Olivia Barnett

2024-2025

Abstract

A tool that generates weather based on a number of conditions, such as climate and season, to create immersive world building through its dynamic nature.

TECHNICAL DESIGN DOCUMENT

Climate-Based Dynamic Weather System

Table of Contents

[Glossary 2](#_Toc190653773)

[\*Introduction 3](#_Toc190653774)

[List of Features 3](#_Toc190653775)

[Planning 3](#_Toc190653776)

[Engine Requirements 4](#_Toc190653777)

[Logic Requirements 5](#_Toc190653778)

[Visual Requirements 5](#_Toc190653779)

[Development Project Requirements 5](#_Toc190653780)

[UI 5](#_Toc190653781)

[Platform Specifications 6](#_Toc190653782)

[Inspiration 6](#_Toc190653783)

[Research 6](#_Toc190653784)

[Purpose of Use 8](#_Toc190653785)

[Potential Upgrades 8](#_Toc190653786)

[Diagrams 9](#_Toc190653787)

[Horizontal Prototype 14](#_Toc190653788)

[Polished System 15](#_Toc190653789)

[Test Plan 16](#_Toc190653790)

[\*System User Manual 21](#_Toc190653791)

[Bibliography 23](#_Toc190653792)

# Glossary

EUW – Editor Utility Widget

WV – Weather Volume

NS – Niagara System

# Introduction

The weather tool will create a weather system for the game on which it is used. It will create realistic weather by selecting attributes based on a number of settings that the user can choose – most notably, climate, but also season and time of day. The aim of this tool is to help accelerate development by allowing the designer to create weather that matches their game’s world with only a few clicks, rather than having to make their own system from scratch. It will be usable in any type of game, but is envisioned mostly for open-world or life-simulation games. By placing a Weather Volume in the scene, weather that the tool generates will be spawned inside of it, and the amount of precipitation that spawns with it will scale accordingly to the size of the volume. Multiple WVs can be placed and will all be affected by the tool, allowing for some areas in maps to have weather present while others are excluded – this could be helpful in areas such as forests, where the designer may not want rain to spawn in areas of densely packed trees, but still wants it outside.

# List of Features

* Editor utility widget with numerous input points:
  + Climate selection dropdown menu
  + Season selection dropdown menu
  + Erraticism value slider
  + Enabling of a day/night cycle checkbox
  + Day length value slider
  + Generate button, to set the weather settings and create ethe weather for the volumes
* Weather volumes, used to contain the weather that is spawned
* A day/night cycle, controlled by the user widget, that will move the sun in the sky at a rate also determined by the user
* Extensive weather data for five different climates and their information for all four seasons
* Precipitation that will spawn randomly depending on its frequency in the current climate|season
* Wind that will affect the precipitation in the world
* Built-in rain and snow Niagara systems
* Ability to place multiple Weather Volumes in the scene
* Scaling capabilities between the weather spawned and the size of the volume
* Dynamically changing weather that transitions at a rate determined by the user
* Easy generation of weather for any game world

# Planning

Most of the planning for this tool is being done by hand, in the form of mind maps and bullet-pointed lists. I began by brainstorming the general areas I knew I wanted and needed to include, such as climates, seasons and a day night cycle, as UI considerations. I then branched out into thinking about how I would store and represent the data I was creating. Following this, I made lists of steps I needed to take in my development process. Some digital planning is also place, and the most helpful plan I made was a spreadsheet listing different weather attributes and a rough idea of what value it should have associated.

# A notebook with writing on it AI-generated content may be incorrect.Engine Requirements

The tool will be developed in Unreal Engine 5.4 but should be compatible with older versions of Unreal Engine 5.

# Logic Requirements

To generate the weather, the program will have to perform a series of calculations, based on values it receives from the user’s selections in the EUW, and from the corresponding row entry in the data table. The results of these calculations will differ depending on the intended intensity of the weather that is being spawned. The program will also have to dynamically find any Weather Volumes that have been placed in the scene, scale the weather to them, and then pass through the results of the calculations. Finally, functionality for transitioning between weather states will have to be implemented to allow for dynamic weather.

# Visual Requirements

Niagara systems will be used to create weather features, such as rain and snow. These will have parameters that can be affected by the program.

