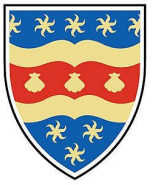

Plymouth University

School of Computing, Electronics and Mathematics

PRCO304

Final Stage Computing Project

2017/2018

BSc (Hons) Computing and Games Development

Samuel Lord

10485852

Unity Weather Package

Acknowledgements

I'd first like to thank Marius Varga for his support throughout the course of developing the package, as Project Supervisor.

Secondly, my fellow Computing and Game Development students for their testing and feedback.

Finally, Bethany Roberts-Rhodes for her patience and ability to spot my comma-lacking sentences.

Abstract

The present report describes a weather system package for the Unity Game Engine, intending to offer a robust, open-source alternative to those currently available.

An extensive context is provided, which describes the importance of weather in games and how procedural content can be beneficial. From this, a background to the project is detailed, in addition to the objectives and expected deliverables.

An objective breakdown of available software explains why Unity, C# in Visual Studio, Github, Axosoft and Sandcastle were selected as the software packages used to complete the project. This is followed by a discussion of the legal, social, ethical and professional considerations prior to, and throughout the project followed by the process of project management employed throughout the project.

The architecture of the system is then described, including an overview of code structure within the Unity game engine and a implemented control structures. A brief summary of testing by developers is used to show to what extent architecture-related objectives have been met.

A sprint-by-sprint project breakdown then details the process by which the project came to fruition. An End Project Report then evaluates the success of the project against the initial objectives, before a Project Post-Mortem critically evaluates the decisions made throughout the project.

A brief conclusion then proceeds the various miscellaneous documents in the appendix.

1. Table of Contents

Word Count: 10,969

2. Submission URLs

Where appropriate and possible, permissions for the following resources have been set to public. Where making resources public was not possible, the project supervisor, Marius Varga, has been sent an invite link or added as a collaborator.

Project Repository
Repository contents on OneDrive
Demo Repository
Demo Repository contents on OneDrive
Demo Build
V1.0 Package Release
Axosoft (Project Management)
Code Documentation

3. Introduction

Weather is a tool used throughout the game industry for atmosphere, story and realism. Powerful tools for producing immersive and engaging weather are often not available to independent and hobbyist developers, due to prohibitive cost or limited feature sets in the available packages. This project provided the opportunity to release a complete, open-source alternative for users of the Unity game engine.

The described project is a weather system built as a package for the Unity game engine in C#. It comprises an object control structure that drives MonoBehaviour-derived components in a generic way via temperature and intensity values. The system provides an extensible and queryable interface, which allows developers to implement only the visual elements and their specific weather controllers. The project has been released open source under the MIT licence. The present report describes the development process, inclusive of development tools, design, system architecture and problems and associated implemented solutions.

4. Project Context

The use of weather a literary device for story-telling and mood-setting is pervasive and has been used as a tool for hundreds, if not thousands, of years (Schulz, 2015). A simple example might be the marked difference between a cruise under a starlit sky and a boat rolling across waves under a troubled, rainswept sky. The tonal difference is clear and sets the scene for a sequence. The use of weather in fictional media is, in fact, so widespread that it is difficult to find examples of works where the weather in not used as a tool to reflect or enhance the content of a scene. This may be associated to the mood changes caused by weather in day-to-day life. Although the effects of weather, especially sunshine, on mood are often overstated, it has been shown that sunshine boosts positive moods and diminish negative moods (Cunningham, 1979). Cunningham found the effect is noticeable enough to cause a measurable change in behaviour. For example, daily stock returns have been shown to have better returns on sunny days (Hirshleifer and Shumway, 2001) and similar weather can cause increased returns for waiting staff in the form of larger tips (Cunningham, 1979).

It is therefore unsurprising that we see a reflection of these effects in fiction. One example in gothic literature, where weather is often used as a method to reflect the emotions of the characters (Epublications.marquette.edu, 2018), is in the first chapter of *Jane Eyre*, in which clouds are used as a reflection of Jane's "sombre" mood (Bronte and Davies, 2006). However use of this technique is truly ubiquitous in written fiction (Schulz, 2015).

Similarly, film has also implemented these techniques. As in the cinematic reimagining of *Let the Right One In (2008)*, in which snow is used a device to cause disquiet through the juxtaposition of the soft, glistening flakes falling in the foreground and the still, sober progression of the opening scene (Nicholls, 2013). Not only does weather provide a parallel to characters' inner thoughts but it also presents an often striking backdrop to fictional media, adding a layer of context to otherwise bland interactions. Further, in visual mediums, weather presents a palette of colours to contrast or compliment those of the presented scene. A salient example of this is in the film adaptation of *Lord of the Rings: The Two Towers*. When Gandalf arrives at the Battle of Helm's Deep, leading Éomer and his cavalry, the sun breaks behind them as they make their charge. A clear contrast to the hoards of black-cladded Uruk-hai in the valley beneath them (The Lord of the Rings: The Two Towers, 2002).

Games, as a creative medium, are no strangers to the use of weather as pathetic fallacy for events in their interactive environments. It has been shown that rich weather systems in virtual environments increase the immersion and realism experienced by players when compared to their static weather counterparts (Roberts and Patterson, 2017). Realism has long been a target for the games industry, with the end goal of virtual environments appearing indistinguishable from the real world (Roberts and Patterson, 2017). From as early as 1962, when Steve Russell's *Spacewar!* (1962) included a backdrop with all the stars as seen from earth down to magnitude 5 (Markowitz, 1999), to the rich and diverse worlds seen in *Grand Theft Auto V*, *Far Cry 5* and *Kingdom Come: Deliverance*. This desire has been reflected in science fiction in various forms of the well know "Holodeck" from *Star Trek*: The ultimate goal of game development (Murray, 1997).

It makes perfect sense that weather would be included in the ever-increasing realism provided by games. In fact, it has been found that including realistic weather effects in games may offer more than just graphical realism in that the combination of "unpredictability" and "compelling simulation" may make the game inherently more "real" as an experience (Barton, 2008), due to the similarities to everyday life.

Murray (1997) stated, "The more pervasive the sensory representation of the digital space, the more we feel we are present in the virtual world". We see this in the more recent explorations of environments in increasingly impressive virtual reality headsets, and the matching increase in believability and 'realness' in the experiences provided by this medium. As more experiences and games are created for more immersive mediums, the requirements for the believability of the environments players enter are only going to become more demanding, and this includes ambient effects such as the weather.

Hand-crafting weather environments to be consistently realistic may be too time-consuming or complex to create for all environments in a game. In these instances, a dynamic weather system may be driven by procedural generation to handle the generation and changes of

the weather at runtime, with gradual, realistic changes in weather in a deterministic way. This presents the illusion of a game world as a living and dynamic virtual space (Barton 2008). Procedural generation allows for a changeable and active system, driving hand-crafted visuals and, optionally, mechanical changes to the game, based on the current weather conditions. Procedural generation has been previously shown to be a powerful tool in creating dynamic game worlds. One of the most popular games of the last decade, *Minecraft*, is often cited as an example when discussing procedural generation due to its generation of worlds that can reach sizes up to that of 8 times the area of Earth (Minecraft Wiki, n.d.). However, procedurally generated content does not preclude failure. In the case of *No Man's Sky*, a explorative space game in which 18 quintillion planets are procedurally filled with flora and fauna, received very mixed reviews upon release (Hudson, 2018), highlighting that dynamic visuals are no replacement for engaging mechanics. This can also be seen in other games that use weather as not only a visual backdrop, but also an integral part of gameplay. *Playerunknown's Battlegrounds*, utilised weather in its battle-royale gameplay in the form of rainy and foggy matches. Rain acted as a silencer to player actions, such as running and firing weapons. This changed the tactics required in those matches (Kengaskhan, 2018), adding further depth to gameplay. Similarly, foggy matches created situations in which players would experience more close-combat gameplay due to the limited view distance (Porreca, 2017; Kengaskhan, 2018). When *Playerunknown's Battlegrounds* removed weather in an update in September 2017, there was widespread criticism online, including on the game's own forum (PLAYERUNKNOWN'S BATTLEGROUNDS Forums, 2018), further illustrating the interesting gameplay weather can create for players. An update due to be released shortly will be re-adding weather, this time with dynamic, rather than fixed, weather effects.

Weather effects also impact mechanics in single player games. For example, *Frostpunk*, *NCAA Football 2008, Dwarf Fortress* and *Sea of Thieves* all use weather to impact gameplay in mechanics as well as visuals. *Frostpunk* is a city builder which uses weather as both its main plot and backdrop as well as for mechanical effect in gameplay. The game takes place during the summer of 1886 in an alternate-reality, steampunk Britain where the player must build 'New London' in a post-apocalyptic icy tundra. Mechanically, weather affects gameplay in "deep freeze" events, in which keeping areas at a survivable temperature becomes a priority, shifting a players focus to increasing generator reach and keeping it fed with coal, at the cost of other targets (Bertz, 2018).

NCAA Football 2008 utilised live weather data for its American football gameplay, using the real-world locations of its virtual stadia to accurately depict the weather (Thomas, 2007). Inclement weather impacts gameplay by affecting the speed at which players can move, whereas extreme heat negatively impacts stamina. However, weather is not dynamic over the course of each game, only updated when a new game is created. *Dwarf fortress* is a fantasy game in which the player can command a group of dwarves or adventure in a randomly generated, persistent world (Bay 12 Games, n.d.). The game implements an

extensive dynamic weather system which tracks wind, humidity and air masses to simulate a realistic weather system. The system includes both real and fantasy weather, including rain, snow, evil clouds and evil rain. Weather is an extensive component of *Dwarf fortress* with various mechanics affected by different weather types. A dwarf that is caught in the rain, for example, has an "unhappy thought" sentiment applied to it, which can cause physical costs to dwarves, potentially to lethal effect. Rain also impacts the game world, filling "murky pools" as precipitation occurs allowing for player-made wells to provide dwarves with a water source. Snow and ice also has impacts on the game-world. When water enters a freezing climate ice is formed, which can kill any creatures caught inside of it. Furthermore, dwarves caught in snowstorms can freeze to death if the player does not dig out a warm place for dwarves to wait out storms. The fantasy weather events add further mechanics: Evil rain, in addition to the "unhappy thought" trait, may also apply fevers and vomiting to dwarves alongside the chance to cause wounds and injuries to creatures and dwarves caught in the downpour, although the specific effects depends on the liquid that forms the rain. Evil clouds cause more serious symptoms that evil rain, in certain cases creating zombies from creatures trapped inside which are more dangerous than their living counterparts (The Dwarf Fortress Wiki, n.d.).

