles in the tileset – tiles are square, so a single ‘size’ can be used for both width and height. While I’d like to be able to implement variable tile sizes [to allow for more complex and interesting generation] at some point, I will have to do this outside the boundaries of my coursework. |
| Base colour | String (hex code), Color | The base colour of the tileset, so that areas where tiles aren’t drawn can be filled in. |
| Tile map | HashMap<String, TileCorner> | Labelled locations of every tile within the tileset spritesheet – the TileCorner object is just an x & y coordinate. |

See the Tileset object as well as FileHandling’s importTilsets method in my Technical Solution for further details on the implementation.

*Abstract version of the JSON*

{  
 "name": “name\_value”,  
 "creator": "creator\_value",  
 "spriteSheet": "spritesheet.png",  
 "tileSize": 8,  
 "baseColorHex": "#000000",  
 "map": {  
 "bottom\_left": {  
 "cornerX": 0,  
 "cornerY": 0  
 },  
 "bottom\_right": {  
 "cornerX": 1,  
 "cornerY": 0  
 },  
 "top\_left": {  
 "cornerX": 0,  
 "cornerY": 1  
 },  
 "top\_right": {  
 "cornerX": 1,  
 "cornerY": 1

}  
 }  
}

This particular made-up tileset is arbitrarily 2x2, and each tile’s pixel location is found at  
(cornerX \* tileSize, cornerY \* tileSize). So, for example, ‘bottom\_right’ would be at x=8, y=0.

Because the declaration of the map section can get quite repetitive, I’ve written a short python script to help with the import process. I chose to write a python script as I wanted the tool for mapping to be external as the way that the json etc. is imported is external too, and python felt like the right choice for this sort of task. If this project was ‘officially’ released, I would include the script under an article on the project’s wiki titled ‘Importing Tilesets’.

Given that it’s not part of the java project that is the main app, I’ve decided to print the python script here rather than as part of the technical solution (even though it is code written by me, that’s part of the project).

def getTiles(*height*, *width*):

values = [[[0 for d in range(3)] for w in range(int(*width*))] for h in range(int(*height*))]

previous\_tile\_names = [“” for i in range(int(width)\*int(height))]

for y in range(int(height)):

for x in range(int(width)):

values[y][x][0] = input("Tile name? [type 'ignore' to not add it]")

if (values[y][x][0] != "ignore"):

while (values[y][x][0] in previous\_tile\_names):

values[y][x][0] = input("Tile names must be unique within a tileset.\nTile name?")

values[y][x][1] = x

values[y][x][2] = y

previous\_tile\_names[tiles\_done] = values[y][x][0]

tiles\_done += 1

return values

def formatMapLine(*line*):

return r'\"' + str(*line*[0]) + r'\":{\"cornerX\":' + str(*line*[1]) + r',\"cornerY\":' + str(*line*[2]) + r'}'

def makeMap(*values*):

jsonString = r'\"map\":{'

for y in range(0, len(*values*)):

for x in range(0, len(*values*[y])):

jsonString = str(jsonString + formatMapLine(*values*[y][x]) + r',')

return jsonString.strip(',') + r'}'

width = input("What's the width of the tilemap? (tiles)")

height = input("What's the height of the tilemap? (tiles)")

output = (makeMap(getTiles(height, width)))

print("\n" + output + "\n")

print(output.replace("\\", ""))

The user inputs the width and height (in tiles) of the spritesheet. Then, going through the spritesheet left to right, bottom to top, asking for a name for each tile. It outputs the json for the map section and prints two versions of the output – with and without the Java escape characters that enable quote marks to ‘work’.

I used this script in the creation of the tilesets that I’ve added to the project.

**Exporting sections of worlds as files**

I thought about using a number of different file types, but in the end settled on .csv as it is quite well known, legible, and simple to reinterpret by users (and grid based!).

**Updated DFD**

Through designing this project, I realised that changes needed to be made to the structure of the database from the original plan. I’ve updated the Data Flow Diagram to reflect this.

As well as database changes, it’s also worth mentioning that I’ve removed the settings change process as I haven’t implemented it.

I’ve chosen for the DFD to just show the flow of data between the app and the database. A diagram that also includes flow within the app would be extremely overloaded with information and therefore not fulfilling the original purpose of a diagram; to convey information in an easy-to-understand manner.

Diagram

Description automatically generated

**Database design**

**Description of record structure**

I’m using PostgreSQL 14, and for this project I’m just hosting the server locally with the official PostgreSQL application.

**On modelling the superclass-subclass relationship of layers**

There are two classes – ‘TerrainLayer’ and ‘MazeLayer’ – which are subclasses of ‘Layer’ (and in theory, an infinite amount of layer subclasses should be able to be created).

I’ve designed the database in a way to eliminate data redundancy. There’s a ‘layer’ table which includes all the variables that TerrainLayer and MazeLayer share (inherited from Layer), and then individual ‘terrain\_layer’ and ‘maze\_layer’ tables for the data specific to them.

To create the necessary link, I’ve given these tables all a ‘layer\_type’ and ‘layer\_id’ field. Layer type is a character, layer ID is an integer and together they make a composite sort of ID – e.g. for the third terrain layer (in the database), it would be (‘T’ | 3 ).

Layer type and layer ID have a unique constraint set on them [which I’ve named ‘Composite LayerID must be unique’], meaning that (‘T’ | 3 ) is considered different to (‘M’ | 3 ) and different to (‘T’ | 4 ) – but there’s only one (‘T’ | 3 ).

Layer type is used in the world loading process, to know which subclass table to get the rest of the data from and what object type to load into the *layers* ArrayList. In the subclass tables, their ‘layer\_type’ value must be the same specific character for every entry – I’ve used ‘T’ for terrain\_layer and ‘M’ for maze\_layer – and I’ve used a check constraint to enforce this.

**Entity Relationship Diagram**

users(username, password, password\_salt, account\_created, last\_logged, settings)

world(world\_name*, username*, created\_timestamp, last\_accessed, world\_seed)

layer(layer\_number, *world\_name, username*, layer\_name, show\_layer, seed, layer\_type, layer\_id, inherit\_seed, center\_x, center\_y, tileset\_name)

terrain\_layer(*layer\_id*, *layer\_type*, scale, octaves, lacunarity, wrap, invert)

maze\_layer(*layer\_id*, *layer\_type*, width, height)

terrain\_tile\_specs(*layer\_id*, tile\_name, lower\_bound)

maze\_tile\_specs(*layer\_id*, tile\_name, orientation\_id)

I’ve kept to the PostgreSQL naming conventions in my actual database (and they can also be seen in the above lines) but didn’t fully stick to them for the ERD, just for aesthetic reasons.

In the ERD (see next page), I’ve included the detail of whether a field can be left empty or not (NOT NULL) as I feel it relates to how the entities link together.

