cliMate

Team 18 - Design Document

Rami Bitar, Ben Denison, Nathan Raine, Alex Shelley, Jason Shipp

Purpose

Roleplayers, worldbuilders, and other fantasy minded people have been looking for a tool that would eliminate the need to design world terrain and map accurate biomes by hand. This multi-stage mapping is extremely difficult manually, so our project aims to automate this process by generating biomes for terrain files, generating random terrain, and allowing users to edit their worlds in a well-designed desktop application.

Functional Requirements:

1. Generating Biomes:
    As a user…
    - I want to have the program automatically assign biomes to different regions.
    - I want to manually add some biomes as a starting point, and then let the algorithm fill in the rest.
    - I'd like to see the temperature of each area displayed on my map.
    - I want to have a saved copy of the completed map for use.
    - I want to see wind patterns displayed on the map.
    - I want to be able to enable or disable the generation of certain biomes.

2. Loading and Generating Terrain Heightmaps:
    As a user…
    - I want to import terrain that was previously created.
    - I want to import only a coastline and generate biomes from that.
    - As a user, I want to add noise to a height map.
    - I want to procedurally generate terrain as an alternative to importing or drawing one.
        - I want to tweak parameters used for terrain generation.
    - I want to be able to hand-craft my own terrain within cliMate.
    - I want to navigate and view the currently imported terrain.

3. Allowing User Edits:
    As a user…
    - I want to tweak terrain after heightmap generation.
    - I want to be able to edit the biomes after they're generated.
    - I want to add notes to my map.
    - I want the software to remember my most recent file and let me load it without browsing for it.
    - I want to be able to undo my last action, preferably many times

Non-functional requirements:

1. Exporting to image files:
    As a user…
    - I want exported maps to not have unsightly distortion.

2. Time for calculation:
    As a user…
    - I want terrain generation from noise and climate calculation to not take more than a few seconds.

3. Accuracy of climate calculation:
    As a developer…
    - I want climate calculation output to be reasonably similar to earth climates when applied to a height map of earth with the correct earth sea level.

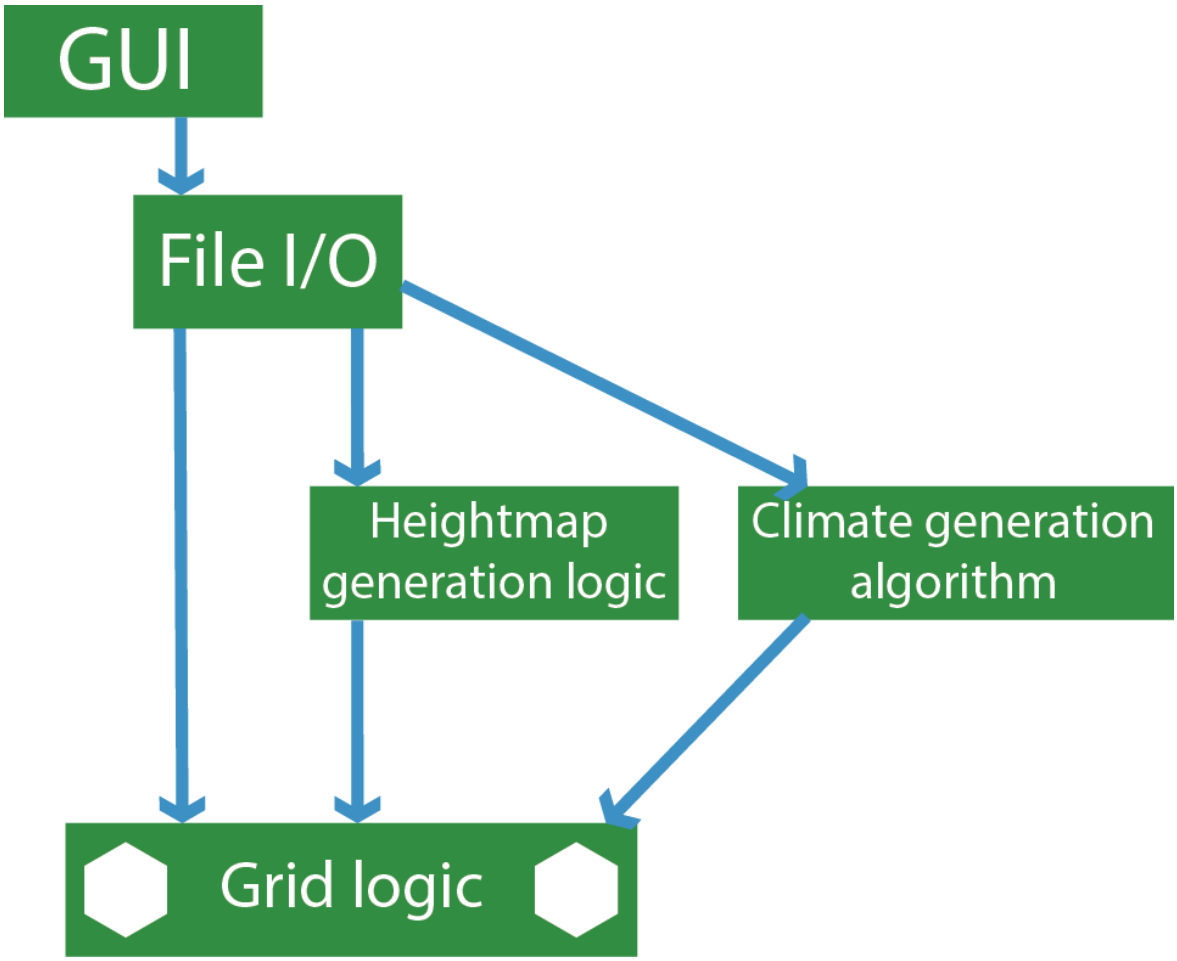
4. Verisimilitude of plate tectonics:
    As a user…
    - I want tectonic simulation to generate similar structures to those created by real plate tectonics e.g. coastal mountains, ocean rifts, and continental shelves