Sea of Thieves, Rare Studio's recent multiplayer pirate game, includes roaming storms in their game worlds. The unlucky ship caught in a storm, whose crew did not notice the approaching dark clouds and rising waves, must be constantly repaired and water bailed from under the deck. These storms were added to exaggerated the vulnerability and exposure felt by players in an exhilarating way (Rare, 2017). This is achieved not only by the damage sustained whilst in storms, but also by the currents of the storm dragging the ship in different directions. This means that players have to actively keep the ship heading in their desired direction. The addition of storms to *Sea of Thieves* adds enjoyable, emergent gameplay and as such, storms have been described as being as thrilling as the missions (Frushtick, 2018).

With the realism and immersion of games being benefited by weather systems, but those systems often being a secondary consideration (Roberts and Patterson, 2017) and taking up valuable time and resources to produce, it may be the case that smaller development studios, independent developers and hobbyists neglect them in favour of other game systems deemed to be more integral to the release of their games. Furthermore, it may be beneficial to those parties to utilise pre-built solutions for their needs, hence the inception of this project.

5. Background, Objectives and Deliverables

5.1. Background

As discussed, the use of weather in games is ubiquitous, albeit with varying levels of impact in both gameplay mechanics and visuals. As such, packages for game engines such as Unity, which provide import-and-go weather solutions, are available. Due to the potential complexity of weather systems, which often include seasonal changes and day-night cycles, the quality and accompanying price tag of available packages varies broadly. Therefore, the more complete weather packages can be prohibitively expensive for hobbyist and independent developers and the equivalent free packages are either closed-source at the point of download and may, understandably, lack the more advanced features provided by the more expensive counterparts. To ascertain the current state of available weather systems, a comparison of Unity weather packages was conducted as part of the project initiation process (Table 1). The analysis highlighted the limitations of cheaper packages; no queryability, lack of audio support and limited terrain interaction.

Table 1 - Comparison of available weather packages

Product Name	Price	Types of Weather	Extensible	Terrain Interaction	Audio support	Example code/ scenes/ assets	Editor-friendly	Queryable
Weather Maker	$42	Fog, rain, snow, hail and sleet	Yes	Colliders and shaders	Yes	Yes	Yes	No
Enviro	$50	Clear Sky, cloudy, raining, stormy, snowy and foggy	Yes	Water only	Yes	No	Yes	No
UniStorm 2.4	$60	Sunny, Mostly Clear, Partly Cloudy, Mostly Cloudy, Foggy, Snow, Rain, Lighting and Thunderstorms	Yes	Shaders only	Yes	Yes	Yes	Yes
Time of Day & Weather System	Free	Sun, cloudy, rain thunderstorm and snow	Yes	No	No	Yes	Largely	No

With the current state of weather solutions being as it is, this project aimed to produce an open-source alternative, offering the oft-missed features of cheaper solutions. The open

source nature of this project also means that community-driven pull requests with fixes and new features is a possibility, resulting in a more robust and feature-rich solution over time (Schindler, 2007).

5.2. Objectives

As described in the project initiation document, this project aimed to produce a Unity package that delivered on the following requirements (Appendix D.1):

1. Be free, open-source and reusable without limitation
2. Be procedural
    a. Will provide a platform from which weather patterns can be new and interesting on each playthrough of any game using the system.
    b. Will be seeded and therefore will generate reproducible weather patterns at the discretion of the developer, and enable easier testing.
    c. Weather will vary across the game world at any given moment, adding more realism. This is in contrast to the researched products which unanimously had ubiquitous weather throughout the game world.
3. Be extensible
    a. Will be designed so as to be extensible by developers using the platform.
    b. Will be well documented so as to be easy to use for other developers.
4. Be queryable
    a. Will allow developers to request information about the weather at specific locations and times. This allow for games which use information such as temperature to, for example, change the look of the player or environment in different conditions or provide information to survival-like games.
5. Be Unity Editor friendly
    a. Changes to initialization values will not require editing code and will be available as sliders/editable text boxes in the Editor.
    b. As and when required, custom editors may be written to automate repetitive tasks.

5.3. Deliverables

The project will culminate in the delivery of a weather system package for the Unity game engine. The package will include a solution to the described objectives and an example Unity scene with a sample setup. The project code will be open-sourced at the culmination of the project.

6. Software

6.1. Engine

There are a wide variety of game engines available for use and several were considered for this project. Engines not able to build for Windows and OSX were immediately discounted as possible choices, as these were the target platforms for the weather system. Unity was an obvious choice due to the extent to which its use has been covered during the course. However, Unreal Engine, CryEngine and Unity Engine were all considered due to their 3D support and free or royalty-based licences.

Unreal Engine was a promising option having been awarded the “most successful video game” by Guinness World Records. Also, having removed the proprietary UnrealScript scripting language in favour of C++ for version 4 of the engine, there was less of a learning curve than in prior version due to the author’s experience with C++. Unreal also offers visual scripting with “Blueprints”, although lack of experience with this feature would likely result in a steep learning curve. Finally, Unreal’s focus on photorealism was considered, however the objectives of this project on the system structure over visuals meant this wasn’t a deciding factor.

CryEngine also advertises “breathtaking visuals” which, again, although a potential positive for users of the final system was not a direct benefit to the completion of the project. CryEngine offers C++, C# and Lua bindings to its engine API, all three of which the author has varying levels of experience with, offering a lesser learning curve to that of Unreal. However, CryEngine requires custom formats to import textures and objects, requiring extensions on third party applications to be able to create game assets. This barrier to entry would limit the work possible in University Laboratories and therefore the productivity over the course of the project.

Unity was ultimately selected as the game engine for this project for two main reasons. Firstly, the author has a deep understanding of Unity and C#. With the limited time for the project, the time overheads of gaining experience in a new engine would be detrimental to the goals of the project. Secondly, Unity has a large independent and hobby developer community who may benefit from the resultant software package. The component-based nature of Unity may also offer benefits for developers who use the package in that custom components can be interfaced with the system structure purely with the drag-and-drop functionality in the editor. Finally, Unity supports direct import of standard asset formats and has a wide variety of third-party assets available to showcase the project.

6.2. IDE

Visual studio was a natural choice as an IDE alongside Unity development as Unity comes packaged with Visual Studio Community Edition. Additionally, Visual Studio's debugging features integrate with Unity for more advanced debugging, for example breakpointing. Alongside this, Visual Studio is a well supported IDE and endorsed by Microsoft, the developer of C#. Alternatives to Visual Studio were considered such a MonoDevelop and Visual Studio Code however the debugging was lacking (non-existent in the case of Visual Studio Code) in comparison to Visual Studio. This is shown Unity's recent termination of MonoDevelop support in favour of Visual Studio.

6.3. Git

GitHub was selected as the host for the project code repository due to its status as the industry standard (Gousios et al., 2014). GitHub also provides a GUI git tool, GitHub Desktop, for easier management of the repository. Furthermore, laboratory computers on the university premises already had GitHub Desktop installed, therefore allowing for project work in university with greater ease than would be the case with other considered solutions, such as Bitbucket and its tool Sourcetree.

6.4. Project Management

There are numerous options for project management software, many tailored to specific project management methodologies. Axosoft was selected as the project management tool due to its tight integration with the management method of the project; Agile/Scrum. The author was also already familiar with the workflow of this tool from previous projects and so Axosoft was the natural choice. Also, by running in the browser (alongside recent mobile support), Axosoft allowed ease-of-use independent of working location. Trello was also

considered as a more lightweight alternative, however the lack of time estimation and SCRUM support made it a significant inferior project.

6.5. Documentation Generation

There are limited options available when generating documentation from XML-style comments in Visual Studio. This is arguably because the most common tool, Sandcastle, fulfills all the needs of developers.
Sandcastle is made up of two parts, *Sandcastle tools* and the *Sandcastle Help File Builder*. Sandcastle tools are command-line tools for generating help files by combining the XML comments in the source code with reflection on the built software or class library. The Sandcastle Help File Builder wraps all these tools into an easy-to-use GUI, which allows project settings to be saved for later regenerating new documentation with the same settings. Sandcastle supports several output formats, most notably as a set of HTML files which can be made accessible on the web. This has obvious benefits for this project where developers may need to access code documentation and that documentation may need to be periodically updated. For this reason, Sandcastle was selected as the documentation generator for this project.

7. Legal, Social, Ethical and Professional Issues

7.1. Legal

The primary legal concerns in this project relate to the open-source nature of the codebase, and the accompanying licensing. The MIT licence (Appendix H.1)(Opensource.org, n.d.) was selected to release the assets created within the scope of this project. This is due to the fact that the MIT licence is permissive, providing very little restriction on the use and redistribution of the produced code. It also frees the producer of code and assets under the licence from liability and explicitly removes any warranty on produced elements. There are no trademarks related to this project and as such, no consideration has been mode for trade marks in open source works.

Second, the licences of third party assets used in the project were closely adhered to, in order to not break the terms of the licence. This included maintaining the licence for the third party assets in the repository and showing accreditation where required. For example, The Volumetric Lighting code was distributed under the MIT Licence and as such required the licence to be displayed with that code in the project repository. The licence was included as a header in each script to comply with those terms.

Furthermore, the licences of the software used in producing the project are also of note, as some software licences limit the manner in which elements produced thereof may be used. This influenced the choice of software used. For example, Unity explicitly grants developers rights to the content they create using its engine: "*you fully own the content you create with a Unity subscription, also if you stop subscribing to Unity*".

Finally, at the conclusion of this project, the package will be submitted to the *Unity Asset Store* and so there was also consideration given to the licences granted to developers downloading the package from this source, given the project's approval to the store. This is of little consequence however, due to the liberal licence under which the source code is released on Github.

7.2. Social

The software produced as a result of this project is unlikely to have any major social impact. There is a social element to be consider, however, in the form of the issue tracking within the repository. There is an onus on the owner of a repository, at least to some extent, to moderate the content of the issue tracker. However, GitHub does not support deleting issues in an issue tracker and so the moderation that could be performed would be limited.

7.3. Ethical

There are limited ethical considerations for this project. In the project initiation document (Appendix B.1.2), it was stated that this project would conform to the approved PRCO304 ethics application. This remains true. Usability testing on the system was conducted on the final system by final stage students, in order to determine that the project had met its objectives. The submissions were anonymous and conformed to the requirements as laid out in the ethics application.

The only other ethical concern to consider is that with the software being released under an open source licence, it is likely that the software will be used by other developers in the future. As such, there may be a moral obligation on the part of the publishing party to ensure the software works and is usable or to provide support (Gotterbarn, n.d.).

7.4. Professional

Professional considerations throughout the project used the BCS, The Chartered Institute for IT's Code of Conduct (Bcs.org, n.d.).