**Diagram

Description automatically generated**

**neighbour\_map**

**Data Dictionary**

***users***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Item Name** | **Example** | **Data Type** | **Size / Dimensions** | **Validation?** |
| username | “joelallison” | varchar(20) | 20 | Yes, presence and length check to make sure name is a good length and present.  Error message: “Username must have a length <= 20.” |
| password | "92aba0af8ac62f7aa3bbbce9eb7df74784ea6f763963c3ace885f91339f72e110af8cb92197c55945325ec87c8aafd128ce1df0f53c1d20013bceaf64c0faf03” (all one line) | text | (Due to SHA-512) 128 [hex digits] | Yes, a format check to help password security.  Error message: “Password must have a minimum of eight characters and maximum of 32, at least one letter, one number, one of these: \*.!@$%\_?/~+-=,  one uppercase character, one lowercase character.” |
| password\_salt | a29b40de1b8a776aa4f14cde0931e34f | bytea | 32 hex digits | None needed. |
| account\_created | "2023-03-01 18:58:48.807936+00" | timestamp with time zone | Precision of 1 microsecond | None needed. |
| last\_logged | "2023-03-01 18:58:48.807936+00" | timestamp with time zone | Precision of 1 microsecond | None needed. |
| settings | {“master\_volume”: 100} | json | 255MB of JSON data | None needed. |

***world***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Item Name** | **Example** | **Data Type** | **Size / Dimensions** | **Validation?** |
| world\_name | “A world” | text | Limit of PostgreSQL text is 65,535 bytes. | Yes, presence check.  Error message: “Name must not be left blank!” |
| username | “joelallison” | varchar(20) | 20 | Yes, presence and length check to make sure name is a good length and present.  Error message: “Username must have a length <= 20.” |
| created\_timestamp | "2023-03-01 18:58:48.807936+00" | timestamp with time zone | Precision of 1 microsecond | None needed. |
| last\_logged | "2023-03-01 18:58:48.807936+00" | timestamp with time zone | Precision of 1 microsecond | None needed. |
| world\_seed | 10203040 | bigint | Maximum of: 9,223,372,036,854,775,807 (same as Java Long) | Format check - must be positive integer given that all the algorithms use a Long as their seed.  Error message: Must be a positive int.” |

***layer***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Item Name** | **Example** | **Data Type** | **Size / Dimensions** | **Validation?** |
| layer\_number | 2 | integer | PostgreSQL integer max is 2147483647. | None needed. |
| world\_name | “A world” | text | Limit of PostgreSQL text is 65,535 bytes. | Yes, presence check.  Error message: “Name must not be left blank!” |
| username | “joelallison” | varchar(20) | 20 | Yes, presence and length check to make sure name is a good length and present.  Error message: “Username must have a length <= 20.” |
| layer\_name | “Wiggly Mountains” | text | Limit of PostgreSQL text is 65,535 bytes. | An internal format check:  Layers with the same name are automatically updated to “layer\_name (1)” and “layer\_name (2)” [etc.]  This is to help users differentiate between layers at a glance. |
| show\_layer | true | boolean | - | None needed. |
| seed | 40302010 | bigint | Maximum of: 9,223,372,036,854,775,807 (same as Java Long) | Format check - must be positive integer given that all the algorithms use a Long as their seed.  No error message, just a regex filter on the input field. |
| layer\_type | ‘T’ | “char” (ASCII character) | Limited to (single) ASCII characters only | None needed. |
| layer\_id | 7 | integer | PostgreSQL integer max is 2147483647. | None needed. |
| inherit\_seed | false | boolean | - | None needed. |
| center\_x | 0 | integer | PostgreSQL integer max is 2147483647. | None needed. |
| center\_y | 0 | integer | PostgreSQL integer max is 2147483647. | None needed. |
| tileset\_name | “Trees & Rocks” | text | Limit of PostgreSQL text is 65,535 bytes. | None. I felt that none was needed, but I |

***terrain\_layer***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Item Name** | **Example** | **Data Type** | **Size / Dimensions** | **Validation?** |
| layer\_type | ‘T’ | “char” (ASCII character) | Limited to (single) ASCII characters only | A check constraint is done to make sure layer\_type is ‘T’ (T for Terrain) |
| layer\_id | 7 | integer | PostgreSQL integer max is 2147483647. | None needed. |
| scale | 0.5 | numeric | Limited to be between 0.005 and 256 within app. | Range – to guide the user towards values that have much more usable outcomes and fight the ‘garbage in / garbage out’ possibility. |
| octaves | 2 | smallint | Limited to be between 1 and 3 within app. | Range – to guide the user towards values that have much more usable outcomes and fight the ‘garbage in / garbage out’ possibility. |
| lacunarity | 2 | numeric | Limited to be between 0.01 and 10 within app. | Range – to guide the user towards values that have much more usable outcomes and fight the ‘garbage in / garbage out’ possibility. |
| wrap | 4 | smallint | Limited to be between 1 and 20 within app. | Range – to guide the user towards values that have much more usable outcomes and fight the ‘garbage in / garbage out’ possibility. |
| invert | true | boolean | - | None needed. |

***maze\_layer***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Item Name** | **Example** | **Data Type** | **Size / Dimensions** | **Validation?** |
| layer\_type | ‘M’ | “char” (ASCII character) | Limited to (single) ASCII characters only | A check constraint is done to make sure layer\_type is ‘M’ (M for Maze) |
| layer\_id | 2 | integer | PostgreSQL integer max is 2147483647. | None needed. |
| width | 32 | smallint | Min: 3  Max: 857  Limited within app, limits are due to the way the algorithm works. | Range – to disallow values that would cause an error. |
| height | 32 | smallint | Min: 3  Max: 857  Limited within app, limits are due to the way the algorithm works. | Range – to disallow values that would cause an error. |
| opaque | true | boolean | - | None needed. |

***terrain\_tile\_specs***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Item Name** | **Example** | **Data Type** | **Size / Dimensions** | **Validation?** |
| layer\_id | 7 | integer | PostgreSQL integer max is 2147483647. | None needed. |
| tile\_name | “tree\_1” | text | Limit of PostgreSQL text is 65,535 bytes. | None needed. |
| lower\_bound | 0.65 | numeric | Min: 0  Max: 1 | Range – terrain generation values are between 0 and 1. |

***maze\_tile\_specs***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Item Name** | **Example** | **Data Type** | **Size / Dimensions** | **Validation?** |
| layer\_id | 2 | integer | PostgreSQL integer max is 2147483647. | None needed. |
| tile\_name | “bottom\_left\_corner” | text | Limit of PostgreSQL text is 65,535 bytes. | None needed. |
| neighbour\_map | 000010000 | text | - | None needed. |

**Entity Design in PostgreSQL**

I considered not including it here, but this SQL is a part of my project. I realise that I could’ve included this information as part of the data dictionary, but I feel that that would reduce the legibility of the data dictionary.

users:

CREATE TABLE IF NOT EXISTS public.users

(

username character varying(20) COLLATE pg\_catalog."default" NOT NULL,

password text COLLATE pg\_catalog."default" NOT NULL,

account\_creation timestamp with time zone NOT NULL DEFAULT CURRENT\_TIMESTAMP,

last\_logged timestamp with time zone NOT NULL DEFAULT CURRENT\_TIMESTAMP,

settings json,

password\_salt bytea NOT NULL,

CONSTRAINT user\_pkey PRIMARY KEY (username)

)

world:

CREATE TABLE IF NOT EXISTS public.world

(

username character varying(20) COLLATE pg\_catalog."default" NOT NULL,

world\_name text COLLATE pg\_catalog."default" NOT NULL,

created\_timestamp timestamp with time zone NOT NULL DEFAULT CURRENT\_TIMESTAMP,

last\_accessed\_timestamp timestamp with time zone NOT NULL DEFAULT CURRENT\_TIMESTAMP,

world\_seed bigint NOT NULL DEFAULT 0,

CONSTRAINT creation\_pkey PRIMARY KEY (username, world\_name),

CONSTRAINT creation\_name\_unq UNIQUE (world\_name, username),

CONSTRAINT creator\_name\_username\_fk FOREIGN KEY (username)

REFERENCES public.users (username) MATCH SIMPLE

ON UPDATE CASCADE

ON DELETE CASCADE

NOT VALID

)

layer:

CREATE TABLE IF NOT EXISTS public.layer

(

username character varying(20) COLLATE pg\_catalog."default" NOT NULL,

world\_name text COLLATE pg\_catalog."default" NOT NULL,

layer\_number integer NOT NULL,

layer\_name text COLLATE pg\_catalog."default" NOT NULL,

show\_layer boolean NOT NULL,

seed bigint NOT NULL,

layer\_type "char" NOT NULL,

layer\_id integer NOT NULL,

inherit\_seed boolean NOT NULL DEFAULT false,

center\_x integer NOT NULL DEFAULT 0,

center\_y integer NOT NULL DEFAULT 0,

tileset\_name text COLLATE pg\_catalog."default" NOT NULL,

CONSTRAINT layer\_pkey PRIMARY KEY (layer\_number, world\_name, username),

CONSTRAINT "Composite LayerID must be unique" UNIQUE (layer\_type, layer\_id),

CONSTRAINT world\_fk FOREIGN KEY (world\_name, username)

REFERENCES public.world (world\_name, username) MATCH SIMPLE

ON UPDATE CASCADE

ON DELETE CASCADE

NOT VALID,

CONSTRAINT "Layer type must be valid type" CHECK (layer\_type = 'T'::"char" OR layer\_type = 'M'::"char") NOT VALID

)

terrain\_layer:

CREATE TABLE IF NOT EXISTS public.terrain\_layer

(

layer\_type "char" NOT NULL DEFAULT 'T'::"char",

layer\_id integer NOT NULL,

scale numeric NOT NULL,

octaves smallint NOT NULL,

lacunarity numeric NOT NULL,

wrap smallint NOT NULL,

invert boolean NOT NULL,

CONSTRAINT terrain\_layer\_pkey PRIMARY KEY (layer\_id),

CONSTRAINT layer\_superclass\_fk FOREIGN KEY (layer\_id, layer\_type)

REFERENCES public.layer (layer\_id, layer\_type) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE CASCADE

NOT VALID,

CONSTRAINT "Layer Type must be of valid type: T" CHECK (layer\_type = 'T'::"char") NOT VALID

)

maze\_layer:

CREATE TABLE IF NOT EXISTS public.maze\_layer

(

layer\_type "char" NOT NULL DEFAULT 'M'::"char",

layer\_id integer NOT NULL,

width integer NOT NULL,

height integer NOT NULL,

opaque boolean NOT NULL,

CONSTRAINT maze\_layer\_pkey PRIMARY KEY (layer\_id),

CONSTRAINT layer\_superclass\_fk FOREIGN KEY (layer\_id, layer\_type)

REFERENCES public.layer (layer\_id, layer\_type) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE CASCADE

NOT VALID,

CONSTRAINT "Layer Type must be of valid type: M" CHECK (layer\_type = 'M'::"char") NOT VALID

)

terrain\_tile\_specs:

CREATE TABLE IF NOT EXISTS public.terrain\_tile\_specs

(

layer\_id integer NOT NULL,

tile\_name text COLLATE pg\_catalog."default" NOT NULL,

lower\_bound numeric NOT NULL,

CONSTRAINT terrain\_tile\_specs\_pkey PRIMARY KEY (layer\_id, tile\_name),

CONSTRAINT layer\_id\_fk FOREIGN KEY (layer\_id)

REFERENCES public.terrain\_layer (layer\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE CASCADE

NOT VALID

)

maze\_tile\_specs:

CREATE TABLE IF NOT EXISTS public.maze\_tile\_specs

(

layer\_id integer NOT NULL,

tile\_name text COLLATE pg\_catalog."default" NOT NULL,

neighbour\_map text COLLATE pg\_catalog."default" NOT NULL,

CONSTRAINT maze\_tile\_specs\_pkey PRIMARY KEY (layer\_id, tile\_name),

CONSTRAINT layer\_id\_fk FOREIGN KEY (layer\_id)

REFERENCES public.maze\_layer (layer\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE CASCADE

NOT VALID

)

**Text, letter

Description automatically generatedData Volumes**

A single user with 5 worlds, resulting in 9 layers total (each with their corresponding layer specific entry), 78 terrain tile spec lines and 32 maze tile spec lines takes up 9281kB. If we assume that a user might have worlds 3x as complex as these, might have 15 worlds, then with 50 users, the database would be:

9281 \* 3 \* 3 \* 50 = 4176450 kB = 3.98GB.

I think that this is a pretty good size, and I would say that it is because of efficient use of space due to seed-based generation (and therefore the generated worlds aren’t stored tile by tile) and also good database modelling, with tables in third normal form and modelled effectively.

**User Interface**

In designing the user interface, I’ve kept user experience strongly in mind, intending to create an easy-to-understand interface with a good flow between sections of the app.

Note: I used LibGDX’s Scene2D.ui for UI elements and may mention Scene2D.ui objects.

Login / Register



Username input field

Text field for the input of a username.

Used for registering and logging in.

Password input field

Text field for the input of a password.

Used for registering and logging in.

Register button

Using the text inputted into the fields above, runs the register process

Login button

Using the text inputted into the fields above, runs the login process

Feedback label

(text set to “” in this screenshot)I added a ‘label’ to the bottom line of the table to be able to send a message back to the user

e.g. ‘Username taken.’ or ‘Username or password incorrect.’

This is the login / register page. The TextFields can be used to register or login, their use depends on which button is pressed. The interactive UI elements are part of a Table to structure them properly, using the .colspan() function to make the fields take up multiple columns.