5. Ease of use and Appearance:
    As a user..
    - I want the program to be easy enough to use without need of tutorial or guidance of any kind
    - I want the same look and feel on every stage of the GUI

Design Outline

High Level Overview

Our project is a desktop application that generates biomes for terrain maps. Our implementation will use a Multi-Layered Pattern model. The biomes generator will access data stored in a terrain file and use the given information on height maps, latitude in relation to the equator, and ocean currents to generate accurate biomes on the landmass.

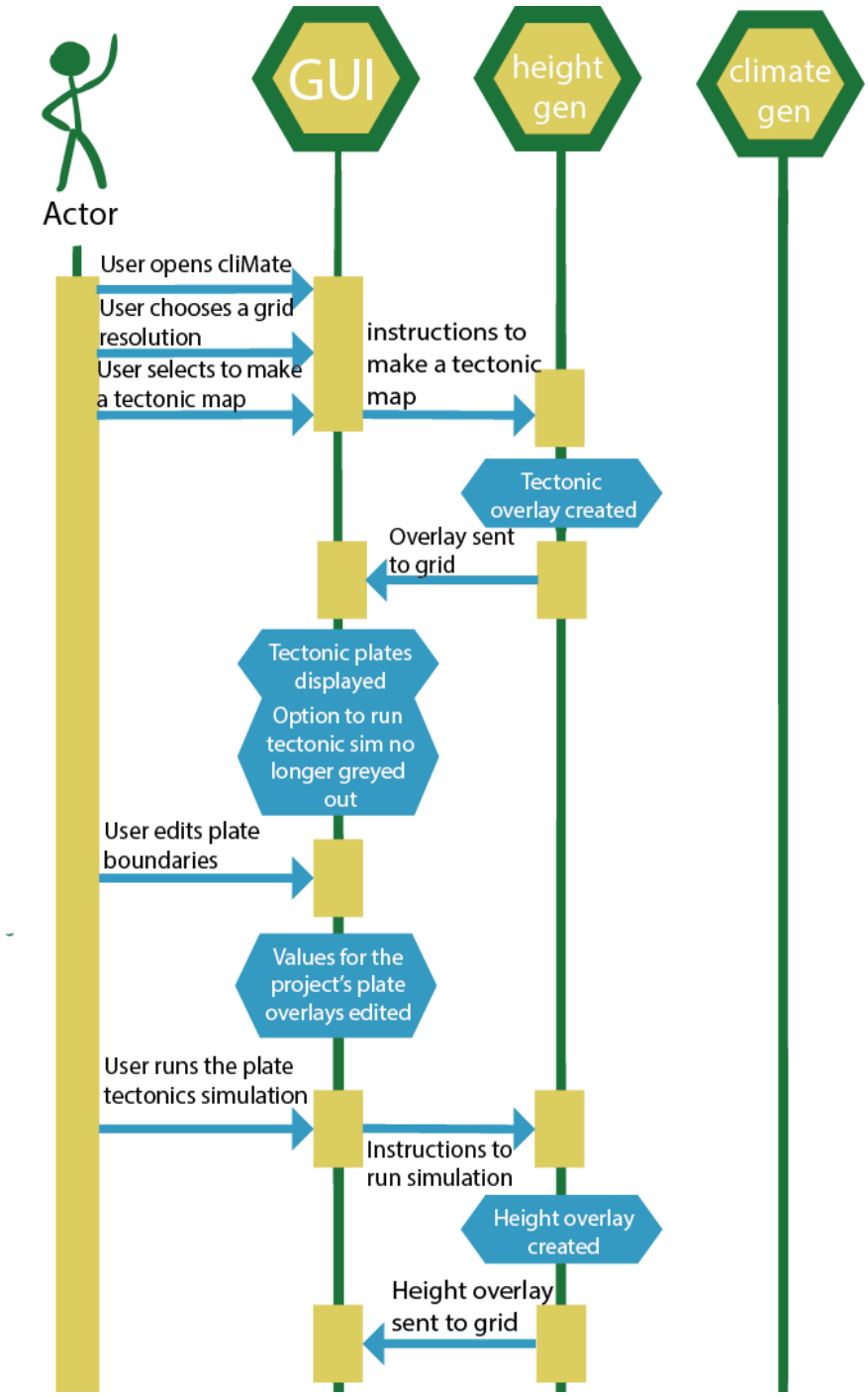


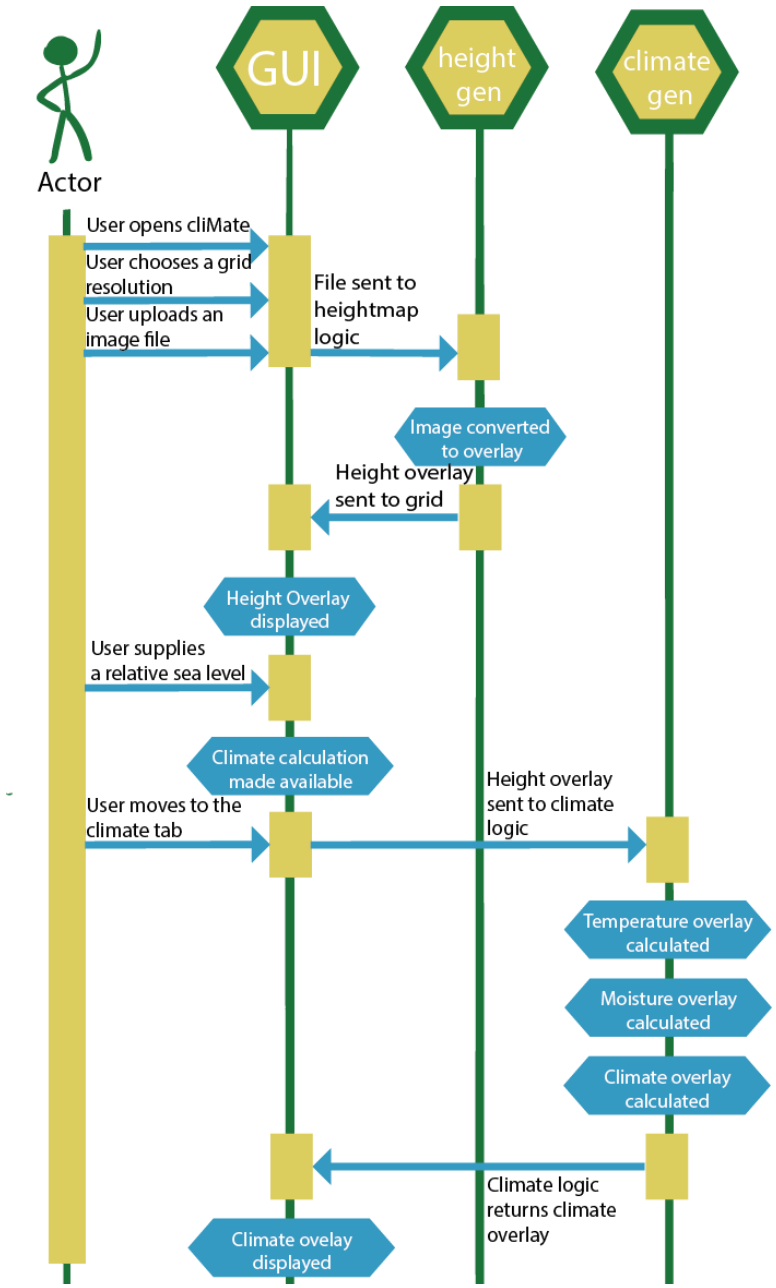
1. Desktop Environment (GUI):
    a. The windowed application will be done in C#.
    b. The windowed application will be given information about the terrain from an image file.
    c. The windowed application will display information about the generated biomes in a few different tabs
    d. The windowed application will allow users to open, view, and save usable image files of the terrain