8. Project Management and Method of Approach

As discussed, Axosoft was chosen as the project management tool for this project due to its strong conformance to the Scrum agile methodology. However, Scrum was not the first methodology considered for this project. Initially, the Cowboy methodology was considered, due to its focus on solo development and iterative development. Cowboy, however, also has a strong emphasis on client interaction, and although the project supervisor could have fulfilled those requirements, it was deemed a better fit for the project to utilise a more robust method in Scrum.

Due to the existing structure of the project, sprints were initially set to one week intervals, starting on a Friday. This allowed the project to neatly align with the weekly highlight reports that were produced. At the beginning of each sprint, work items were created in the backlog to be completed by the end of the sprint, and items not completed in the last sprint were rolled over into the new sprint. New items, rather than being given points as is usual in Agile, were given time estimates in much the same manner (Cheng, 2012). This was useful when used alongside Axosoft's item allocation feature in which time estimates were used to calculate if the work items in the current sprint could be completed in the remaining available time. Items were also given a classification - task, bug or user story. This allowed for easier management of the sprint and prioritisation of the work to be done.

It is usual that alongside agile methodology in software development, source control is used with a branching strategy. In this instance, git was used as the source control, however no strict branching strategy was imposed. This was driven by there only being a single developer working on the project, and as such only one feature or bug could be worked on at a time. As such, at the project's inception, a feature-based branching strategy was deemed to be unbeneficial to the project.

Coding practices were maintained with a "Best Practices" document (Appendix G.1), which defined the way in which code should be written to conform to a coherent style across the code base. Furthermore, documentation was ensured to be pervasive throughout the code, with all methods and classes having XML documentation. This allows for documentation generation to be automated. Further in-line comments were made around unclear or obtuse code sections.

9. Architecture

Working within a game engine provides a base structure to build from without repeating large amounts of boilerplate code for every project. Equally, there are some limitations

imposed by utilising any third-party code or platform, game engines included. In Unity's case this means conforming to a component-based model.

In software engineering, components are an encapsulation of a set of functions as a module, resource or package. Each system process is its own component with all of the data and functions within a component being semantically linked, in much the same way as classes in traditional object oriented programming. This allows for a modular system in which components can be reused in different contexts. For example, in Unity, a GameObject (the base class for all scene objects) for an enemy may have a "Health" component which allows for damage to be taken from player attacks and destroys the object when the health is less than zero. Equally, the same component could be used for destructible objects or the player itself, providing the same functionality.

In Unity, all components must inherit from MonoBehaviour somewhere in their parent structure. MonoBehaviour provides a number of in built methods that are called in a specific order at runtime (Docs.unity3d.com, 2018). For example, *Awake()* is called as soon as the script instance is being loaded, *Update()* is called every game tick and *OnDestroy()* is called when an object is destroyed. Also, MonoBehaviours are kept track of by Unity, allowing for MonoBehaviours to find other components at runtime.
Unity also provides an alternate parent class, ScriptableObject, for objects that do not need to be attached to GameObjects, and can be treated as assets. These objects are persistent between game instances and offer a powerful interface for settings or uses such as inventory management, or any data container. They also interface well with Unity's serialization system. ScriptableObjects lack many of the methods provided by MonoBehaviour as they are not designed to be updated at runtime.
Both Monobehaviours and ScriptableObjects have been utilised in the creation of the weather system, and the design has been shaped by the strengths and limitations of these predefined structures.

The architectural design of the weather system was driven by the core objectives, especially the extensibility, query-ability and editor-friendliness of the developed system. The editor-friendliness of the system was achieved with custom editors for the produced behaviours. Perhaps counter-intuitively, the Unity editor is largely written in the Unity game engine, using the same APIs exposed to game developers. This means that the editor is entirely scriptable meaning custom editor windows and component editors can be written to simplify or automate tasks. In the case of the present project, several editor scripts were written. Firstly, the enum for the available types of weather has its own editor in Unity, which does metaprogramming to allow developers to easily add, remove or edit weather types without having to modify internal elements of the final package. Also, custom or generic data types (except for List<T>), such as the DoubleDictionary created for this project, are not serializable by Unity. In instances where they are required, custom editors must be produced to correctly display and edit those objects. The DoubleDictionary<T,U,V>

object created to lookup values using two keys was not serializable by Unity due to the fact that it supported generic types and used a dictionary of dictionaries as its underlying data structure. In this instance the WeatherLookup object, which is internally a DoubleDictionary object, required a custom editor for selecting values for each key pair. Further, due to the serialization limitation, the DoubleDictionary has to be converted into List<T> objects on serialization and then, on deserialization, unpacked back into a DoubleDictionary.

The extensibility of the system was important to ensure that developers using the system could further build out specialised changes that are pertinent to the game they are creating. This was achieved with a hierarchical structure in which data could be passed down, being mutated as appropriate at each level. At its core, the weather system uses an intensity value to drive each weather element, eg. precipitation, wind etc. In the resultant implementation, a data object is passed down the hierarchy , where intensity values are modified at each level by curves (Figure 1). Eventually, the data is passed to MonoBehaviour-derived components which control the visual elements related to each weather element, using the intensity value and associated data, i.e temperature and humidity. However these values could be used for any purpose a game developer saw fit, and would only require inheriting

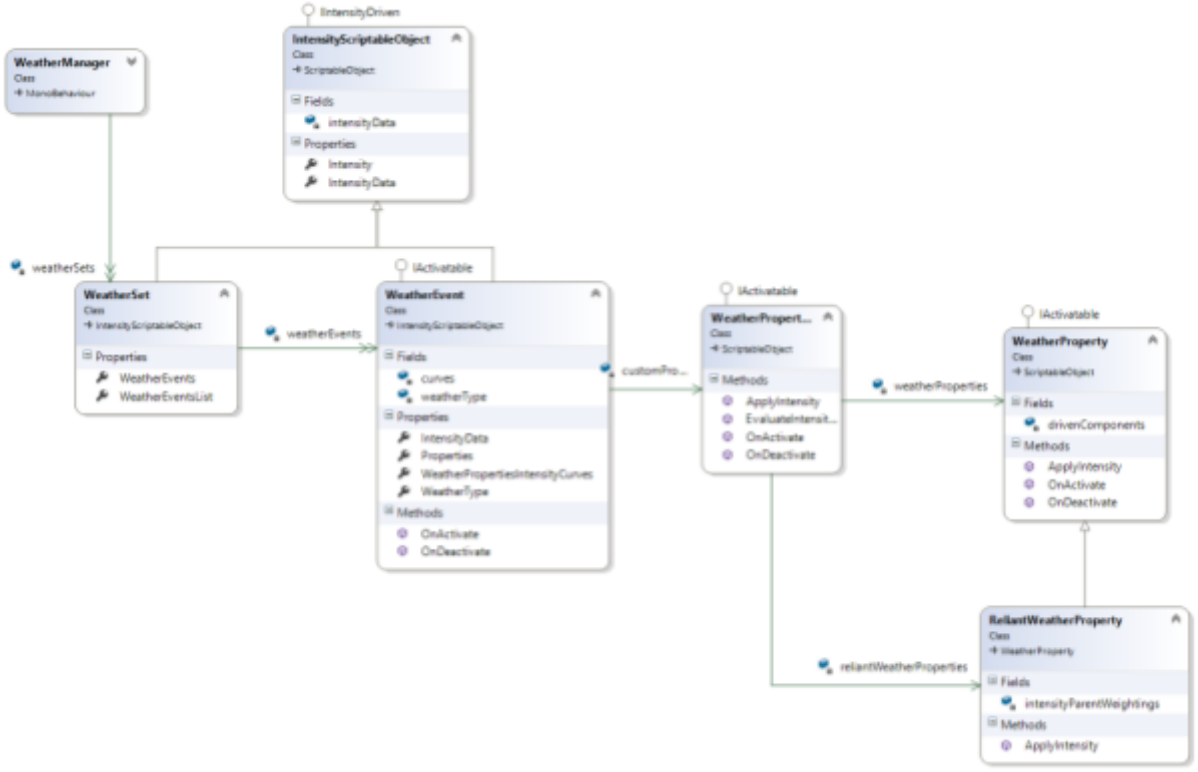

Figure 1 - UML diagram of the intensity flow structure in the weather system

from an IntensityDrivenBehaviour. Several example derived controllers for specific use cases such as shaders, particle effects and audio have all been included as example implementations with the package. Furthermore, due to the structure deriving from a central WeatherManager object, queries for absolute data (temperature, humidity and

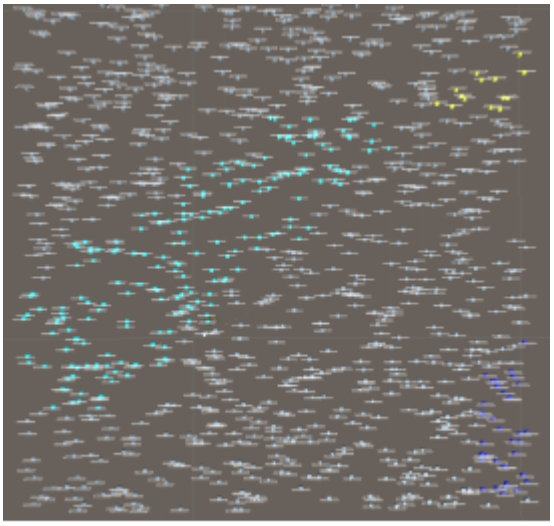

Figure 2 - The querying of weather type by 1000 objects at various locations in the game world. Grey denotes overcast, turquoise hail, yellow clear and blue rain.

intensity) for a specific location can be made directly to that object (Figure 2). This allows extension of the system, especially in procedural mode, to drive any weather-affected element of the game world. Queryability was also maintained at the lowest level of the procedural weather. This was achieved by keeping the weather generation as simple as possible whilst also maintaining the believability of the produced weather. Primarily, this resulted in a single, static class for Perlin-based generation, which contained arbitrary offsets for the noise generation for each weather element - temperature, humidity, intensity, and wind. Unity's *Mathf* library already included a Perlin noise implementation and this was deemed suitable for the project's objectives. However, to meet the objective to be seeded, to allow for easier testing and repeatable weather patterns, some additional boilerplate code was required. This involved the addition of a fixed value alongside the calculated position, in the call to the perlin noise method. Additionally, to ensure that each weather element sampled the noise at a different position (and therefore did not consistently match), the aforementioned fixed offsets were used as a modifier to the queried position. Finally, the scale at which the noise was queried was made modifiable from WeatherManager, allowing for a realistically scaled weather 'fronts' irrespective of gameworld size or scale.