# Development Project Requirements

The project in which the tool is developed will need to be a project that can be altered in C++, as I will need to go beyond the capabilities of Blueprints. The game scene will be mostly empty, but I will add an object that casts a shadow as a way to test the sun movement in my day/night cycle. All walls and floors will be untextured so as to see the particles that spawn clearly.

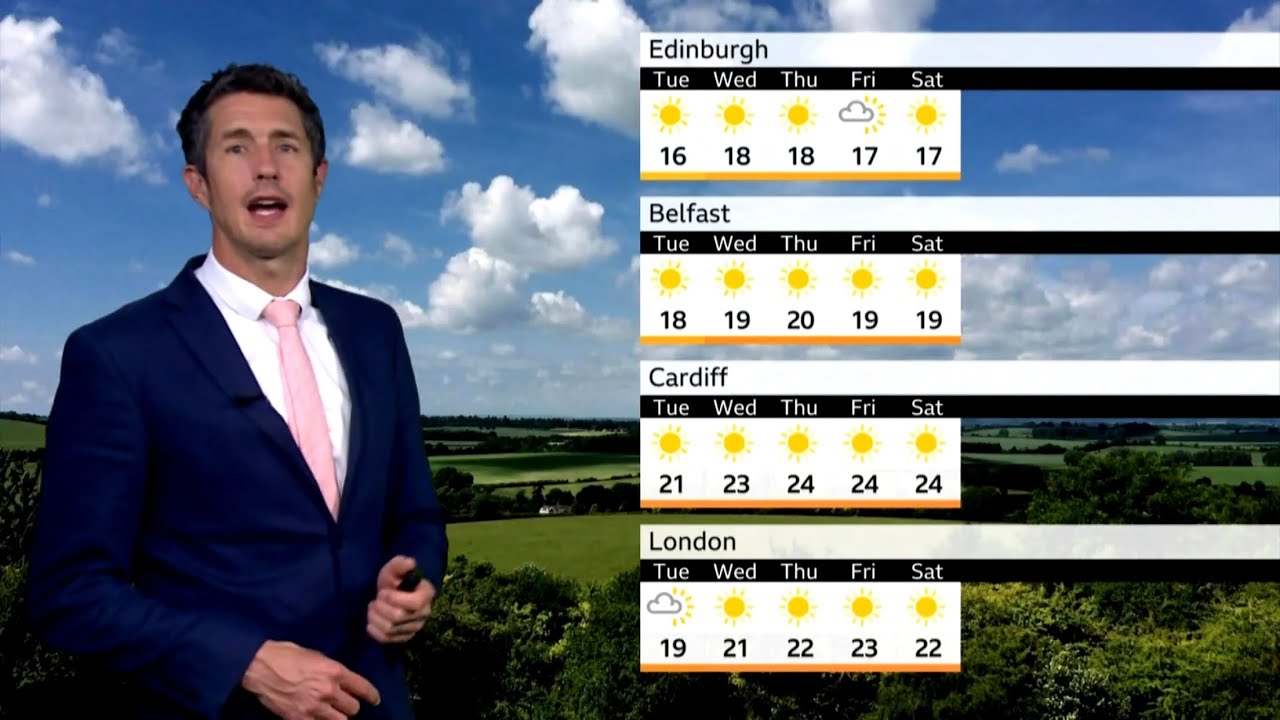
# UI

The UI of the tool needs to be simple as to what each setting is, and it needs to be easy for the user to be able to select the value they want. For the user’s inputs, the UI will include combo boxes (drop-down menus), spin boxes (sliders with type-in capability for choosing a value within a range), and checkboxes. In order to keep to the weather theme, it will be designed similarly to a televised weather broadcast, with a landscape picture showing a cloudy sky as the background. The button to press to generate the weather will be large and obvious, so that there is no confusion as to what has to be done in order to get the system working.

A screen shot of a weather system

AI-generated content may be incorrect.

**Preliminary UI design**



**Example of weather forecast being used for inspiration**

# Platform Specifications

The tool requires relatively high processing power and graphical capabilities, so is more aimed at being used on projects being created for PC games or powerful home consoles, rather than games for mobile devices.

# Inspiration

As a fan of open-world games, I have always been interested in the small aspects that help to make a world feel immersive. In a previous project, I began developing a game in an open-world style, and, while I was able to create quite an involved scene through landscaping and decorating, I realised that something that would have made it feel much more “alive” would have been weather that changes throughout runtime. Games in franchises such as *The Sims* and *Animal Crossing* include wide landscapes with changing weather, and although it is not a core feature of the gameplay, it creates a more natural atmosphere, and subconsciously keeps the player’s interest and prevents the game from becoming stale.