2. Biome/Climate Generation Algorithm:
    a. Height can be generated by...
        i. Methods of terrain generation using parameterized noise
        ii. Methods of terrain generation using a rudimentary implementation of plate tectonic movements.
        iii. Getting data from an image file supplied by the user
    b. Climate Generation:
        i. Climate generation will only require a heightmap and a user-supplied sea level value
        ii. Climate for a given area will be determined using the Whitaker scheme (displayed later)
3. Grid Logic:
    a. Will supply the rest of the application with tools to modify and display the grids
    b. Logic will be based on icosahedral projections/maps

Flow of Events

The diagram below shows a typical biome generation. The sequence begins when a user opens to software application from their local computer. The user will then chose to either input one of their terrain picture files, generate a random terrain file, or start from a blank slate. The user will then have the option to tweak their heightmap using the heightmap editor. Next, the user will generate biomes based off of the given terrain. Next, the user will have the option of tweaking the generated biomes using the biome editor. Finally, the user will save their terrain and biome data, which can now be interpreted by other software.


GUI
height gen
climate gen
Actor
User opens cliMate
User chooses a grid resolution
User selects to make a tectonic map
instructions to make a tectonic map
Tectonic overlay created
Overlay sent to grid
Tectonic plates displayed
Option to run tectonic sim no longer greyed out
User edits plate boundaries
Values for the project's plate overlays edited
User runs the plate tectonics simulation
Instructions to run simulation
Height overlay created
Height overlay sent to grid


GUI
height gen
climate gen
Actor
User opens cliMate
User chooses a grid resolution
User uploads an image file
File sent to heightmap logic
Image converted to overlay
Height overlay sent to grid
Height Overlay displayed
User supplies a relative sea level
Climate calculation made available
User moves to the climate tab
Height overlay sent to climate logic
Temperature overlay calculated
Moisture overlay calculated
Climate overlay calculated
Climate logic returns climate overlay
Climate ovelay displayed

Design Issues

Functional Issues:

Issue: What programming language should we use?

- Java
- C++
- **C#**

We decided to go with C#. Java has some frameworks that could be forced into complying with our situation, but not as good as Unity or OpenTK. C++ also has nice drawing functions, and fast vector operations which would prove useful, but the threat of memory management is a major downside. Overall C# offers all the advantages of a Object-Orientated language, while providing access to OpenGL, Unity, and OpenTK.

Issue: What (if any) graphics framework should we use?

- Option 1: Unity
- **Option 2: OpenTK**
- Option 3: No additional package

We chose to use OpenTK as it abstracts away a substantial amount of the drawing and OpenGL programming necessary to implement in Unity. We are only going to use the framework to project the icosahedrons from a sphere onto a flat surface, the map, or vice versa. This would be very difficult without any package, but using OpenTK wrappers it can be achieved.

Issue: How should we break up the map for calculations?

- Leave it as a bitmap
- **Transform it to an icosahedron**

We are going to transform the map into a icosahedron in order to avoid excess distortion and breaks when the nodes move, specifically during tectonic simulation.

Issue: How should we store the information about the nodes for later retrieval?