Graphical user interface, text

Description automatically generatedWorld Select Screen

Worlds

A Table of TextButtons, displayed in a ScrollPane so that the user can have more worlds than fit on the screen.

Back button

When clicked, a pop-up appears warning the user that they will be logged out if they continue.

If ‘Continue’ is clicked, they’re logged out and taken back to the Login / Register page.

Clicking ‘cancel’ closes the pop-up.

New World button

When clicked, a pop-up appears for the creation of a new world.

The pop-up has a field for world name and a field for world seed.

If the input data is valid, clicking ‘Go’ loads a new world with those settings into the main app screen.

Clicking ‘cancel’ closes the pop-up.

Delete button

When clicked, a pop-up appears confirming that the user wants to delete the world to the left of the button.  
  
If ‘Yes’ is clicked, the world is deleted.

Clicking ‘No’ just closes the pop-up.

World (button)

Displays metadata about a world created by the user.

When clicked, that world is loaded into the main app screen.

This is the world select screen. The worlds are ordered by ‘Last accessed’, as this was specified in the SQL query for the world data. Timestamps are formatted using java.time.format.DateTimeFormatter and a format string that I specified.

Main App Screen

**Graphical user interface, text

Description automatically generated**

**3**

**10**

**11**

**9**

**8**

**7**

**6**

**5**

**4**

**2**

**1**

(I felt that numbering the screenshot works better here than annotating it directly)

1. *Back button:*

Takes the user back to the world select screen. When the button is clicked, a pop-up is shown, warning the user that they may lose their progress if they haven’t saved.

If they click ‘Continue’, they’re taken to the world select screen. If they click ‘Cancel’, the pop-up is closed.

1. *World details label:*  
   Displays world name, world seed and the user’s current position within the world. It’s updated as the details change.
2. Generation settings panel:  
     
   The controls for the parameters of generation for the selected layer. The elements within the panel change with layer type (as different controls are needed for different generative algorithms). When the parameters are changed with these controls, the user can immediately see these changes affecting the generated world in the ‘World view box’.
3. Tile specs panel

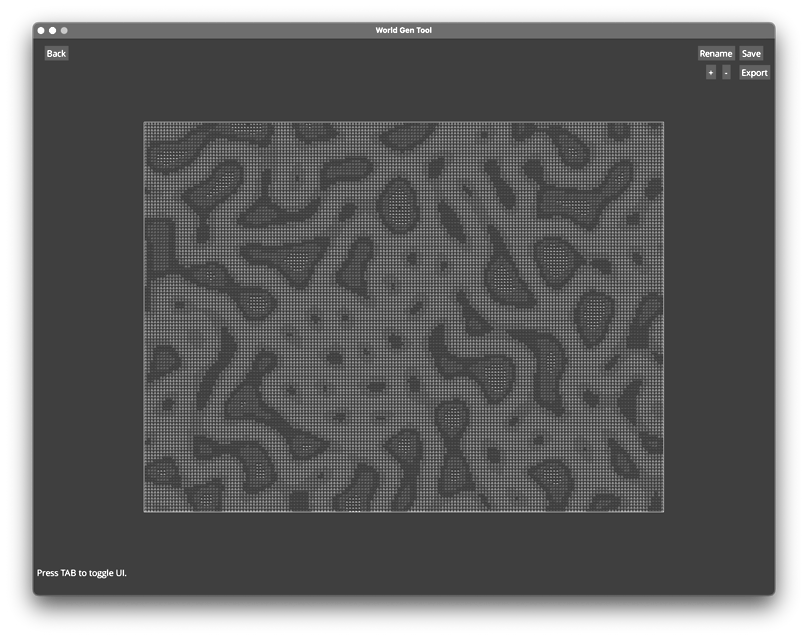
Qr code

Description automatically generated  
Controls for the ‘Tile Specs’ of the selected layer. The elements within the panel change with layer type (as there are different tile spec types for different generative algorithms). ). When the parameters are changed with these controls, the user can immediately see these changes affecting the generated world in the ‘World view box’.  
  
  
Note: this is how Generation settings and Tile specs change for a maze layer (rather than a terrain layer, seen above)

1. World view box  
     
   Displays the generated world based on the generated data (controlled with the generation settings panel), how that data is mapped to tiles (Tile specs panel) and the hierarchy of layers (Layers panel). Basically, this box displays the ‘final product’ the user has created.
2. Layers panel  
     
   Controls for the hierarchy of layers, layer names, whether layers are visible or not, and adding and removing layers. I took inspiration from Photoshop’s layer system in designing this panel’s functionality and layout.
3. Graphical user interface

   Description automatically generatedRename button  
     
   When clicked, a pop-up is displayed for the renaming of the world.

1. Save button  
     
   When clicked, the world is saved and a pop-up is shown confirming this.

1. A screenshot of a game

   Description automatically generated with medium confidenceA screenshot of a video game

   Description automatically generatedRender scale buttons (+ and -)  
     
   Changes the number of tiles in the world view box; scaling up or down the width and height (together).

1. Export button  
     
   When clicked, a pop-up is displayed for exporting a portion of the world as a file.

Graphical user interface

Description automatically generated

All these screenshots (plus some others) can be seen in colour here: <https://imgur.com/a/8ZinBTU>

**Algorithms**

SQL

* Use of SELECT statement to load worlds, check if a username is available, and many other uses.
* Specific use of SELECT COUNT for finding number of layers in a world – used for displaying world metadata but also knowing whether to UPDATE or INSERT for saving layers.
* Specific use of SELECT MAX for calculating layer ID.
* Use of UPDATE statement for saving and renaming worlds.
* Use of INSERT INTO for adding a new user, saving worlds.

ArrayList usage

ArrayLists used for:

* Storing and handling layers in the world – where a dynamic, ordered structure is obviously very important.
* Storing and handling tileSpecs.

HashMap usage

HashMaps used for:

* The makeLayerNamesUnique() method – storing raw layer name (String) as the key, with the number of occurrences of that name (int) as the value.
* Storing loaded tilesets – tileset name (String) as the key, with tileset data (object - Tileset) as the value.
* Storing tile locations within spritesheet – with tile name as the key, and a coordinate (object – TileCorner)

Recursion

Recursion was used for the [recursive] depth-first maze generation algorithm. See the Classes section (MazeLayer) for the pseudocode of this algorithm.

Hashing

Hashing was used so that users’ passwords would be secure. For extra security, I chose to also generate a ‘salt’ (random byte data) for use in the hash.

Regular expressions

Regular expressions were used for:

* Validating password format
* Validating username length
* Validating data types of input – e.g. using “-?[0-9]+” to filter a text field that should only have integers input into it
* Removing unnecessary whitespace in JSON files – using “\s{2,}” to identify whitespace of length 2 or more (which is then replaced with just a single space).

Dynamic generation of objects, ‘Complex’ OOP Coding

OOP used:

* Based on user specification, world objects, layer objects (terrain and maze), tile specification objects, are generated dynamically. The layer objects are designed with inheritance and polymorphism.