# Research

#### Games

In addition to the previously mentioned games of *Animal Crossing and The Sims*, I also looked at the weather present in *SimCity*. As the day goes on, rain can be seen forming across the map, alongside wind, and there is a day/night cycle, with the two stages of equal duration. *SimCity* was my main inspiration for my tool, as its weather does not have any impact on gameplay, whereas there are small elements within *Animal Crossing* and *The Sims* that work differently depending on the current weather. I wanted what I produced for this project to be completely for aesthetic purposes: to help with the game’s environment, but not impact the actual gameplay that design teams separate from myself would be implementing alongside it.

I specifically looked into *The Sims 4 (2014)*’s ­*Seasons* expansion pack, to try and gauge how well my tool could be received. Reading a myriad of reviews told me that, for many users, having seasons and weather as a part of their gaming experience made the world feel a lot more expansive, and helped them feel more connected with their characters. Many have also said that they felt that this weather/season functionality should have been included in the base game, rather than only available as an expansion pack. This research told me that weather is something that players are looking for in games, and would prefer to have it being in games similar to *The Sims 4*  as standard.

A screenshot of a computer

AI-generated content may be incorrect.

**Critic reviews for *The Sims 4: Seasons*** *(sourced from metacritic.com)*

#### Weather

I hoped to make my tool align with the weather of the real-world. To ensure this would be the case, I looked at multiple data sources studying the weather tendencies in five world climate types: arid, mediterranean, polar, temperate and tropical. I collated a large amount of information and created a spreadsheet to keep track of the intensity of different weather features for each climate and for each season. I made sure that the sources I used featured official meteorological data for the best results.

A table with different weather conditions

AI-generated content may be incorrect.  
**An example of weather data I used for my research** *(sourced from Climate Box, a platform from the United Nations Development Programme)*

# Purpose of Use

The weather tool is something that can be used by designers to speed up development time. If they want a weather system in their game in order to add more detail to their scenes, they can add the tool to their project, select what kind of weather they will need, and, once placing a weather volume in their scene, it will work and be present instantly. If the weather is not an integral part of gameplay, developing a system would most likely be of lowest priority, and could be potentially overlooked or left behind if there turned out to not be enough time or resources to work on it. Instances like this are where the tool would be perfect, as it would take a very short amount of time to add and setup the tool compared to starting developing a system from scratch.

# Potential Upgrades

A tool of this type has a lot of room for improvement:

* Fine-tuning weather generation to correlate more with the real world could go on indefinitely, as there are hundreds of small changes that could be implemented to account for further unpredictability
* More climates, or sub-climates could be added for further customisation
* Additional weather features could be added, such as rainbows or hurricanes

The way in which the tool is made means that developing it further would be very easy to do – all that would need to be done is add extra rows or columns to the data table and then represent this in the Weather Selector EUW. There is no limit for what could be added to the tool, so long as the user has a way of accessing it.

# Diagrams

A diagram of a computer program

AI-generated content may be incorrect.

A diagram of a flowchart

AI-generated content may be incorrect.

A diagram of a flowchart

AI-generated content may be incorrect.A diagram of a graph

AI-generated content may be incorrect.

A diagram of a process

AI-generated content may be incorrect.