Due to the limitation of ScriptableObjects not having an *Update()* call to dynamically change at runtime, the structure of intensity flow is not fixed at compile time. Rather, IntensityDrivenBehaviours are given a reference to their parent WeatherProperty or ReliantWeatherProperty (both ScriptableObjects), from which they wish to be driven and notify the parent at runtime that they wish to be updated. Although this extra step seems somewhat convoluted it allows for WeatherProperty and WeatherProperties objects to be treated as assets in the editor, which in turn allows developers to define the system behaviour in the editor, rather than having to define behaviour in specialised scripts. Further, developers can inherit from IntensityDrivenBehaviour to define their own custom controllers for a new weather property just by adding that property to a weather event in the editor or, alternatively, can define a new IntensityDrivenBehaviour of a existing WeatherProperty by adding a reference to that weather element to their script.

In order to determine to what extent the system had met its objectives of editor-friendliness, extensibility and queryability, a questionnaire was completed by final year students and an average rating was calculated (Appendix F.1). Each section, scored 70% in the ratings overall.

10. Stages

Each stage of the project was treated as a sprint in terms of the SCRUM management of the project. Furthermore, each sprint was concluded with a "Sprint Assessment" in order to better keep track of the completed objectives on a sprint-by-sprint basis, in addition to the management through Axosoft (Appendix C.1.1 - C.1.11).

10.1. Sprint 1 - ending 06/02/18

During the first sprint of the project a requirements document was produced (Appendix D.1) using the MoSCoW method. This summarised the project Epics that made up the minimum viable product; customisable weather assets, believable weather, extensibility, queryable weather and accessibility. It also clarified what the project would not deliver; photorealistic assets and simulation-level realism. Once the core requirements of the project had been documented, project management was initalised with a new Github Repository and Axosoft instance.

A prototype of the weather system was also produced, which demonstrated particle effects being driven by a procedural weather algorithm, in which the weather changed over time. This included a simple particle management system for displaying the correct particle system for each weather type. This prototype was driven by perlin noise (Scratchapixel, n.d.) in the Unity Game Engine and two values were produced representing the humidity and temperature for a given two-dimensional coordinate position in the game world. The two coordinate values were used to calculate a weather type. For example, a high temperature and high humidity might result in stormy weather whereas mid temperatures and mid humidity might produce sleet.

Real weather data collected from passive logging over the last year was compiled into a single document to be used as reference for realistic weather transitions (Table 2).

Table 2 - Sprint 1 work summary

Goals	**Time Allocated (hrs)**	**Time Used (hrs)**	**Complete (Y/N)**
Initialise project repo with Unity project	2	0.5	Y
Setup Axosoft	2	1	Y
Produce Requirements Document	2	1.5	Y
Throwaway prototype of procedural weather	16	15	Y

Compile real weather data in single document for later testing	2	2	Y

10.2. Sprint 2 - ending 14/02/18

A new repository was created with a custom gitignore, including a simple readme and licence. This was to ensure that prototype code was not included in the final version.A UML was constructed in Visual Studio from which all the base classes were generated. The classes were then edited to correct any inheritance errors, eg WeatherSet and WeatherEvent were changed to inherit from ScriptableObject.

Custom editors were produced for dynamically adding WeatherTypes enums. Similarly a custom editor was made for WeatherManager which allows for dynamically switching between procedural and manual modes in the editor. An editor for a DoubleDictionary data structure was created which requires two keys, rather than the usual one to lookup a value. This is the underlying data structure for ProceduralWeatherLookup. Further, a ProceduralWeatherLookup editor was produced, however serialization issues at this stage resulted in the underlying ScriptableObjects from storing the data.The scripts and assets created at this time were organised in Unity to make management easier in the longer term. A new namespace was introduced, 'WeatherSystem.Internal', for the inner workings of the system as to not expose, unnecessarily, too much information to the end user.

The user tests and visual implementation research that were scheduled for this sprint were delayed to the next sprint in order to focus on bug fixes (Table 3).

Table 3 - Sprint 2 work summary

Goals	**Time Allocated (hrs)**	**Time Used (hrs)**	**Complete (Y/N)**
New project repository for actual implementation	0.5	0.5	Y
Create base class implementations based on prototype	8	4	Y
Refresh editor code knowledge through implementation of first editors	32	28	Y
Create object for lookup using two values -	8	4	Y

“DoubleDictionary”			
Design user tests	8	0	N
Research common implementations of weather effects (eg shaders vs particle effects etc)	8	0	N

10.3. Sprint 3 - ending 22/02/18

The serialization issues experienced in the custom ProceduralWeatherLookup editor were fixed in this stage. Similarly, bugs in the WeatherTypes editor were fixed. Firstly, a null reference would be caused when editor code was re-compiled and secondly no edits would be possible until the array was resized. Both bugs were solved.

Noise generation functions were implemented based on the work conducted for the prototype, including seeding to allow for repeatable procedural weather. Further, basic noise-based wind represented by a Vector2 object was implemented and used to drive the sample location of the weather perlin noise map. This created a more believable changeable weather system for testing. Further, extra scripts were produced to visualise the noise generation and enable faster testing and iterations. Finally, preliminary user test designs and weather system visualisation research were produced. (Table 4).

Table 4 - Sprint 3 work summary

Goals	**Time Allocated (hrs)**	**Time Used (hrs)**	**Complete (Y/N)**
Design user tests	8	1	Y
Research common implementations of weather effects (eg shaders vs particle effects etc)	8	4	Y
Fix serialization issues in WeatherLookup editor	8	8	Y
Implement procedural noise generation with seeded values	16	16	Y
Develop tools to	8	8	Y

test/visualise procedural noise			

10.4. Sprint 4 - ending 01/03/18

In this stage, research was conducted into Unity's AnimationCurve system and, using information gathered from the research, intensity transitions had curves added for the modification of intensity values during weather changes. As part of testing the new curves system, new weather events were added and updated. Also, weather transitions were fixed, stopping an issue where weather would constantly loop between two weather types. In addition, a preliminary implementation of WeatherManager weather queries was added to allow external objects to get weather at their positions from which a new weather visualisation was created in the form of 'Weather Stations'. These display the type of weather at their position and update periodically. Further to this, an accompanying distributor, which spawns a number of stations was created. (Table 5).

Table 5 - Sprint 4 work summary

Goals	**Time Allocated (hrs)**	**Time Used (hrs)**	**Complete (Y/N)**
Research Unity curve system	4	4	Y
Implement WeatherEvent hierarchy	32	32	Y
Implement Weather transitions with lerping intensity	4	4	Y
Develop "weather stations" to display weather at various location	2	1.5	Y

10.5. Sprint 5 - ending 08/03/18

A major refactor occured in this stage. This introduced weather properties and reliant weather properties. This allows for properties to be independently applied to each WeatherEvent, and for a hierarchical structure of WeatherEvents. Due to the new structure, a new WeatherEvent editor for adding curves through to WeatherProperties objects at

runtime was implemented. Further to this, WeatherProperty was overhauled to use MonoBehaviours inheriting from IntensityDrivenBehaviour which are found at runtime and makes the system much more flexible.
In preparation for adding visual elements, a first person controller and volumetric lighting package were imported. Alongside this Terrain was added from a previous project as a placeholder. Also, the first visualisation was created in this stage: a rain particle effect.
The WeatherStation introduced in the prior stage was extended to distribute weather stations on the terrain (Table 6).
Table 6 - Sprint 5 work summary

Goals	Time Allocated (hrs)	Time Used (hrs)	Complete (Y/N)
Implement weather properties as a child in the Weatherevent hierarchy that are also driven by intensity values	8	15	Y
Test potential demo controllers and scenery	4	8	Y
Create a particle effect to be later driven by a weather property	4	3.5	Y
Add WeatherEvent editor for curve application via weather properties	4	2	Y

10.6. Sprint 6 - ending 15/03/18

In this stage a focus was placed on the 'Manual' weather mode in which WeatherEvents are queued to occur. Substantial progress was made in this regard.
New IntensityDrivenComponents were added to the project for controlling materials, particle effects and audio. Thought was also given to how to get temperature and humidity values from a WeatherEvent at a particular position in the 'Manual' weather mode (Table 7).
Table 7 - Sprint 6 work summary

Goals	Time Allocated (hrs)	Time Used (hrs)	Complete (Y/N)
Manual weather	16	25	N

mode implementation			
Add IntensityDrivenComponents for particle effects, shaders and audio	8	7.5	N
Add shaders, particle effects and audio to be driven by new components	8	8	N
Reverse lookup of weather and humidity values at a given point	16	2	N

10.7. Sprint 7 - ending 22/03/18

Stage seven focused on the project report write up. This included the writing up of stages one to six. Alongside this, research was conducted to for the legal, social and ethical section of the report.

Investigation into how to implement the temperature and humidity reverse lookup in the DoubleDictionary data structure. This will be used for WeatherEvents in the 'Manual' weather mode. Considerations were made to performance, due to the fact that, internally, DoubleDictionary is a dictionary of dictionaries meaning the time complexity of access via the stored inner value is O(n). However, due to the small number of elements held in the DoubleDictionary this was deemed an acceptable value (Table 8).

Table 8 - Sprint 7 work summary

Goals	**Time Allocated (hrs)**	**Time Used (hrs)**	**Complete (Y/N)**
Manual weather mode implementation	5	2	Y
Add IntensityDrivenComponents for particle effects, shaders and audio	2	1	Y
Add shaders, particle effects and audio to be driven by	2	1.5	Y

new components			
Reverse lookup of weather and humidity values at a given point	14	2	N
Stages write up for final report	4	3	N
Legal section research for final report	2	1	Y

10.8. Sprint 8 - ending 29/03/18

This stage focused on visual changes, a fix for weather transitions suddenly snapping to new values and editor refactors. Firstly, a new terrain was created for testing and as a potential platform for the final project demonstration. Alongside this, trees were generated for later testing of integration with Unity's "WindZone" component.

Secondly, a fix was implemented to stop weather events snapping to new intensities suddenly during transitions, causing highly unrealistic jumps in weather. The fix involved implementing a time-tracking class that acted as a wrapper to Unity's static "Time" class; "TimeExtension". TimeExtension provided a way to only update the time since the level was loaded in frames where a check was made against the time. For example, a frame in which no time check was made would result in the time between that and the previous frame being discarded from the total tracked time since the level loaded. This was ultimately unsuccessful in fixing the snapping issue, but did highlight a bug which caused the WeatherStations' queries for weather information at their locations to be artificially inflated the cumulative tracked value of the wind. This bug was resolved and the wind strength was increased as a result, to offset the fact that it was no longer being increased 50 times per frame rather than the desired single increase.