And throughout the project.

File handling

File handling:

* Exporting files with custom file names – making a new file where necessary, overwriting pre-existing files.
* Importing Tilesets (JSON), name parsed separately to reset of tile data.

Use of JSON

(See JSON section) Used for the import of tilesets.

Optimisation

Instead of rendering tiles on top of each other unnecessarily, no tile is drawn if there is already a tile on a higher layer that would cover it up. This is done with a 2D boolean array indicating whether a tile is already drawn at that grid position.

Without this system in place, tile drawing becomes increasingly resource-intensive with every layer added. This makes the time complexity go from O(n2) to O(log2n), due to layers filling up the draw space more and more with every one that is added.

Every method is written with optimisation and efficiency in mind, but this is the most blatant, strongest instance of optimisation.

Pattern matching

The user can import their own tiles to the app (through the tilesets feature). For the walls of a maze, different tiles need to be displayed based on the surrounding tiles (e.g. a top-left corner needs to ‘connect’ to the tile to the right and the tile below).  
  
I chose to use a 3x3 2D boolean array (with the user’s input coming from a grid of checkboxes) for the user to define how a tile connects.  
  
The pattern matching occurs when the tiles are drawn in. For each tile being drawn, the 3x3 boolean array is found for that tile based on the maze data (1s for walls, 0s for paths). Then, the first tile with that ‘neighbour map’ in the tileSpecs list is the one that’s drawn – tileSpecs is ordered alphabetically, one tile has to take precedence over another somehow.  
  
This means that the maze generation algorithm can just generate walls and paths, without having to process any further details about the maze – which is the perfect outcome.

The terrain generation threshold system is another instance of pattern matching, albeit a simple one.

**Technical Solution**

**Testing**

**Tests**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Test ID*** | ***Description*** | ***Test Data*** | ***Expected Outcome*** | ***Actual Outcome*** |
| Login / Register Page | | | | |
| 1 | Check that the user  can register if all  fields are completed  correctly | (Valid data)  Tested by logging in with valid details.   * Username field:  Username which is in database – ‘appUser4’. * Password field:  A password that meets requirements – e.g. ‘secureP4ssword!!’ * ‘Register’ button clicked. | Screen changes from Login / Register to World Select, displaying just the ‘New World’ and ‘Back’ buttons, as the user hasn’t created any worlds yet. | Chart  Description automatically generatedScreen changes from Login / Register to World Select, displaying just the ‘New World’ and ‘Back’ buttons, as the user hasn’t created any worlds yet. |
| 2 | Check that user can’t register with an already taken username | (Invalid data)  Tested by trying to register with a username already in database and valid password credentials.   * Username field:  Username which is in database – ‘johnDoe1’. * Password field:  Arbitrary password which meets password requirements – ‘aPassword1!’. * Register button clicked. | Screen stays on Login / Register page (no registration occurs), and text stating ‘Username taken’ is shown. | A green screen with white text  Description automatically generated with low confidenceScreen stays on Login / Register page (no registration occurs), and text stating ‘Username taken’ is shown. |
| 3 | Check that password must have 8 characters, 1 special character, 1 number | (Boundary data)  Tested by trying to register with a username not in database and invalid password credentials.   * Username field:  Username which isn’t in database – ‘freshUser123’. * Password field:  Arbitrary password which doesn’t meet password requirements – ‘wordpass’   Register button clicked. | Screen stays on Login / Register page (no registration occurs), and text informing that password doesn’t meet requirements and stating requirements is shown. | Text  Description automatically generatedScreen stays on Login / Register page (no registration occurs), and text informing that password doesn’t meet requirements and stating requirements is shown. |
| 4 | Check that user can login successfully when correct details are entered | (Valid data)  Tested by logging in with valid details.   * Username field:  Username which is in database – ‘johnDoe1’. * Password field:  The password associated with ‘johnDoe1’ username.   ‘Login’ button clicked. | Screen changes from Login / Register to World Select, displaying the list of worlds made by ‘johnDoe1’. | A screenshot of a computer  Description automatically generatedScreen changes from Login / Register to World Select, displaying the list of worlds made by ‘johnDoe1’. |
| World Select Page | | | | |
| 5 | Check that user can load a stored world | (Valid data)  On an account which has saved worlds, one of the worlds is clicked. | Screen changes from World Select to the Main App Screen, with that specific world being the one that was loaded [fully and properly]. | Screen changes from World Select to the Main App Screen, with that specific world being the one that was loaded [fully and properly]. |
| 6 | Check that user can make a new world, entering a name and seed | (Valid data)   1. Click ‘New World’ 2. Enter world name in ‘Name’ field 3. Enter integer format seed in ‘Seed’ field 4. Click ‘Go’ | A new world is created with that name and seed, and a single terrain layer with values randomised within the set range. | A new world is created with that name and seed, and a single terrain layer with values randomised within the set range.  Graphical user interface  Description automatically generatedGraphical user interface, website  Description automatically generated |
| 7 | Check that user can’t make world with empty name | (Invalid data)   1. Click ‘New World’ 2. Enter blank world name in ‘Name’ field 3. Enter integer format seed in ‘Seed’ field 4. Click ‘Go’ | Screen stays on World Select with pop-up screen staying open, and text is displayed stating that the name field must not be left blank. | Graphical user interface  Description automatically generated |
| 8 | Check that world gets deleted upon delete | (Valid data)   1. Click delete button for a saved world 2. Click ‘Yes’ on pop-up, accepting deletion of world 3. Log out 4. Log back in to confirm it’s been fully deleted | World disappears from the list as soon as ‘Yes’ is clicked, and remains deleted after logging back in too. | World disappears from the list as soon as ‘Yes’ is clicked, and remains deleted after logging back in too. |
| 9 | Check that user can’t use a name for the world that’s already taken (within their account) for world | (Boundary data)   1. Click ‘New World’ 2. Enter already taken name in ‘Name’ field 3. Can just leave ‘Seed’ blank (with the intention of it being random) 4. Click ‘Go’ | Screen stays on World Select with pop-up screen staying open, and text is displayed stating that the name must be unique. | Graphical user interface, text  Description automatically generatedScreen stays on World Select with pop-up screen staying open, and text is displayed stating that the name must be unique. |
| Main App Screen | | | | |
| 10 | Check that changing the terrain layer tile specs works as expected | Without changing any generation settings, adjust the lower bound sliders for a selected Terrain Layer, click ‘[update]’ button. | The mapping of tiles to noise ‘height’ changes appropriately. | The mapping of tiles to noise ‘height’ changes appropriately. |
| 11 | Check that maze layer tile specs system works | Set up the tileSpecs for each tile in the ‘Walls’ tileset. | The maze goes from ‘invisible’ to fully rendered with tiles in the right places. | The maze goes from ‘invisible’ to fully rendered with tiles in the right places.  Shown here: <https://youtu.be/07tJpPoIXlg> |
| 12 | Check that only working values for maze size can be inputted | (Boundary data)   1. Select a maze layer 2. Click the ‘Size’ button 3. Input data that’s outside valid boundaries – e.g. 999 4. Click ‘OK’ | Pop-up box doesn’t close, no size change occurs. ‘Resize:’ text changes to text informing the user of the valid range. | Pop-up box doesn’t close, no size change occurs. ‘Resize:’ text changes to text informing the user of the valid range.Graphical user interface, text, application  Description automatically generated |
| 13 | Check that non-integer value can’t be input for seed | (Invalid data)   1. Click the seed button 2. Try to type alphabetic text in the input field | Only characters that match the regular expression ‘-?[0-9]+’ are allowed | Graphical user interface  Description automatically generatedOnly characters that match the regular expression ‘-?[0-9]+’ are allowed |
| 14 | Check that layer names are made unique | 1. Name layer to be arbitrary name 2. Name another layer to have that same arbitrary name | The makeLayerNamesUnique() method works, layers are updated to have (1) and (2) in their names, differentiating them. | Graphical user interface, application  Description automatically generatedThe makeLayerNamesUnique() method works, layers are updated to have (1) and (2) in their names, differentiating them. |
| 15 | Check that blank rename of world isn’t allowed | (Boundary / invalid data)   1. Click rename button 2. Leave name field blank. 3. Click ‘OK’ | Name label changes to alert user that the inputted name isn’t valid. | Graphical user interface, application  Description automatically generatedName label changes to alert user that the inputted name isn’t valid. |
| 16 | Check that rename to taken name isn’t allowed | (Boundary data)   1. Click rename button 2. Enter an already taken name into ‘Name:’ field of pop-up 3. Click ‘OK’ | Name label changes to alert user that the inputted name isn’t valid. | Graphical user interface  Description automatically generatedName label changes to alert user that the inputted name isn’t valid. |
| 17 | Check that save works | 1. Make changes to a saved world 2. Click save 3. Close the save confirmation pop-up by clicking ‘OK’ 4. Go back to the World Select screen 5. Select the world | Changes are saved, the new changed version of the world gets loaded from the database. | Changes are saved, the new changed version of the world gets loaded from the database. |
| 18 | Check that export works for name without .csv | 1. Click ‘Export’ button 2. Enter a name which doesn’t end in .csv 3. Click ‘Export’ button in pop-up 4. Click ‘Close’ following success message. | File is generated successfully with proper file extension. | File is generated successfully with proper file extension.  Qr code  Description automatically generated  File (success): <https://pastebin.com/pbYGPTDP>  A picture containing text  Description automatically generated |
| 19 | Check that export works for name with .csv | 1. Click ‘Export’ button 2. Enter a name which ends in .csv 3. Click ‘Export’ button in pop-up 4. Click ‘Close’ following success message. | File is generated successfully with proper file extension. | File is generated successfully with proper file extension.  Graphical user interface  Description automatically generated  A picture containing logo  Description automatically generatedFile (success): <https://pastebin.com/QYChcLFS> |
| Tilesets | | | | |
| 20 | Python script can be used to successfully create JSON for new tileset’s map section | Using the script with a small tileset in mind, fill in the appropriate parameters. | Properly formatted output is generated for use in the app. | Properly formatted output is generated for use in the app.  <https://youtu.be/aachuVibvJI> (00:00 until 01:57) |
| 21 | Check that right JSON imports properly and is usable within the app | Create a data.json file with the correct format and a map that works for its tileset, placing these into a new Tileset folder. | Newly created tileset is available for use in the app. | <https://youtu.be/aachuVibvJI> (01:57 til end). |
| 22 | Check that wrong JSON won’t crash program | A screenshot of a computer  Description automatically generated with medium confidenceChange a pre-existing data.json to have an invalid field name. | Program stays running and the broken tileset doesn’t load, but the others do. | Program stays running and the broken tileset doesn’t load, but the others do. |