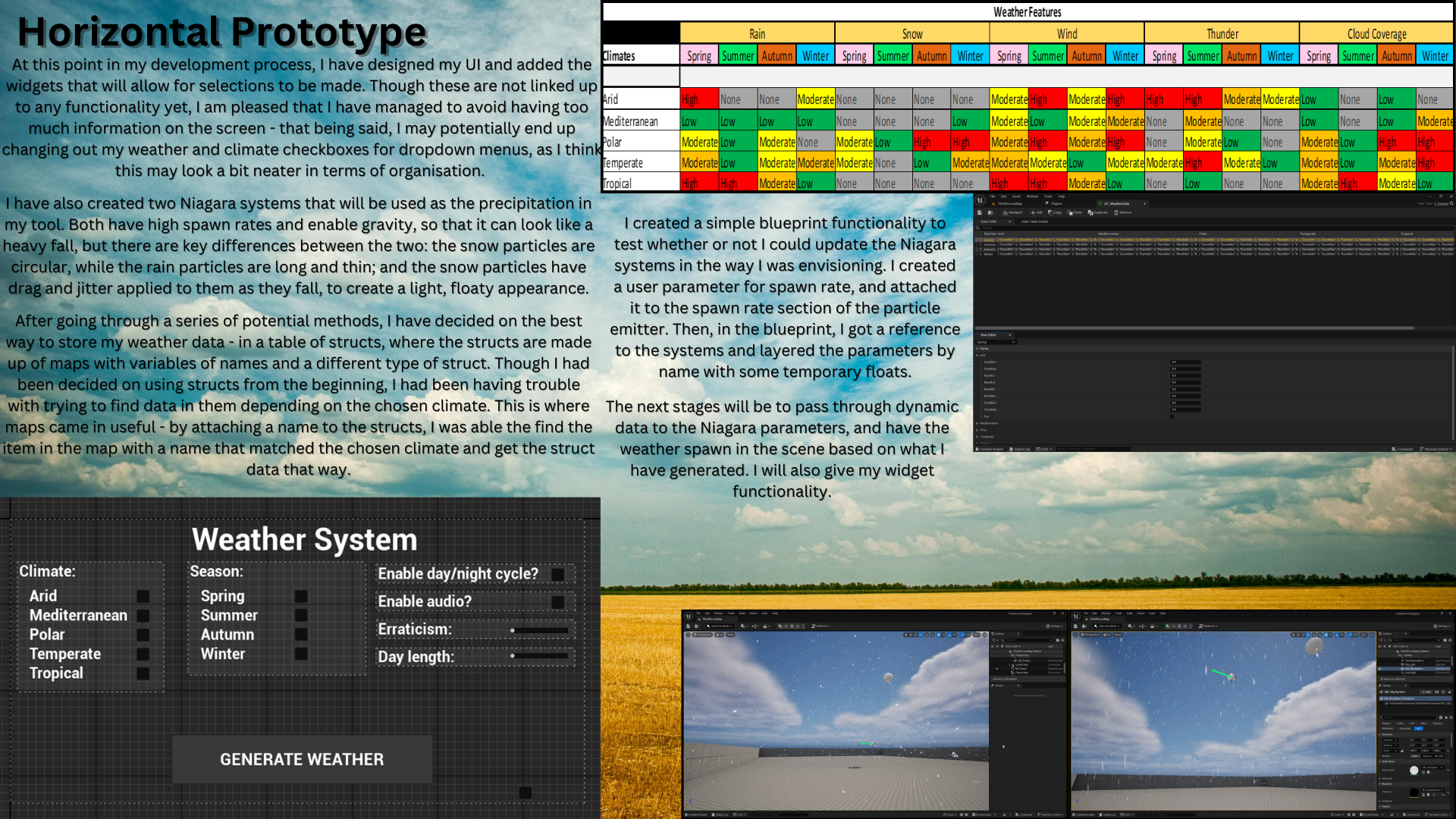
A diagram of a process

AI-generated content may be incorrect.

A diagram of a weather forecast

AI-generated content may be incorrect.

# Horizontal Prototype



# Polished System

The Weather Tool in its final stage for this development is a fully functioning system that generates weather based on the user’s desired climate and season. It is coded almost entirely in C++, with the only editor-made features being the data table (not including the structs that make up its data), the Niagara systems and the composition of the widgets – all technical functionality, as well as the weather volume, structs and the widgets themselves, were completely scripted.

**Editor Utility Widget**

The editor utility widget is the Weather Selector. Featuring dropdown menus, sliders and checkboxes, it is perfect for introducing users to the system, as it is clear, concise, and laid out in such a way that usage is carefree. The dropdown menus each have a clear button to get rid of an unwanted selection, and the sliders are spin boxes, which have a fixed range from which the user can choose a value, and have direct text entry as another option. At the bottom of the screen is a large button with the text “GENERATE WEATHER”. When clicked, the text will change to read “WEATHER GENERATED”, and it is at this point that the calculations for the weather are performed. Weather can be regenerated at any time by repressing the button, with either the same or a different selection of attributes to the previous set. **A screenshot of a weather login

AI-generated content may be incorrect.**

**Weather Calculations**

The program reads the data table items that correspond with the selection the user has made – for example, if the user were to choose Tropical and Spring, the program would find the row struct that matches Spring, and would then find the map within the returned struct that matches Tropical. This data is then used to calculate values for the weather that will be stored in a struct and added to a queue. This queue is then read by the Weather Volume one by one. The values of the data being used in the calculations is dependent on the intensity of that weather feature in the specified season and climate in the real-world.