Finally, a refactor of the WeatherEvent and WeatherManager editors was undertaken in order to consolidate shared functionality, in the form of generic field-drawing methods, into a common "WeatherEditor" parent class (Table 9).

Table 9 - Sprint 8 work summary

Goals	**Time Allocated (hrs)**	**Time Used (hrs)**	**Complete (Y/N)**
Reverse lookup of weather and humidity values at a given point	14	8	Y

Stages write up for final report	1	1	Y
Tree generation	4	6	Y
Time Extension	4	3	Y
Consolidate editor functionality to parent class	2	3	Y

10.9. Sprint 9 - ending 12/04/18

Stage 9 added callback delegates to weather transitions. This allows any script to attach a method to be called when the transition begins, at each transition step and upon completion of the transition. Each delegate is called with a “WeatherChangeEventArgs” object as a parameter, exposing the weather events (and therefore underlying values such as intensity) involved in the transition.

Most instrumentally, a major refactor was made to the intensity call hierarchy. The old method of passing a single float was replaced with an “IntensityData” object. Alongside the old value, the current temperature and humidity value enums were also included. This allows for visual component controllers to conditionally activate on temperature and humidity values.

Additionally, visual snow elements were produced. This included a simple snow particle effect for snow fall and a shader for snow buildup over time. An IntensityDrivenComponent derived parent class was also produced, “TempHumidityIntensityDrivenComponent”, using the new data provided by the intensity hierarchy to allow rain and snow to determine when to activate (Table 10).

Table 10 - Sprint 9 work summary

Goals	**Time Allocated (hrs)**	**Time Used (hrs)**	**Complete (Y/N)**
Weather transition callback delegates	4	4	Y
More data passed through intensity hierarchy	16	14.5	Y
Snow particle system	4	1.5	Y
Snow shader	8	7.5	Y
Conditional intensity driven behaviours	4	3.5	~Y

using new hierarchy structure			

10.10. Sprint 10 - ending 03/05/18

The most dramatic change in this stage was the resolution of the snapping-weather bug. This was done by altering the way in which IntensityDrivenComponents, and derived controllers, were disabled. Rather than an instantaneous on-to-off-style disable, a coroutine was added to gradually disable weather effects. This means that weather effects shared by multiple weather types, for example cloud visualisations and their associated controller being used in rain and overcast weathers, when disabled as part of the transitioning process, will only change by a negligible amount prior to having the disable coroutine cancelled when its enabled in the next frame. The transitions were also improved by flipping the order in which the transitioning weather events are updated halfway through the transition in order to further reduce any snapping. Finally, curve values were updated to get more realistic weather patterns for each weather event (Table 11).

Table 11 - Sprint 10 work summary

Goals	**Time Allocated (hrs)**	**Time Used (hrs)**	**Complete (Y/N)**
Fix snapping weather bug	8	32	Y
Update hierarchy curves	4	3.5	Y
Conditional intensity driven behaviours disable behaviour fix	4	4	Y

10.11. Sprint 11 - ending 17/05/18

This was the final sprint and as a result included many minor fixes and changes in preparation for the conclusion of the project. This included a fix for volumetric lights for changing suddenly even after a gradual change towards the values. Similarly, some behaviours were snapping to values too quickly and so changes were implemented to make transitioning to new values more gradual.

A couple of final features were also added. For example, a wind controller behaviour that tied into Unity's WindZone system was added, which allowed particle effects and trees to be affected by the generated wind. As part of this addition, extra query methods were added to the weather manager in order to access required data, such as the cumulative wind and the wind at the present moment. The final feature to be added was the ability to explicitly set the generation seed through the weather manager.

Finally, the demo was created to show of the complete weather system, using an example scene by Unity, “The Viking Village”. The WeatherSystem package was imported and the example prefabs used to set up the system in the scene, including an onscreen display of the weather data at the current position. Testing was performed on OSX to confirm the build ran successfully on both operating systems as per the requirement presented in the project initiation document. (Table 12).

Table 12 - Sprint 11 work summary

Goals	**Time Allocated (hrs)**	**Time Used (hrs)**	**Complete (Y/N)**
Instance events implementation - use noise generator to ensure deterministic	4	3	Y
Further refinement of snapping issue	8	9	Y
Volumetric lighting snapping fix	8	6	Y
Visual and audio for all weather events	16	14	Y
Integration with Unity WindZones	8	7	Y
Queryability of wind and intensity values	2	1	Y
Weather info UI display	2	1	Y
Seed setting through weather manager	2	0.5	Y
Update documentation	4	2	Y

11. End-Project Report

The only concrete method in which to determine if a project has been a success is to scrutinise the extent to which it has met its objectives. As such, a critical evaluation of the resultant product is presented here. Two sources of data have been used in the evaluation; a questionnaire completed by final stage students (Appendix D.1) and a comparison of generated weather and real world weather data (Appendix E.1, E.2).

11.1. Be free, open-source and reusable without limitation

The project delivered and surpassed on this objective. The project is available under the MIT licence on Github whilst also providing documentation and example scenes and implementations. Furthermore, the project will be submitted to the Unity Asset Store on completion of the project, increasing the exposure of the package and further increasing the ease with which developers can integrate the package with their games.

11.2. Be procedural

Procedural generation is the quasi-random algorithmic generation of some content. In this instance, that content is the temperature, humidity, wind, and weather intensity and the generation comes from two-dimensional perlin noise, which results in gradual output changes as the input gradually changes (Figure 3). Because perlin noise is deterministic, the same outputs are given for the same inputs. This means that by offsetting the positional data given to the noise generation function by some fixed "seed", new data can be generated to allow for original content, or the seed can be fixed to have a repeatable experience (Figure 4). As such, the produced weather system delivers on this objective as a whole. However, the degree to which it meets each sub-objective is variable. The goal to "provide a platform from which weather patterns can be new and interesting on each playthrough of any game using the system" is fully realised, with seed values which can be modified programmatically or in the editor, that result in new weather patterns. This

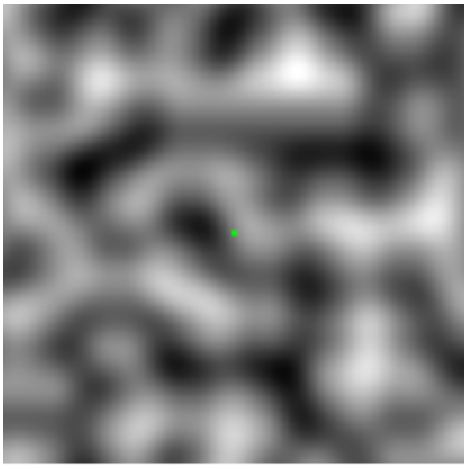

Figure 3 - A visualisation of perlin noise generation for each pixel in a texture. White denotes a value of 1 and black a value of 0. The green square denotes the sampling of the perlin noise at some position.

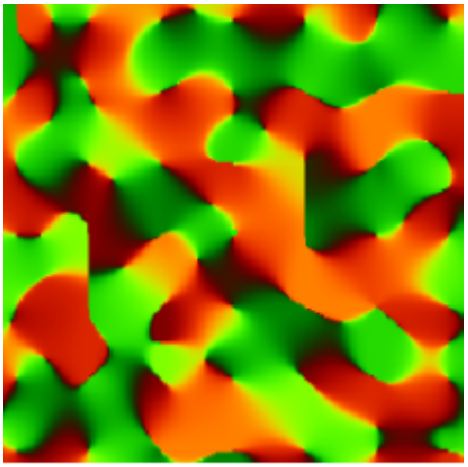

Figure 4 - A visualisation of the perlin noise generated wind vectors in the system. Each pixel coordinate is used as the input for the generated noise. Red denotes force in the positive y axis, and green the positive x axis

obviously also fulfills the requirement that the system allow for seeded generation and therefore is able to produce reproducible weather patterns at the discretion of the developer. Furthermore, the generated patterns can also be modified by the developer through changes to in the editor; varying the temperature and humidity values which result in a certain weather type, increasing changeability by increasing the wind strength, or changing the size of areas of weather through changing world size and scale values.

The third procedural objective is where there is an argument the system falls short. Although weather varies gradually across the game world at any given moment in procedural mode, the added "Manual" weather mode, in which weather events are predetermined by the developer, results in a constant weather, with fixed temperature and humidity values across the game world. Equally, it could be argued that the system delivers on this goal in addition to providing the added feature of developer-curated content.

Related to the procedural generation is the believability of the produced system, which was highlighted as an important factor in the project initiation document. As such, testing has

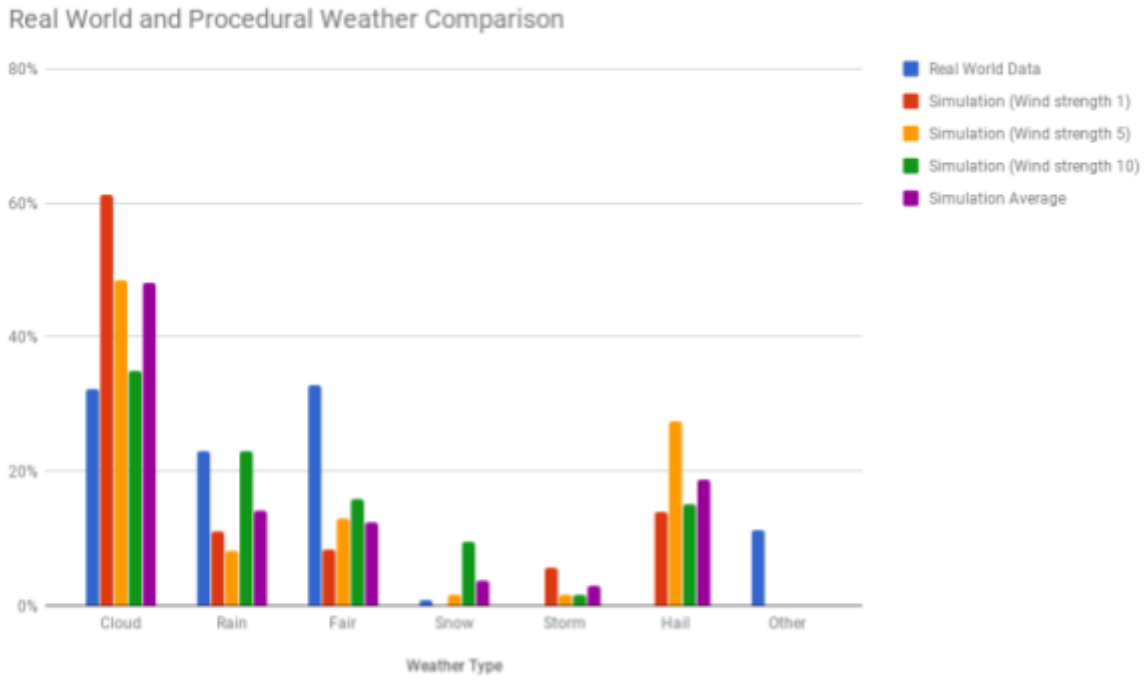

Figure 5 - A comparison of weather type occurrence in real world data and procedurally generated content

been performed to evaluate how similar the procedurally generated data is to that experienced by the author between January 31st 2017 and April 22nd 2018. This data was collected using the "If this then that", *IFTTT*, platform which allows a trigger, in this case the weather changing at a phone's position, to run a task, in this case saving the weather data to a spreadsheet. A script was written to log similar data to that of the IFTTT applet for the procedural weather system with the changeability of the system at three different settings. The grouping of the data is shown to be reasonably similar across all weather types included

in the procedural system, with the exception of hail, which was noticeably more prevalent in the procedural data (Figure 5). This may be due to the number of weather events included in the procedural generation. Increasing this value may result in greater similarity to the real world data. More importantly, in the questionnaire completed by final year students, the generated weather was considered to be 66% believable on average. This shows that even with sizable difference in the actual occurrence of weather events, the perceived difference is relatively small.