- Rectangular Array
- **Augmented Triangular Matrix**
- Just store the map and regenerate upon load

We are going to store in the augmented triangular matrix. When trying to store the nodes, which are best represented as the hexagons and pentagons of the icosahedron, in the custom format it is difficult to directly map them to any existing data structure. They can, with some difficulty be forced into a rectangular array, but the retrieval from those is complex. In addition it is likely we would be forced to use empty blocks in the array. The Augmented Triangular Matrix would allow us to map faces of the icosahedron directly into either the upper right triangle opposite other faces without distortion. To store the bitmap and simply regenerate the climates it upon loading would be to computation heavy and slow down the process.

Non-Functional Issues:

Issue: What type of architecture should we use?

- Option 1: Pipe and Filter System
- **Option 2: Multi-Layered Pattern**

We opted for the Multi-Layered Pattern architecture for its ability to logically abstract out the different components of the actual biome generation. There is a natural flow of events which need to be completed semi-sequentially, and can report back to the higher levels of abstraction once complete. The Pipe and Filter System was originally appealing because it allowed for the path of data to go from method to method, but as we do not have a constant stream of input and we have several layers of user input the Multi-Layered Pattern made more sense.

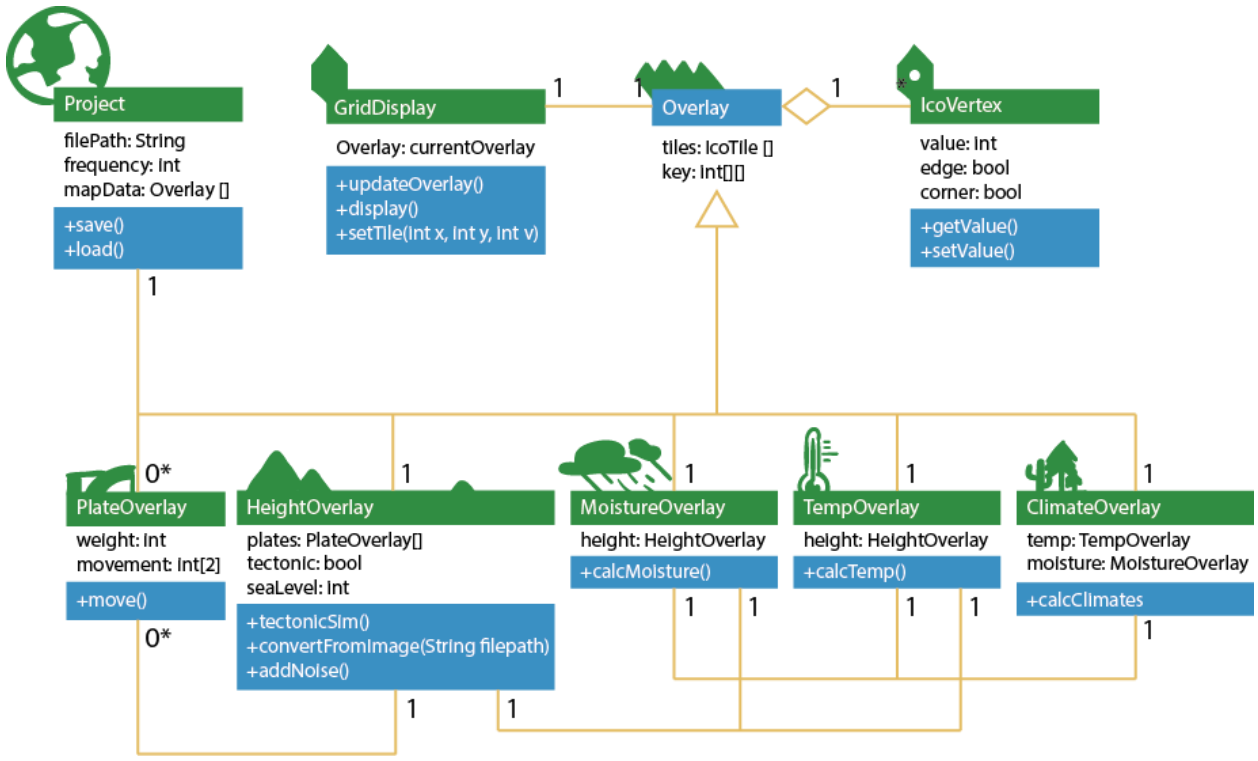
Issue: How should we allow users to switch between different overlays?