**Weather Volumes**

The Weather Volumes are what make the weather visible in the world – the components being affected by the tool and calculations are part of the volumes. When the game begins, weather will be generated within the bounds of the volume, and the amount will be dependent on not only the calculations, but the size of the volume as well – this is because it would not look right if a 1x1x1 cube had the same amount of rain within it as a 100x100x100 cube. The volumes get their weather information from the EUW, and for it to be successfully passed through, the EUW has to find all the Weather Volumes in the scene and pass through the data as a struct. The WV then stores this data in its own structs and sets parameters and variables within its components to create the visible weather. The weather can only be created if there is at least one WV present in the level. Also within the WV is the handling of weather transitions. The duration of a weather state is dependent on a value that the user enters for erraticism, and is controlled by a timer. Since all the data is in a queue, once a weather state has reached its conclusion, the program will dequeue it – however, it gets stored in a temporary variable before being removed and is then added to the back of the queue. This allows the weather to cycle forever without it ever stopping because the queue has emptied. In the final part of a weather state’s active duration, any precipitation currently present will adjust to a midpoint of its own spawn rate value and that of the next weather state in the queue – this makes the switch between states appear less harsh.

**Day/Night Cycle**

The day/night cycle is also attached as a component to the Weather Volume. It is disabled by default, and must be enabled by the user in the EUW. It works by rotating the directional light of the sun and affecting the skybox as it does so (the directional light and skybox have to be selected by the user in the editor). The length of the days and nights are dependent on the value that the player chooses for the Day Length spin box in the EUW; the value corresponds to the rate of rotation of the sun, meaning that higher values result in a faster cycle, and smaller values cause it to change more slowly.

[](https://www.youtube.com/embed/MoAGmNJ8bLo?feature=oembed)