**Evaluation** Objectives

|  |  |  |
| --- | --- | --- |
| **Objective** | **How I met this objective** | **How well I met this objective** |
| ***New users can register an account to be able to save worlds to the server.*** | There’s a register page, and a database to store world data in its entirety. | I believe that the objective has been fulfilled entirely in terms of functionality. However, I believe that specific improvements could be made. These are listed below in the 'sub-objectives’ of this objective. |
| When creating an account, the user must complete a form with a username and a password. | The Login / Register page has these fields, the values entered into these fields are used to create accounts. | The registration system works well and functions fully, but the following improvements could (and should) be made:   * Separating the login and registration pages would be better than having them on the same page. * Password should be entered twice upon registering to confirm that the user is comfortable typing in their password and knows what it is. |
| The username must be unique within the system. | This is validated upon account generation; not allowing a username if it’s already taken. | This has been met completely in terms of functionality.  This project was my first time using SQL with Java – and there are some changes I would make to my way of interacting with the database in my code in the future; closing connections and using ResultSet objects’ features more effectively. |
| The password must be at least 8 characters long and contain at least one letter of the alphabet, and at least one special character and at least one number. | This is validated upon account generation; not allowing a password if it doesn’t fit the format. | This has been met completely in terms of functionality.  It was achieved efficiently with a regular expression.  In my opinion, the only issue with my implementation of this system is that the UI shifts when displaying the message informing the user of password requirements, which doesn’t look very good.  Text  Description automatically generatedText  Description automatically generated with medium confidence |
| Passwords must be stored in a secure manner – e.g. via a method that uses hashing, or a method that uses encryption. | Passwords are hashed with an SHA-512 hash and a salt. The hashed password and salt are stored in the database. | This has been met completely in terms of functionality.  It seems to me that there’s not any improvement needed – though my understanding of password security and password hashing extends only to what was done in this project! |
| ***To log in, the user must enter their username and password.*** | When a username and its associated password are entered on the Login / Register page, the user is taken to a page with all the worlds of that account. | In terms of functionality, this was achieved fully. However, as mentioned before, it would have been better to separate login and registration pages. |
| The username will be used to find the associated secure password, which will be used to verify that the entered password is correct. | An SQL select statement finds the hashed password and salt associated with that username.  The entered password is hashed using that stored salt, and then compared against the stored hashed password. | Code that does exactly what the objective states has been written, and works well. |
| If the login details are correct, the user is logged in. | If the two hashed passwords are the same, the user is logged in. | This objective has been met completely. |
| If the login details are incorrect, the user is notified that some part of the login was incorrect but is not told what specifically was wrong (as a small way of increasing security).­ | User is not logged if passwords are not the same, and text is displayed stating “Username or password is incorrect” via a Label. | This objective has been met completely. |
| ***The user must be able to save and load their created worlds to and from a server.*** | The database is properly structured for the full saving of worlds, and there are methods which save and load worlds in their entireties. | In terms of functionality, this objective has been fulfilled entirely.   However, there are missing features that I think would greatly improve user experience:   * The ability to rename worlds on the world select screen * Improved world select UI * Some way of previewing a world on world select – like a scaled down simplified version of the world (this would take a lot of work though; involving finding the most prominent colour of a tile and using that to represent it, etc.) |
| Metadata about worlds should be displayed to the user to help them select the one they’re thinking of. | WorldUI.loadWorldsIn() creates TextButtons for every world made by the user, with:   * World name * Date created * Date last accessed * Number of layers * World seed | This objective has been met completely, but I think that more metadata about worlds could definitely be displayed (like the preview mentioned a row above). |
| When selecting worlds, the worlds should be displayed in a non-random order based on a piece of metadata. | The SQL used to get that world data is sorted by Last Accessed – so that more recently edited worlds are at the top. | This objective has been met completely, however there are improvements that I would make developing this further:   * Users should have the ability to sort by any piece of metadata, I think I would likely implement this with a merge sort. * Users should have the ability to search by any piece of metadata, I think I would implement this with a merge sort (to order the worlds) and then a binary search. |
| There must be no discrepancy between the world the user creates, what’s saved, and how it gets loaded back – i.e. saving and loading should work in their entireties, without causing problems for the user. | Database.saveWorld, Database.saveLayer and Database.getWorld achieve this. | The implementation of this works fully. I’ve designed this in a way that allows any new layer type to be added easily – layer-type-specific handling would just need to be added to a switch case – but I think the process could be written much clearer; abstracted into more methods to improve legibility. |
| ***The app must have the capability to display multiple generated layers ‘on top of each other’, in a specific hierarchy.*** | The ‘Layers’ panel contains all the controls required for this functionality. This requirement is also fulfilled by the AppScreen.drawLayers() method. | This objective has been met completely. |
| They must be able to be ‘moved’ up and down within the hierarchy. | This is done with the World.swapLayers() method and the arrow buttons for each layer. | Ideally, I would want users to be able to drag and drop layers to change their order but I couldn’t find a way to implement this with LibGDX’s Scene2D.ui system (which is why there are arrow buttons).  Despite this, I think that the arrow buttons are still quite an elegant and easy to understand solution. |
| Individual layers must be able to be toggled between shown/hidden. | This is done with the layerShown boolean and the show/hide button. | Ideally I would want the ‘[show/hide]’ button to be icon based (like how Photoshop uses an eye) and change based on if the layer is visible or not (opening and closing), but the system in place does work quite well, and there is text informing the user whether the layer is hidden or not. |
| Layers must be able to be edited separately. | Layers can be selected with the \* button, which changes the ‘selectedLayerIndex’, and in turn changes the generation settings panel and tileSpecs panel to relate to the selected layer. | The objective has been met fully in terms of functionality.   Ideally, I would want to the user to just be able to click on the layer to select it but it didn’t seem like there was a straight-forward way of implementing this with LibGDX’s Scene2D.ui |
| The user must be able to add and delete layers, allowing for a variable number of layers. | This is done with the add and remove layer buttons on the ‘layer panel’, and the AppUI.addLayer() method, which creates a pop-up where users can set the tileset for the layer. | The objective has been met fully. Users can create new layers with ease and remove layers with ease. |
| ***There must be a way for a user to import their own tileset.*** | This is achieved via the ‘tilesets’ folder [which users would have access to in a compiled distributed version]. | This objective has been met fully.  The system I’ve designed is just as straight-forward to use as Minecraft’s resource pack system – which has been used by millions of people. |
| As this will mean needing to map out the individual tiles within an overall tilemap, a suitable way of modelling this must be defined. | This was achieved with the Tileset object, JSON format and ‘tilesets’ HashMap. | This objective has been met fully, I think that the use of a HashMap with String keys and TileCorner values works really well. |