- **Option 1: Multiple Tabs**
- Option 2: Dropdown Menu

We decided to go with Multiple tabs over dropdown menus to switch between multiple overlays. Tabs are clearly visible and anyone who has used an internet browser is familiar with them, but dropdown selection is less common, sometimes contains hard to read text, and you’re more likely to click on the wrong option with a dropdown menu than tabs.

Design Details

Data Class Level Design



Description of Data Classes and Their Interactions

Project - an array of Overlays that can be pulled from or saved to custom file format denoted .cli.

GridDisplay - an openTK construct that displays information from overlays in the form of a flattened icosahedron.

Overlay - a class upon which all other overlays implement. It contains a matrix of icoTiles. Each coordinate in the matrix maps to some hexagon or pentagon on an icosahedral map for the GridDisplay. Each overlay also contains a key that lets the display know how to color each tile based on its value.

IcoVertex - a node representation of a tile that contains values for a tile. These values include whether the tile is a corner or on the edge. Values will have getters and setters accessible to all instances of Overlay.

PlateOverlay - Overlay extension that represents a tectonic plate. Vertices would be given a value of 0 or 1 indicating whether or not it is a part of the plate. This will contain the method move() which will translate all tiles that are part of the plate along the grid.

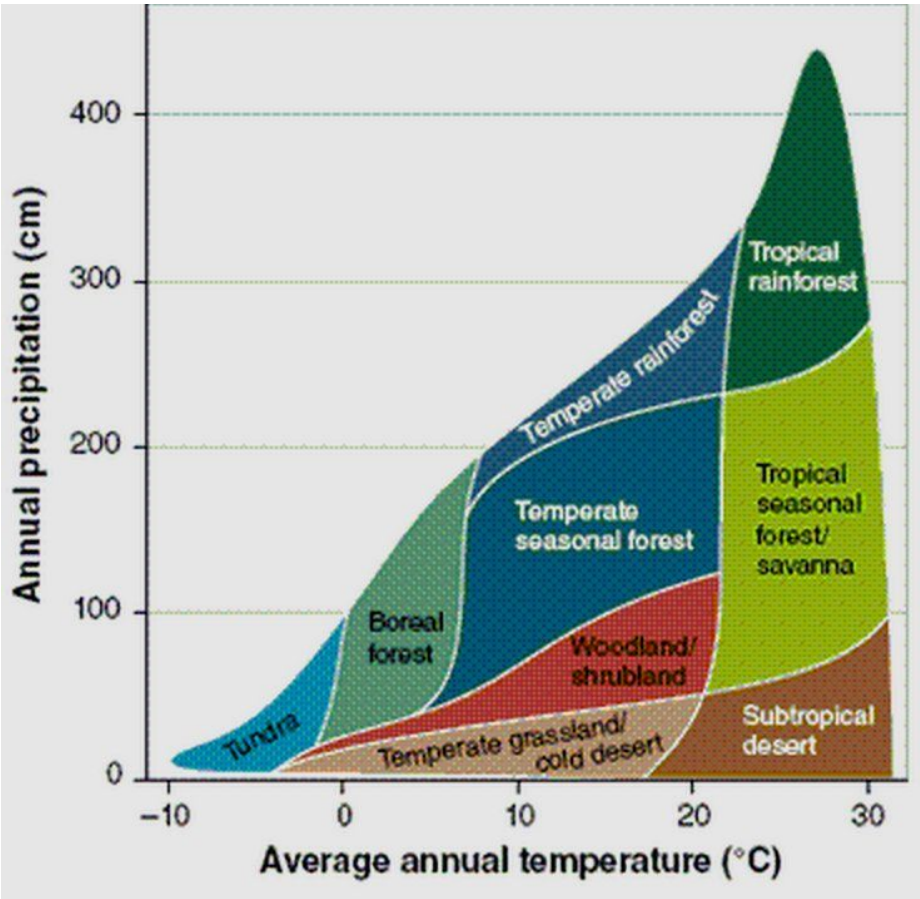
HeightOverlay - Overlay extension that shows the topography of the icosahedron that represents the project's world. Each tile in a height overlay will have a byte value assigned to it to represent height. The key will instruct the map to display tiles as grayscale values. Sea level will be determined by the user and will be used to determine the boundaries of continents for later calculation. Height Overlay will contain methods to start up a tectonic simulation. This will set the boolean value "tectonic" to true and instantiate an array of PlateOverlays that will be attributed to the project's height overlay. These can be used later to display plate boundaries. convertFromImage takes a filepath to an image and projects it onto an overlay, converting it to grayscale and warning the user if it already isn't.