# Test Plan

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Number** | **Climate + Season** | **Test** | **Expected Behaviour** | **Actual Behaviour** | **Time** | **Success?** | **Fix** |
| 1 | Temperate, Winter | In weather types with both snow and rain possible, the two should not occur simultaneously | If rain is present, snow should not be present, and vice versa | No weather spawned | 14/02/2025 13:41 | No | Fix in code with new conditions based on its data table information |
| 2 | Temperate, Winter | Weather spawning | Weather should spawn based on moderate-high rain and snow, and moderate-low wind | Wind too high, unable to determine | 14/02/2025 13:53 | Indeterminate |  |
| 3 | Temperate, Autumn | Wind should generate with intensity based on data table | Low rain should be affected by low wind | Low rain, low wind | 14/02/2025 14:17 | Yes |  |
| 4 | Polar, Winter | Transitioning between weather states, including random no rain | It should stop raining rarely | Transitions, but very abruptly | 14/02/2025 14:22 | Somewhat |  |
| 5 | Tropical, Spring | Wind should generate with intensity based on data table | High rain should be affected by high wind | No wind change | 14/02/2025 14:34 | No | Setting wrong data in struct; changed variable in code |
| 6 | Tropical, Spring | Wind should generate with intensity based on data table | High rain should be affected by high wind | High, rain, high wind | 14/02/2025 14:35 | Yes |  |
| 7 | Temperate, Winter | Wind should generate with intensity based on data table | Moderate-high rain should be affected by moderate-low wind | Wind far too high | 14/02/2025 14:52 | No | Significantly decreased multiplier in wind calculations |
| 8 | Temperate, Winter | Wind should generate with intensity based on data table | Moderate-high rain should be affected by moderate-low wind | M-H rain, M-L wind | 14/02/2025 15:02 | Yes |  |
| 9 | Temperate, Winter | In weather types with both snow and rain possible, the two should not occur simultaneously | If rain is present, snow should not be present, and vice versa | Snow never spawns | 14/02/2025 15:06 | Somewhat | Rewrite snow calculations |
| 10 | Temperate, Winter | In weather types with both snow and rain possible, the two should not occur simultaneously | If rain is present, snow should not be present, and vice versa | Snow and rain both spawn | 14/02/2025 15:11 | No | Adjust bool checks |
| 11 | Temperate, Winter | In weather types with both snow and rain possible, the two should not occur simultaneously | If rain is present, snow should not be present, and vice versa | Snow and rain do not spawn simultaneously | 14/02/2025 17:17 | Yes |  |
| 12 | Tropical, Spring | Scale weather amount with volume size | Rain spawn rate should be higher for large volume and lower for small | Particle count maxed out | 14/02/2025 17:50 | No | Adjust scaling calculation |
| 13 | Tropical, Spring | Scale weather amount with volume size | Rain spawn rate should be higher for large volume and lower for small | Particle count maxed out | 14/02/2025 17:52 | No | Adjust scaling calculation |
| 14 | Tropical, Spring | Scale weather amount with volume size | Rain spawn rate should be higher for large volume and lower for small | Not enough particles spawning | 14/02/2025 17:54 | No | Adjust scaling calculation |
| 15 | Tropical, Spring | Scale weather amount with volume size | Rain spawn rate should be higher for large volume and lower for small | Working, but more adjustment needed. Needs to scale better with very large volumes | 14/02/2025 18:00 | Somewhat | Adjust scaling calculation |
| 16 | Tropical, Spring | Scale weather amount with volume size | Rain spawn rate should be higher for large volume and lower for small | Working, but more adjustment needed. Needs to scale better with very large volumes | 15/02/2025 18:53 | Somewhat | Adjust scaling calculation with cube root |
| 17 | Tropical, Spring | Softening of transitions between weather states | Rain should become closer to the next value before actually transitioning | Transitions more softly, but now it will never reach zero | 15/02/2025 12:32 | Somewhat | Debug while loop; it isn't actually looping |
| 18 | Tropical, Spring | Softening of transitions between weather states | Rain should become closer to the next value before actually transitioning | Transitions more softly, but now it will never reach zero | 15/02/2025 12:43 | Somewhat | Try using tick |
| 19 | Tropical, Spring | Softening of transitions between weather states | Rain should become closer to the next value before actually transitioning | Transitions more softly, but now it will never reach zero | 15/02/2025 13:00 | Somewhat | Make function for calling timer |
| 20 | Tropical, Spring | Softening of transitions between weather states | Rain should become closer to the next value before actually transitioning | Transitions more softly and can still reach zero | 15/02/2025 13:13 | Yes |  |
| 21 | Temperate, Winter | Erraticism factor should affect rate of transition | Higher erraticism values should have faster transitions, lower should have slower | Transitioning too fast for both high values and low values | 15/02/2025 13:48 | No | Wasn't setting erraticism in struct from user input |
| 22 | Temperate, Winter | Erraticism factor should affect rate of transition | Higher erraticism values should have faster transitions, lower should have slower | Lower values are transitioning faster, higher are slower | 15/02/2025 13:57 | No | Change timer calculations |
| 23 | Temperate, Winter | Erraticism factor should affect rate of transition | Higher erraticism values should have faster transitions, lower should have slower | High values have fast transitions and low have slow, but softening time is too long | 15/02/2025 14:03 | Somewhat | Change timer calculations |
| 24 | Temperate, Winter | Softening of transitions between weather states | Softening time should be proportional to erraticism factor | Inverse of softening rate is being applied | 15/02/2025 14:16 | No | Change timer calculations |
| 25 | Temperate, Winter | Softening of transitions between weather states | Softening time should be proportional to erraticism factor | Softening is being called too soon because it is within the transition timer | 15/02/2025 15:25 | No | Create a timer manager |
| 26 | Temperate, Winter | Softening of transitions between weather states | Softening time should be proportional to erraticism factor | Program failed to build because of recursion causing stack overflow | 15/02/2025 14:37 | No | Create a softening manager |
| 27 | Temperate, Winter | Softening of transitions between weather states | Softening time should be proportional to erraticism factor | Program failed to build because of recursion causing stack overflow | 15/02/2025 14:51 | No | Create a delay |
| 28 | Temperate, Winter | Softening of transitions between weather states | Softening time should be proportional to erraticism factor | Transitions, but correct softening is inconsistent | 15/02/2025 15:17 | Somewhat | Soften calculation is getting incorrect elements from queue |
| 29 | Temperate, Winter | Softening of transitions between weather states | Spawn rate should soften to midpoint of current and next weather | Transitions, but correct softening is inconsistent | 15/02/2025 15:32 | Somewhat | Incrementing head of queue |
| 30 | Temperate, Winter | Softening of transitions between weather states | Spawn rate should soften to midpoint of current and next weather | Softening to midpoint with rate proportional to erraticism | 15/02/2025 15:41 | Yes |  |
| 31 | Mediterranean, Winter | Weather spawning | Low rain | Extremely high rain | 15/02/2025 15:44 | No | Weather is scaling inversely |
| 32 | Tropical, Spring | Scale weather amount with volume size | Climate|Seasons with low rain should have low rain, and high should have high, rather than doing the opposite and scaling inversely | Low rain Climate|Season has low rain, high rain has high rain | 15/02/2025 16:50 | Yes |  |
| 33 | Tropical, Spring | Scale weather amount with volume size | Should have high rainfall no matter the size of the volume | High rainfall no matter the size of the volume | 15/02/2025 16:51 | Yes |  |
| 34 | Mediterranean, Winter | Scale weather amount with volume size | Should have low rainfall no matter the size of the volume | Low rainfall no matter the size of the volume | 15/02/2025 16:52 | Yes |  |
| 35 | Polar, Winter | Weather spawning | High snow spawn rate | High snow spawn rate | 15/02/2025 17:52 | Yes |  |
| 36 | Tropical, Spring | Wind changes should be more drastic | Wind should remain high, but change precipitation angles more drastically | Difference not very noticeable | 15/02/2025 17:55 | No | Adjust range of values in data table |
| 37 | Tropical, Spring | Wind changes should be more drastic | Wind should remain high, but change precipitation angles more drastically | Angle change is drastic | 15/02/2025 18:04 | Yes |  |
| 38 | Polar, Winter | Weather spawning | High snow and high, drastically changing wind | Spawn rate and wind angle being set to 0 | 15/02/2025 18:17 | No | Properly defined snow data in struct |
| 39 | Polar, Winter | Weather spawning | High snow and high, drastically changing wind | UE\_LOG is showing correct values but weather isn't changing | 15/02/2025 18:38 | No | User parameter wasn't set to change the gravity in Niagara system |
| 40 | Polar, Winter | Weather spawning | High snow and high, drastically changing wind | Values are definitely correct but are too small to be visible | 15/02/2025 18:46 | Somewhat | Change data in table |
| 41 | Polar, Winter | Weather spawning | High snow and high, drastically changing wind | High snow and changing wind | 15/02/2025 18:49 | Yes |  |