| This data should integrate smoothly with the app. | Tilesets are usable within the app. If they’re in the folder, they’ll show up on the drop-down list for selection by the user. | This objective has been met fully in terms of functionality. |
| The import process doesn’t have to be fully within the UI but should be completable by users. | The python script assists users in the creation of the map part of the JSON, otherwise the JSON is quite legible and is the sort of thing that could be copied and pasted (and modified) from a Wiki post. | This objective has been met fully in terms of functionality.  I think that there would be no issues in importing tilesets if the users were provided with a small wiki post describing the layout of the data.json file; the system is straight-forward and easy to understand. |
| ***Users should have the ability to modify the way that tiles interact with the data generated by the algorithm from within the app.*** | The ‘Tiles and aesthetics’ panel contains all the controls required for this functionality. | This objective has been met fully in terms of functionality.  The only issue is that the order of the TerrainLayer tileSpecs doesn’t update automatically, as I couldn’t find a way to seamlessly detect when the user has stopped selecting the sliders of the UI. |
| This must be on a per-layer basis. | Layers each have their own ‘tileSpecs’ ArrayList. When a layer is selected on the ‘Layers’ panel, the ‘Tiles and aesthetics’ panel changes to reflect this, allowing the user to directly edit the tile specs of the selected layer. | This objective has been met fully in terms of functionality. |
| This data must be considered part of layer data, as it is integral to the world that the user designs and therefore must be stored as part of a world save. | TileSpecs are modelled in the database. | This objective has been met fully in terms of functionality.  There are different tables for different types of ‘tileSpec’. I believe that this was the best way of modelling it. |
| ***Algorithms must be ‘seedable’ – in part so layer data is stored efficiently in database.*** | Both the TerrainLayer generation and MazeLayer generation are seedable. With the same seed and other parameters, the exact same outcome is generated. (But different seed same parameters creates something similar looking, but different!) | This was achieved fully.  I think that my system for seeding the maze generation works really well. |
| There must be a wide range of values for seeds, allowing greater customizability. | The ‘Long’ (and bigint in database) data type was chosen for seed so that there could be a wide number of possible values. | This was achieved fully – a Long (or bigint) has a [very big] range of -263 to +263. |
| There must be an overarching world-seed as well as layers having the option to have a different seed. | The ‘inherit seed’ checkbox and boolean, as well as the World.getLayerSeed() method achieve this. | This has been achieved fully in terms of function, however I would have liked to have implemented a system where world seed can be changed. |
| ***Algorithms must be designed in a way that allows for multiple noticeably different outcomes from them.*** | As shown earlier in the imgur post linked to under User Interface - [https://imgur.com/a/8ZinBTU], both the terrain generation and maze generation algorithms allow for a good amount of variety.  (In the case of maze generation, I’d argue that a wide variety of sizes is noticeable variety). | I believe that this objective has been achieved fully. |
| To achieve this, parameters that control the output should have ranges of values that produce a wide range of outputs. | TerrainLayer and MazeLayer have constants for minimums and maximums of the range. These are used for the ranges of the controlling sliders. | This objective has been met completely. The ranges allow for a wide scope of output variety. |
| Algorithms must be able to produce at least three highly distinct outcomes – e.g. three different, characterful outcomes with their own ‘personalities’ rather than three quite plain shapes that might as well just be the same one. | TerrainLayer generation:   * The ‘wrap’ feature (as well as the invert) brings a lot of variety to the outputs * With the range of input values, the outputs are highly varied. * With the tileSpecs system and the way that tiles can be mapped to the terrain generation, even more variety is created.   The screenshots in the imgur post [https://imgur.com/a/8ZinBTU] show that terrain generation has a noticeably wide range of outputs.  MazeLayer generation:   * Has a range of sizes from 3-857, which equates to a range in area of 9 tiles – 734449 tiles. * Customisability of tiles allows for variety of ‘looks’ for the maze.   The screenshots in the imgur post [https://imgur.com/a/8ZinBTU] show the wide range of sizes that maze generation can produce. | I believe that this objective has been achieved fully, the variety of outputs produced by both the terrain generation system and maze generation system is really strong.  In terms of variety, the maze generation is definitely the weaker algorithm (although I would argue that this is to be expected). This could be improved upon by implementing a directional bias feature. This could be done by changing the ‘random directions’ method to always have a specific direction (horizontal/vertical) first, depending on what the user has selected.  A horizontally biased maze looks like this:  Maze generation algorithm - Wikipedia |
| ***Users should be able to use what they’ve created outside of just my software.*** | This is achieved via the CSV Export feature and also the ability to hide the UI to make screenshotting cleaner and simpler. | This objective has been met fully in terms of functionality.  CSVs are widely used and highly legible; I think they were a decent choice for export.  Ideally, I would want to provide the ability to export to as many file types as possible – specifically file formats which integrate directly into game development frameworks and image formats. |
| Users must be able to export a portion of their world to a pre-existing file format for use elsewhere. | CSV Export feature - export button and FileHandling.export() method - fulfils this.  Users specify the corners of the export region and the filename. | This objective has been fully met.  A feature I would want to add is the ability to drag and select the rectangle area they want to export, but users still have the ability to precisely select the export region even without this implemented. |
| ***UI should be designed in a way enables easier use of the software.*** | Throughout the development process, I have made my choices with user experience strongly in mind.   * I have taken influence from the layout of photoshop in the design of the panels in the app screen in order to provide familiarity. * Controls are organised by function -e.g. layer controls are grouped together * Layers are selected and then the relevant controls are shown, rather than cluttering the user’s view with more and more windows with each new layer. * Back buttons enable seamless navigation between screens | I believe that this objective has been met completely. |
| Tips and guidance for controls etc. should be displayed. | There is a Label at the bottom of the screen with control ‘tips’ that would not otherwise be obvious. | This objective has been met fully in terms of functionality, however I would ideally want to implement a more dedicated more detailed controls guide section.  Additionally, I would have also liked to implement a section where users can customize keybinds etc. |
| Coordinates for the user’s current ‘position’ should be displayed. | There is a Label displaying [live-updating] user ‘position’ within the world. | This objective has been met fully.  A feature where users can directly state what position they want to be at (x & y) would improve user experience. |
| Icons should either use easy to understand symbols or appropriate similarly easy to understand text. | Except for commonly accepted symbol use, most buttons are text-based. | I believe that this objective has been met fully.  Ideally, text-based buttons would be able to be in any language. I could use Google’s translation API to achieve this dynamically. |
| The UI, or most of it, should be able to be hidden - for easier software use and so that higher quality screenshots can be taken. | Pressing TAB hides all of the UI except the drawn tiles and a message reminding the user how to unhide everything (by pressing TAB again). | This objective has been met fully.   While the screen can be cleared to take better screenshots, I think that an image export feature would be of great use to users of the app. This would work similarly to the file export system, with users specifying the corners of the export region. This would allow the user to export extremely large images at full resolution (whereas screenshots wouldn’t be able to capture that due to monitor size / resolution). |