11.3. Be extensible

The extensibility of the system was an important consideration in its design due to the open-source nature of the project. Due to the fact that system has been designed with a hierarchical nature, and in an OO compliant manner, extensibility is inherent to the design. In fact, the example IntensityDrivenBehaviour extensions are implemented in the same way in which other developers could implement their own controllers. Questionnaire responses showed developers felt that the system was 70% extensible.

The software is also extensively documented so as to be easy to use for other developers, and therefore delivers on that objective. This is reflected in the questionnaire responses which rated the documentation at 68%. Furthermore, based on feedback on the documentation, further updates have been implemented resulting in more information on the structure of the scriptable object hierarchy and the functionality of manual mode alongside a User Guide (Appendix A).

11.4. Be queryable

In procedural weather mode, all values are fully queryable for any given position through the WeatherManager; raw temperature and humidity values and their equivalent enum values, weather type, intensity and wind values. The system, therefore, delivers on the requirement to allow for querying at arbitrary positions. This is highlighted by the feedback received from the questionnaire in which developers rated the queryability of the system at 70% on average.

Furthermore, callback methods are provided by the WeatherManager in the form of delegates for weather transitioning information. Separate callbacks are triggered at the start of a transition, at each transition “tick” and upon completion of the transition. These provide the weather events which are being transitioned between and therefore exposes the current temperature, humidity and intensity values.

11.5. Be Unity Editor friendly

The weather system provides several custom editors for improved usability. Primarily the WeatherManager component only displays properties that are pertinent to the current mode of the weather system. For example, the weather lookup object is displayed in both

modes whereas the weather event sequence list is only displayed in manual mode. This means that alongside being editor-friendly, the system also allows for changes to initialization values without editing code and displays properties in an suitable manner in the Editor. An editor was additionally provided for adding new weather events which allows developers a fast initial setup, again without having to edit any code. This resulted in questionnaire respondents rating the editor-friendliness of the system at 70%.

11.6. Further features

During development, the scope of the project extended, adding new components and functionality.

11.6.1. Manual mode

The most dramatic addition to the system was included in the scope of the project within the second stage when it became apparent that the procedural element could easily be delivered within the project timescale. This mode added functionality to be able to set up sequences of weather events with associated data such as the length of time and intensity of each weather event. This additional feature allows developers to provide curated weather environments for key plot points or scripted scenes. It also allows games to show off their environments in a variety of weather backdrops for game trailers or in demonstrations.

Table 13 - Comparison of available weather packages in relation to the produced weather system

Product Name	Price	Types of Weather	Extensible	Terrain Interaction	Audio support	Example code/ scenes/ assets	Editor-friendly	Queryable
Weather Maker	$42	Fog, rain, snow, hail and sleet	Yes	Colliders and shaders	Yes	Yes	Yes	No
Enviro	$50	Clear Sky, cloudy, raining, stormy, snowy and foggy	Yes	Water only	Yes	No	Yes	No
UniStorm 2.4	$60	Sunny, Mostly Clear, Partly Cloudy, Mostly Cloudy, Foggy, Snow, Rain, Lighting and Thunderstorms	Yes	Shaders only	Yes	Yes	Yes	Yes
Time of Day & Weather System	Free	Sun, cloudy, rain thunderstorm and snow	Yes	No	No	Yes	Largely	No
Weather System	**Free + Open source**	**Clear, Cloudy/Overcast (variable), fog, storm, rain, snow, hail**	**Yes**	**Developer Defined, any**	**Yes**	**Yes**	**Yes**	**Yes**

11.7. Product Comparison

The final analysis on the success of the project is a comparison with the packages identified and the project's initiation (Table 13). The resultant package matches, and in some cases surpasses the functionality of even the most expensive of identified packages and as such must be viewed as a success.

12. Project Post-Mortem

12.1. Objective Delivery

The drive for this project was the lack of availability to fully featured, open-source weather systems, specifically for hobby and independent developers. The objectives, therefore, provided a platform from which the resolution of this goal could be met. As such, the project was successful in both its delivery of the final product and the solution to the core issue.

12.2. Technology

The selection of technology for this project has been shown to be advantageous. The choice of Unity and C# was well founded for delivery of the project to the identified market, and productivity of development time. Further, Unity's integration with C#, and therefore compatibility with Visual Studio, has also allowed for productive development and fast implementation due to the prior fluency in the language and platform. Similarly, prior experience with the Axosoft platform has enabled powerful project management within an agile SCRUM framework.

The selection of Github as the source control platform has not only provided a robust solution for code management but also has increased the author's exposure to the industry-standard git platform.

12.3. Project Management and Methodology

The selection of SCRUM over Cowboy has proven to be the right decision. This is not only due to the availability of existing tools, such as Axosoft, for SCRUM support but also due to the stricter rule-set the SCRUM methodology provided over Cowboy, in turn allowing for a more productive, iterative development process.

The application of project management techniques largely conformed to SCRUM methodology in that work was segmented into sprints with backlog items being assigned for development in each iteration, with consideration given to the time required versus the

available time. However, the concept of “daily scrums” were not followed due to the fact that development was completed by a single developer. Therefore, the work remaining for the sprint was reviewed whenever a development session was started, to gauge progress towards the sprint goals, following a similar mentality to that of daily scrums. Similarly, the sprint review and retrospect element was fulfilled by the highlight meetings and reports alongside the Sprint Assessment documents.

Although no formal branching strategy was employed in the source repository, the project did not suffer for it. This is primarily due to the fact that the project was conducted by only a single developer. Had there been other developers, a feature-based branching strategy would certainly have benefitted the project. Arguably, feature branches could have improved productivity by allowing work on multiple different features simultaneously. With that said, had the lack of branching become a problem in this project then a strategy would have been implemented.

12.4. Developer Performance and Lessons Learned

Overall, performance over the course of the project has been considerable. Intermediary deliverables were delivered in a timely manner and to a high standard. Similarly, the produced code is of a high quality, conforming to accepted coding standards and object-oriented principles whilst integrating well with Unity’s component based system. Furthermore, the design of the system has enabled the delivery of the queryability and extensibility objectives.

The fact that the target deliverables were surpassed and additional functionality was included in the project highlights that the performance throughout the project was significant.

The design elements of the project could have been improved, however. Although designs for the system as a whole and its subsystems were developed, the process was often relatively informal, with use of a whiteboard. Furthermore, a more rigorous, all-encompassing process could have been employed in order to better predict the requirements of the system once the core objectives had been met. This may have allowed for a more flexible implementation and therefore eased the addition of additional features. Similarly, examining implementations of available packages may have been beneficial to developing the system structure. Additionally, although the project supervisor in some regards acted as a client, it may have been advantageous to identify a formal client figure, who also had good knowledge of the problem area in order to better prioritise features and allocate development time, as per their needs. This may also have provided a more effective process by which to critically evaluate the resultant implementation.

13. Conclusions

The project delivered on and surpassed its objectives. The resultant product offers a powerful alternative platform for developers to implement weather into their projects without monetary cost, allowing for greater immersion and realism in more games. The release of the project under a permissive, open-source licence allows developers both the freedom to use the software as they see fit, and presents an opportunity for the project to be further enhanced with further features and bug fixes by other developers.

The success of the project is due to the clear objectives defined at the project's inception, and the effective identification and execution of both project management methodology and choices of software. However, development could have been improved through the implementation of a more formal client and better defined features in excess of the minimum viable product.

Finally, the benefits of the project as an experience is not to be dismissed. The development of the project has offered the opportunity to better understand the management of a larger software package and the development of a software solution in a formal setting, as a solo-developer.

14. References

Barton, M. (2008). How's the Weather: Simulating Weather in Virtual Environments. *The international journal of computer game research*, [online] 9(1). Available at: http://gamestudies.org/0801/articles/barton [Accessed 12 May 2018].

Bay 12 Games. (n.d.). *Bay 12 Games: Dwarf Fortress*. [online] Available at: http://www.bay12games.com/dwarves/features.html [Accessed 12 May 2018].

Bcs.org. (n.d.). *Code of conduct*. [online] Available at: https://www.bcs.org/category/6030 [Accessed 12 May 2018].

Bertz, M. (2018). *Frostpunk Review – A Frigid, Unrelenting Survival Success*. [online] Game Informer. Available at: http://www.gameinformer.com/games/frostpunk/b/pc/archive/2018/04/23/frostpunk-review-a-frigid-unrelenting-survival-success.aspx [Accessed 12 May 2018].

Bronte, C. and Davies, S. (2006). *Jane Eyre*. 3rd ed. Suffolk: Penguin Classics, p.xi-296.

Cheng, C. (2012). *Story Points Versus Task Hours*. [online] Scrumalliance.org. Available at: https://www.scrumalliance.org/community/articles/2012/august/story-points-versus-task-hours [Accessed 12 May 2018].

Cunningham, M. (1979). Weather, mood, and helping behavior: Quasi experiments with the sunshine samaritan. *Journal of Personality and Social Psychology*, 37(11), pp.1947-1956.

Docs.unity3d.com. (2018). *Unity - Manual: Execution Order of Event Functions*. [online] Available at: https://docs.unity3d.com/Manual/ExecutionOrder.html [Accessed 20 May 2018].

Epublications.marquette.edu. (2018). *Glossary of the Gothic: Weather | Glossary of the Gothic | Marquette University*. [online] Available at: https://epublications.marquette.edu/gothic_weather/ [Accessed 12 May 2018].

Frushtick, R. (2018). *Sea of Thieves review*. [online] Polygon. Available at: https://www.polygon.com/2018/3/22/17151248/sea-of-thieves-review-xbox-one-pc [Accessed 12 May 2018].