# System User Manual

Following installation of the tool to the project, there are a number of details that must be set up before the tool can operate. This manual will serve as a guide as to how to make the weather tool work.

1. **Place a weather volume** 
   * This is the most important part of the setup process. Without placing a weather volume, the tool will not work. Within the Unreal Editor, click the quick add button, represented by a cube and a green plus sign. In the dropdown menu that appears, scroll down to Volumes, or type “Weather Volume” into the search bar. Click to select it, and then drag it into the level. Resize it as needed using the typical UE5 scaling functionality. This process can be repeated for as many Weather Volumes are needed.
2. **Assign day/night variables**
   * Within the Weather Volume, scroll to the component named \_DayNightComponent. In this component are boxes labelled Sky Sphere and Light Source. Assign the values of these boxes to the directional light and sky sphere in the level.
   * *Note: to prevent complete darkness at night, the level will need a second directional light added with a low intensity (so that it doesn’t over-light the daytime). The suggested name for the second directional light is ‘Moon Light’.*
3. **Running the Editor Utility**
   * Once the setup is complete, the tool can be run. The UI that will be presented upon the running of the tool will have a number of areas for selections to be made.
     + To generate the weather, a climate and season must be selected
     + The value for the erraticism can be adjusted as needed to affect how quickly the Weather Volume will switch between weather states; use a high value for a high transition rate, or a low value for a slower rate
     + The Day/Night Cycle can be enabled or disabled by ticking or unticking the checkbox on the UI.
     + The value in the spin box for the day length will only matter if the Day/Night Cycle has been enabled. A high value will result in a shorter day, and a low value will make it last longer
   * When all options have been selected, the button at the bottom of the screen with the text “GENERATE WEATHER”. This marks the end of the setup process.
   * Weather can be regenerated at any time by running the utility again and following these same steps.
4. **Play**
   * The final step of this manual is to simply press play. By this point, the tool will have calculated everything it needs to, and the Weather Volume will have obtained all the information it needs.