I think that every objective has been met to a good standard, and I think it’s worth noting that I plan to use an analysis-informed objectives system when working on projects in the future as I found it useful for scoping out the project and keeping on track to creating what I believe is a meaningful outcome.

**User feedback**

I was able to get a couple of people to test my system (I showed them the python script and how to do the tileset import, but they were fully in charge of that), and answer some questions.

User 1

*What did you like about the app?*

The layout was intuitive and quite easy to use.

I really liked the things I was able to make, I think that the terrain generation is really cool.

*What did you dislike about the app?*

The system for selecting the maze tiles was a bit tedious, but it seems like the best solution.

I wanted to change the tileset of a layer, keeping the settings the same but changing the look, but couldn’t.

I was able to use the python import for a tileset, but would prefer it to be more within the app.

What changes / features would you like to see?

It would be cool for the mazes to have entry and exit points.

A copy / paste feature within the app would be really useful (particularly for generation settings).

User 2

*What did you like about the app?*

The page navigation system worked well.

The maze mapping feature is awesome! Really satisfying seeing the tiles kinda load in as you define them.

The level of customisability of the layers seems really powerful!!

*What did you dislike about the app?*

I wish the controls were more tactile, there are a few cases where there a buttons but there could be more direct interaction. The ability to select stuff in the view window bit might also really improve user experience.

What changes / features would you like to see?

The generation algorithms are really cool but I would really like there to be a lot more.

Image export would be amazing – and (as an artist and game developer) I think image export is more important than tilemap file export, as many game engines do have procedural generation options and it’s usually easier to just do everything within the same piece of software (despite this app being able to produce some really cool stuff), but software like Photoshop doesn’t have anything like this!

Click sound effects could be fun.

More generation algorithms would always be good.

**Improvements / Extensions**

Based on user feedback and aspects that I’ve identified myself, I’ve come up with a list of features / changes for the app.

* There should be more generation algorithms (layer types) available, with the intention of bringing more variety to the system.
  + One of these could be terrain that’s wave-function-collapse based – similar to what I proposed in Analysis?
  + I could also do a user defined structure system that uses wave-function-collapse. This would involve creating a pop-up window where the user can draw (possibly drawing with pixels, possibly drawing with tiles) – similar to this: <https://youtu.be/Ten6MIWd2DA>
* I would like to enact upon the UI changes that I suggested in the Objectives section of evaluation, as well the controls suggestions suggested by User 2.
* The addition of sound effects for button presses to improve user experience.
* A settings menu with controls for screen resolution, sound effects volume, keybinds to improve user experience.
* A hue-shift feature on a per-layer basis – this would likely be implemented with shaders – where the entire colour of a layer can be shifted, in order to get more use out of a tileset.
* A copy-paste feature, which would allow users to store generation data to the clipboard – this could possibly be achieved as a JSON string with an identifier at the start to declare exactly what’s been copied?
* The direction-bias feature for maze generation that I mentioned earlier in evaluation would make maze generation a lot stronger.
* The implementation of more file formats that can be exported would greatly improve the app’s functionality as a tool that can be used in a larger creative process.