Gotterbarn, D. (n.d.). [online] Cscietsu.edu. Available at: http://csciwww.etsu.edu/gotterbarn/artge2.htm [Accessed 12 May 2018].

Gousios, G., Vasilescu, B., Serebrenik, A. and Zaidman, A. (2014). Lean GHTorrent: GitHub Data on Demand. In: *MSR 2014 Proceedings of the 11th Working Conference on Mining Software Repositories*. [online] New York: ACM. Available at: https://dl.acm.org/citation.cfm?id=2597126 [Accessed 12 May 2018].

Hirshleifer, D. and Shumway, T. (2001). Good Day Sunshine: Stock Returns and the Weather. *SSRN Electronic Journal*.

Hudson, L. (2018). *No Man's Sky Is Like Real Space Exploration: Boring, Except When It's Sublime*. [online] Slate Magazine. Available at: http://www.slate.com/articles/technology/future_tense/2016/08/no_man_s_sky_offers_18_quintillion_planets_for_players_to_explore.html [Accessed 12 May 2018].

Kengaskhan (2018). *How Will Fog on PUBG's Map Affect Your Gameplay?*. [online] Gameskinny.com. Available at: https://www.gameskinny.com/pwj6q/how-will-fog-on-pubgs-map-affect-your-gameplay [Accessed 12 May 2018].

Let the Right One In. (2008). [DVD] Directed by T. Alfredson. Sweden: EFTI.

Markowitz, M. (1999). *Spacewar*. [online] Games of Fame (Archive). Available at: https://web.archive.org/web/20060426231325/http://www3.sympatico.ca/maury/games/space/spacewar.html [Accessed 12 May 2018].

Minecraft Wiki. (n.d.). *The Overworld*. [online] Available at: https://minecraft.gamepedia.com/The_Overworld#Limitations [Accessed 12 May 2018].

Murray, J. (1997). *Hamlet on the holodeck*. Cambridge, Mass.: MIT Press.

Nicholls, M. (2013). A Cold Day In Hell: Gothic Horror in Let The Right One In. *The Missing Slate*. [online] Available at: http://themissingslate.com/2013/04/15/a-cold-day-in-hell-gothic-weather-in-let-the-right-one-in/ [Accessed 12 May 2018].

Opensource.org. (n.d.). *The MIT License | Open Source Initiative*. [online] Available at: https://opensource.org/licenses/MIT [Accessed 12 May 2018].

PLAYERUNKNOWN'S BATTLEGROUNDS Forums. (2018). *Weather removed?*. [online] Available at: https://forums.playbattlegrounds.com/topic/163622-weather-removed/ [Accessed 20 May 2018].

Porreca, R. (2017). *It's hard as hell to see during PUBG's foggy weather*. [online] Destructoid. Available at: https://www.destructoid.com/it-s-hard-as-hell-to-see-during-pubg-s-foggy-weather-461377.phtml [Accessed 12 May 2018].

Rare (2017). *Sea of Thieves Inn-side Story #16: Storms*. [video] Available at: https://www.youtube.com/watch?v=v7DgXtX2rXk [Accessed 12 May 2018].

Roberts, S. and Patterson, D. (2017). Virtual weather systems: measuring impact within videogame environments. In: *Australasian Computer Science Week Multiconference*. [online] Geelong. Available at: https://dl.acm.org/citation.cfm?id=3014878 [Accessed 12 May 2018].

Schindler, E. (2007). *Enterprise Developers Programming Speed? Check. Time to Fix Bugs? Not So Fast. | CIO - Blogs and Discussion*. [online] Advice.cio.com (Archive). Available at: https://web.archive.org/web/20100109123853/http://advice.cio.com/esther_schindler/ent

erprise_developers_programming_speed_check_time_to_fix_bugs_not_so_much [Accessed 12 May 2018].

Schulz, K. (2015). Writers in the Storm. *The New Yorker*. [online] Available at: https://www.newyorker.com/magazine/2015/11/23/writers-in-the-storm [Accessed 12 May 2018].

Scratchapixel. (n.d.). *Perlin Noise: Part 2*. [online] Available at: http://www.scratchapixel.com/lessons/procedural-generation-virtual-worlds%20/perlin-noise-part-2/perlin-noise [Accessed 12 May 2018].

Steve Russell (1962), *Spacewar!* [Video game]. Steve Russell
The Dwarf Fortress Wiki. (n.d.). *DF2014:Weather - Dwarf Fortress Wiki*. [online] Available at: http://dwarffortresswiki.org/index.php/DF2014:Weather [Accessed 12 May 2018].

The Lord of the Rings: The Two Towers. (2002). [DVD] Directed by P. Jackson. New Zealand and United States.

Thomas, A. (2007). *NCAA Football 08 Review*. [online] GameSpot. Available at: https://www.gamespot.com/reviews/ncaa-football-08-review/1900-6175449/ [Accessed 12 May 2018].

15. Appendices

Appendix A - User Guide

Can be found online in PDF and Markdown formats.

Appendix B - Project Management Artefacts

B.1 - Project Initiation Documents

B.1.1 - Project Proposal

PRCO304 - Project Proposal

Samuel Lord, Computing and Games Development - 10485852

samuel.lord@students.plymouth.ac.uk

Keywords

Weather simulation, parallel programming, game developer tool, C++ Lib, Unity asset

A (semi) realistic weather system API for game developers utilising parallelised GPU code written for CUDA (i.e. NVIDIA cards), with a view to compiling through HIP or a similar tool to allow for use on other (OpenCL compatible) systems. The system will be inspired by forecasting and simulation techniques and reduced/simplified in order to run on consumer hardware in games.
The culmination of the project will be demonstrated through Unity game engine as an example front end for the system.
The project will build on principles from SOFT354 Parallel Computation and Distributed Systems through the addition of further weather phenomena, interaction and more realistic simulation.

The employed methodology will be agile, sprint-based, scrum-like method called Cowboy (1)

Development will be C/C++ for the core system with any front end code for Unity in C#, both developed in Visual Studio

Required learning is majoritively in meteorological systems. Introductory books on the topic are available in the library to support online materials.

1. Cowboy development methodology (https://scholarscompass.vcu.edu/cgi/viewcontent.cgi?article=1740&amp=&context=etd&amp=&sei-redir=1&referer=https%253A%252F%252Fwww.google.com%252Furl%253Fq%253Dhttps%253A%252F%252Fscholarscompass.vcu.edu%252Fcgi%252Fviewcontent.cgi%253Farticle%25253D1740%252526context%25253Detd%2526sa%253DD%2526ust%253D1512661428337000%2526usg%253DAFQjCNFWBjETpZQt4AaamKENk4yWB700aA#search=

%22https%3A%2F%2Fscholarscompass.vcu.edu%2Fcgi%2Fviewcontent.cgi%3Farticle%3D1740%26context%3Detd%22)

Learning Required

Current weather forecasting/simulation techniques including fluid dynamics and thermodynamics.
Meteorological/atmospheric models.
https://www.theguardian.com/science/alexs-adventures-in-numberland/2015/jan/08/banking-forecasts-maths-weather-prediction-stochastic-processes
http://maths.ucd.ie/~plynch/LECTURE-NOTES/DYNAMIC-Met-2004/

B.1.2 - Project Initiation Document

PRCO304 Project Initiation Document

Samuel Lord 10485852

Introduction

Many games require the use of weather for immersive environments and atmosphere. The weather solutions available vary in quality, scope and their focus on visuals or simulation.
However the availability of free, open-source, extensible and consistently repeatable weather package for developers to use are near non-existent. As such, this project looks to address these shortcomings by providing a procedural weather platform to which developers can add their own assets to create immersive experiences.

Product Case

Product Need

There are many options available to developers looking to integrate weather into their games. Packages vary from purely visual to more mechanistic packages which contain little to no visuals. They also vary broadly in quality and scope and those with worthwhile depth often, understandably, have a notable price attached. An open-source alternative, which could later become a community-driven project, would have applications for 'indie' developers, and in educational settings.

In order to inform the specification of the product, research has been conducted to identify the functionality offered by similar products, and is shown in the table below (Assetstore.unity.com, 2018).

Product Name	Price	Types of Weather	Extensible	Terrain Interaction	Audio support	Example code/ scenes/ assets	Editor-friendly	Queryable
Weather Maker	$42	Fog, rain, snow, hail and sleet	Yes	Colliders and shaders	Yes	Yes	Yes	No
Enviro	$50	Clear Sky, cloudy, raining, stormy, snowy and foggy	Yes	Water only	Yes	No	Yes	No
UniStorm 2.4	$60	Sunny, Mostly Clear, Partly Cloudy, Mostly	Yes	Shaders only	Yes	Yes	Yes	Yes

		Cloudy, Foggy, Snow, Rain, Lighting and Thunderstorms						
Time of Day & Weather System	Free	Sun, cloudy, rain thunderstorm and snow	Yes	No	No	Yes	Largely	No

The identified products and key areas each package covers has informed the project objectives and the initial scope.

Project Objectives

The presented weather platform aims to meet five key objectives:

1. Be free, open-source and reusable without limitation
2. Be procedural
    a. Will provide a platform from which weather patterns can be new and interesting on each playthrough of any game using the system.
    b. Will be seeded and therefore will generate reproducible weather patterns at the discretion of the developer, and enable easier testing.
    c. Weather will vary across the game world at any given moment, adding more realism. This is in contrast to the researched products which unanimously had ubiquitous weather throughout the game world.
3. Be extensible
    a. Will be designed so as to be extensible by developers using the platform.
    b. Will be well documented so as to be easy to use for other developers.
4. Be queryable
    a. Will allow developers to request information about the weather at specific locations and times. This allow for games which use information such as temperature to, for example, change the look of the player or environment in different conditions or provide information to survival-like games.
5. Be Unity Editor friendly
    a. Changes to initialization values will not require editing code and will be available as sliders/editable text boxes in the Editor.
    b. As and when required, custom editors may be written to automate repetitive tasks.

Initial Scope

1. The following weather processes will be included in the platform: Clear sky, rain, storms, snow, overcast and integration with Unity's wind zones. The weather system will be produced in C# as a package for Unity Game Engine. The weather system will control particle and audio effects alongside simple shaders and lighting.

2. The product will not be a simulation, however it will take inspiration from weather simulation and historical data. For example, historical weather data may be analysed to determine the chance of one type of weather leading to another and this may inform the balancing of the system. This will enable an immersive experience to players through realistic transitions between weather states.
3. The product will be cross platform (Windows/OSX) compatible, both for editing/development and gameplay.
4. An example demo implementation utilising the package will be produced in Unity Game Engine (Unity, 2018) for Windows. The demo will allow a first person perspective view of a game world that has the produced weather package integrated, showing weather information at the players current position on screen whilst visual weather effects can also be seen.