MoistureOverlay - Overlay extension that is used to calculate moisture for climate determination. This class has access to the Project's HeightOverlay object and its data. The calcMoisture() method will use data from the HeightOverlay to determine the shape of the continents in order to get an approximation of air pressure and therefore moisture. MoistureOverlay's key is arbitrary and will have a default case and something the user can redefine.

TempOverlay - TempOverlay, much like MoistureOverlay, is mainly for calculating climate later, but has been left a project attribute in case the user wishes to display or export it later. TempOverlay uses data from the Project's HeightMap to calculate a given tile's average temperature. The key for this overlay is also arbitrary and may be changed by the user later based on their preference.

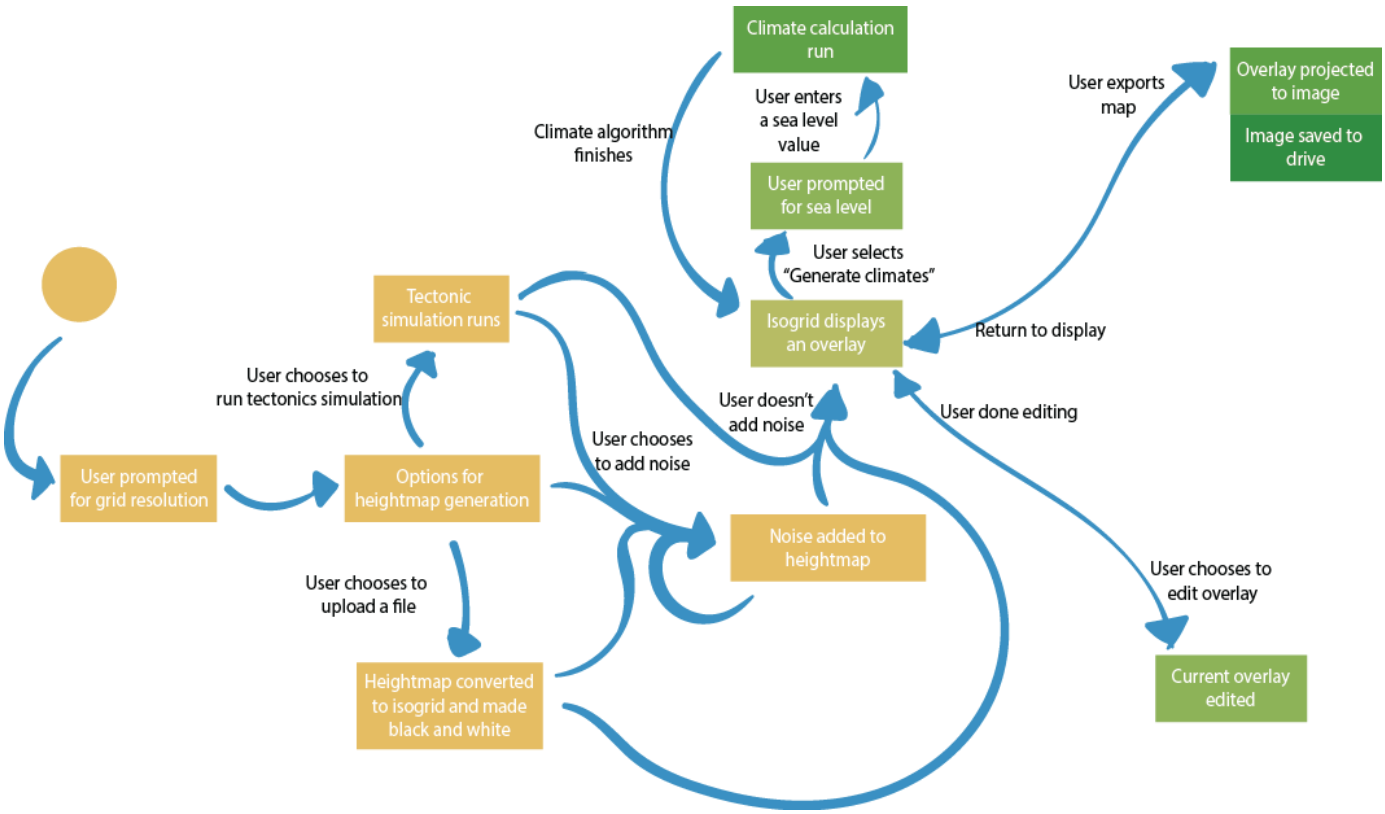
ClimateOverlay - This will use the data from the Project's TempOverlay and MoistureOverlay to calculate climates based on the following chart:

(Whittaker's Scheme)



State Diagram:

- For creating a new map from scratch the user is first prompted for the grid resolution they wish for their map to be. They are then allowed to choose to upload a height-map or generate one via noise or a tectonic movement simulation. If they choose to generate via tectonics or uploaded their own map they are still allowed to add noise to the file for more variety. After having a map with or without noise the Isogrid displays one of the overlays defined above. This then generates a loop from which users can generate the climates, export the map, or edit different overlays.

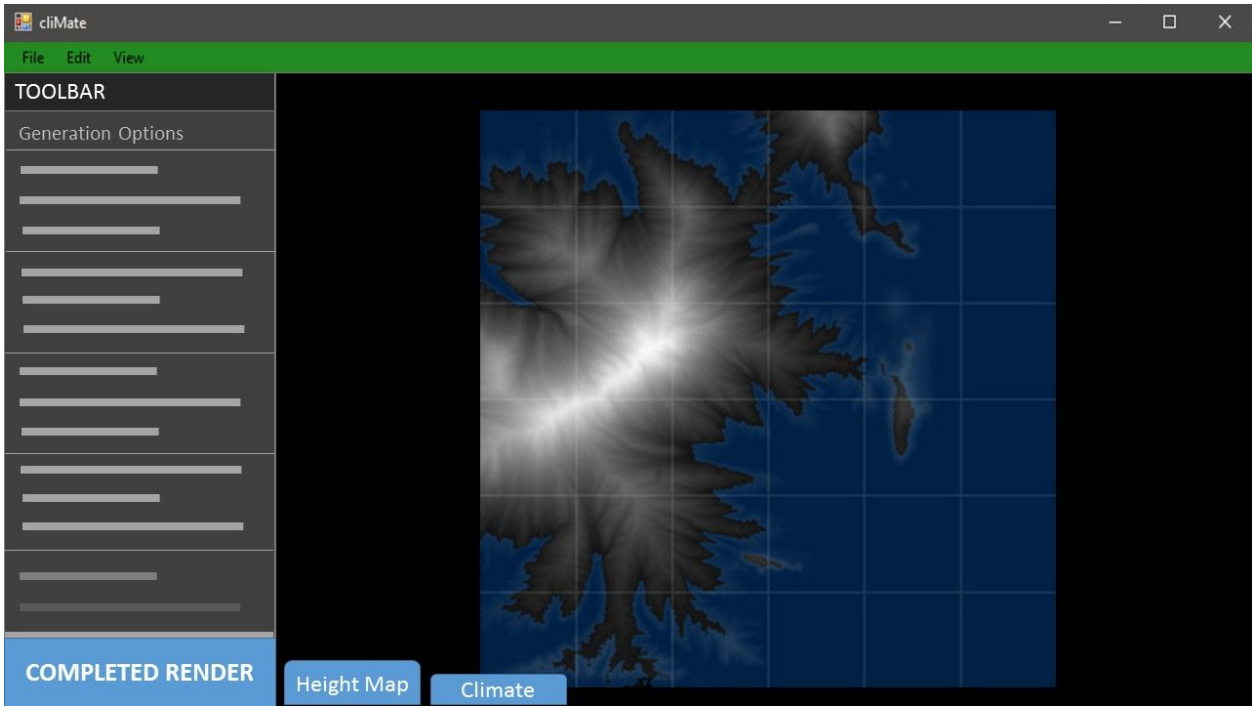


UI Mockup

- **Splash Screen to open on launch:**

- **Program at initial start point:**

- **Terrain with Generated Biomes (Heights Tab):**



- **Terrain with Generated Biomes (Climates Tab):**