Method of Approach

The project will utilise a SCRUM agile methodology for software development, with sprints lasting one week to match the project highlight report requirements. The meetings as defined in SCRUM will be slightly modified to better fit the project requirements and as such, meetings with the Project Supervisor Marius Varga, will act as an end-of-sprint meeting equivalent.

The utilised technologies will be C# and Unity Game Engine for software development. Simple game assets, for example, textures will be produced for the demo and these will be created using Photoshop and GIMP.

Initial Project Plan

Initial Project Plan Deadlines			
Stage	**Start Date**	**End Date**	**Outcomes**
1. Initiation	16 Jan	26 Jan	This document
2. Detailed requirements	Mon 29 Jan	Mon 5 Feb	Requirements document, Initial backlog creation, Licence creation
3. System Design	Mon 5 Feb	Mon 12 Feb	High level UML, Library structure, Development environment ready, user tests designed
4. Increment 1	Mon 12 Feb	Mon 26 Feb	Package V1 and first testing session
5. Increment 2	Mon 26 Feb	Mon 5 March	Review testing results

			and integrate feedback, produce package V2. Second testing session.
6. Increment 3	Mon 5 March	Fri 19 March	Integrate changes based on feedback. Demo 'front-end' V1. Third testing session.
7. Increment 4	Mon 19 March	Mon 26 March	Integrate changes from third testing session. Demo 'front-end' V2
8. Complete System	Mon 26 March	Fri 30 March	Integrate any further changes. Produce Weather Package V1
9. Testing and repackage	Mon 16 April	Mon 23 April	Bug fixes, rebuilt package, code documentation finalised, library made public on github
10. Assemble + complete final report	Mon 23 April	Fri 4 May	Final Report

Control Plan

Code increments will utilise agile sprints, on a weekly basis. Highlight reports will be conducted in line with sprints, or weekly when no sprints are active. Further, stage reports will be generated at the end of each stage.

Communication Plan

Review meetings with the project supervisor will initially occur weekly and the frequency of these meetings will be reviewed once stage 4 begins.

Initial Risk List

Project Risks and Mitigations	
Schedule Overrun	The easter break has been discounted from the initial project plan and may be used as contingency should it be needed. In the event that the project is behind schedule more than 1

	week, a meeting with the project supervisor will be arranged and an exception plan developed.
Illness	The project supervisor will be notified in the event of illness. Other mitigation as above.
Lack of knowledge to implement required features	Meteorological knowledge: The library has an extensive collection of works on meteorology, alongside other works available online. Software knowledge: Project supervisor and other staff members for support and extensive work online for reference.
Technology failure	Written reports will be stored in google drive and copied to SPMS for submission. Code will be kept in a GIT repo hosted on github.com, with copies kept on University and personal computers as required.
Testing	In the case of a lack of testers at the end of each code sprint, the project supervisor will act as a tester. Further tests will be performed as soon as possible

Quality Plan

Requirements	As defined in the project objectives and initial scope. Further depth will be added in stage 2. It will be ensured that they are relevant, correct and complete. Throwaway prototyping will be used for implementing subsystems.
Design Validation	The design will be checked against the requirements document produced in stage 2. The design must comply with code structure standards which will largely follow Microsoft's Coding Techniques and Programming Practices (Msdn.microsoft.com, 2018).
Sprint Review	Each sprint will end with a review in which the backlog is re-prioritised and work completed that sprint is confirmed to have fulfilled the requirements necessary.
Publication	The publication of the package will occur in stage 9, after the extensive testing of stage 8.

	Documentation will have been completed iteratively during development.

Legal, Ethical and Social Issues

This project does not require research on human subjects.
The code produced in this project will be released open source. As such, the licence for the released code needs to be considered to enable developers to be able to use the resulting library. The licence will be written in stage 2.
The licences for Unity also needs to be considered. Unity's licence states that content created with the engine belongs to the creator.
During the development of the weather package testing may take place, using other students or university staff. This testing is covered by the approved PRCO304 usability study application.

References

Axosoft.com. (2018). *Agile Project Management | Axosoft*. [online] Available at: https://www.axosoft.com [Accessed 11 Jan. 2018].

Assetstore.unity.com. (2018). *Enviro - Unity Asset Store*. [online] Available at: https://assetstore.unity.com/packages/tools/particles-effects/enviro-sky-and-weather-33963 [Accessed 29 Jan. 2018].

Assetstore.unity.com. (2018). *Time of Day & Weather System - Unity Asset Store*. [online] Available at: https://assetstore.unity.com/packages/tools/particles-effects/time-of-day-weather-system-40374 [Accessed 29 Jan. 2018].

Assetstore.unity.com. (2018). *UniStorm - Unity Asset Store*. [online] Available at: https://assetstore.unity.com/packages/tools/particles-effects/unistorm-2714 [Accessed 29 Jan. 2018].

Assetstore.unity.com. (2018). *Weather Maker - Sky, Weather, Fog, Volumetric Light and Dynamic Environment (2D & 3D) - Unity Asset Store*. [online] Available at: https://assetstore.unity.com/packages/tools/particles-effects/weather-maker-sky-weather-fog-volumetric-light-and-dynamic-envir-60955 [Accessed 29 Jan. 2018].
GitHub. (2018). *HIP*. [online] Available at: https://github.com/ROCm-Developer-Tools/HIP [Accessed 11 Jan. 2018].

Msdn.microsoft.com. (2018). *Coding Techniques and Programming Practices*. [online] Available at: https://msdn.microsoft.com/en-us/library/aa260844(v=vs.60).aspx [Accessed 11 Jan. 2018].

Unity. (2018). *Unity*. [online] Available at: https://unity3d.com/ [Accessed 11 Jan. 2018].

B2 - Stage Highlights

B.2.1 - Highlight Report 1

PRCO304: Highlight Report
Name: Sam Lord
Date: 06/02/2018
Review of work undertaken *A requirements document was produced to more clearly define the requirements of the project.* *A project github repo and the project management tool, Axosoft, was set up.* *A prototype of the weather system was produced demonstrating particle effects being driven by a procedural algorithm, with weather changing over time. This included a simple particle management system for displaying the correct particle system for each weather type. This prototype was driven by perlin noise and two values were produced representing the humidity and temperature for a given x,y position in the game world. These two values were used to calculate a weather type. For example, a high temperature and high humidity might result in stormy weather whereas mid temperatures and mid humidity might produce sleet.* *Real weather data collected from passive logging over the last year was compiled into a single document to be used as reference for realistic weather transitions.*
Plan of work for the next week *Develop detailed UML diagram and implement the base package structure from this design.* *Design the user tests to be performed in each testing session.* *Consider and document the various visual impacts that might be affected by changes in weather and research by what methods developers are most likely to implement those in games. For example, snow might be displayed by shader or particle effect. This will inform some of the package structure.*
Date(s) of supervisory meeting(s) since last Highlight 07/02/2018
Brief notes from supervisory meeting(s) since last Highlight *The use of real world data being used to inform the changes in the weather system was discussed, and deemed to be a useful tool for judging the realism of the system.* *Weekly meetings were decided upon for the immediate future.* *The visual changes that might be impacted by weather changes were discussed alongside the kind of control and customisation that will be offered to developers in the package's final version.*

B.2.2 - Highlight Report 2

PRCO304: Highlight Report
Name: Sam Lord
Date: 14/02/2018
Review of work undertaken *A repository for the project proper was created with a custom gitignore, including a simple readme and licence.* *A UML was constructed in Visual Studio from which all the base classes were generated. The classes were then edited to correct any inheritance errors, eg WeatherSet and WeatherEvent were changed to inherit from ScriptableObject.* *Custom editors were produced for dynamically adding WeatherTypes enums. Similarly a custom editor was made for WeatherManager which will allow for dynamically switching between procedural and manual modes in the editor. An editor for* *A DoubleDictionary data structure was created which requires two keys, rather than the usual one to lookup a value. This is the underlying data structure for ProceduralWeatherLookup. Further, a ProceduralWeatherLookup editor was produced, however serialization issues currently make it unusable.* *The scripts and assets created so far were reorganised in Unity to make management easier in the longer term. A new namespace was introduced, 'WeatherSystem.Internal', for the inner workings of the system.* *The user tests and visual implementation research that were scheduled for this sprint were delayed to the next sprint in order to focus on bug fixes.*
Plan of work for the next week *The user tests for the end of the next sprint (2 weeks time)* *ProceduralWeatherLookup serialization in editor fix.* *Procedural Weather implemented matching the functionality of the prototype.* *Research into Unity's curves system and whether that might be applicable to the weather system's manual mode.* *Research by what methods developers are most likely to implement visual weather changes in games.*
Date(s) of supervisory meeting(s) since last Highlight 14/02/2018
Brief notes from supervisory meeting(s) since last Highlight *Discussed the possibility of subsystems for WeatherEvents whose intensity could be driven by curves.* *Discussed transitioning between weather types using curves.*

B.2.3 - Highlight Report Stage 3

PRCO304: Highlight Report
Name: Sam Lord
Date: 22/02/2018
Review of work undertaken *The serialization issues experienced in the custom ProceduralWeatherLookup editor were fixed. Similarly bugs in the WeatherTypes editor were fixed. Firstly, a null reference would be caused when editor code was recompiled and secondly no edits would be possible until the array was resized.* *Noise generation functions were implemented based on the work conducted for the prototype, including seeding. Further, basic noise-based wind represented by a Vector2 was implemented and used to drive the sample location of the weather perlin noise map.* *Some scripts were produced to visualise the noise generation and enable faster testing and iterations.* *Preliminary user test designs and weather system visualisation research were produced.*
Plan of work for the next week *Research into Unity's curves system and whether that might be applicable to the weather system's manual mode.* *Further research by what methods developers are most likely to implement visual weather changes in games.* *Produce early MVP version of the weather system.*
Date(s) of supervisory meeting(s) since last Highlight [None]
Brief notes from supervisory meeting(s) since last Highlight *No meeting since last highlight*

B.2.4 - Highlight Report Stage 4

PRCO304: Highlight Report
Name: Sam Lord
Date: 01/03/2018
Review of work undertaken *Did research into Unity's curves system and implemented curves for intensity transitions for weather changes.* *Updated and added new weather events* *Fixed weather transitions where weather would constantly loop between two weather types.* *Implemented first WeatherManager weather queries for external objects to get weather at their positions.* *Added new visualisation in the form of 'Weather Stations' which display the type of weather at their position and changes weather. Also created an accompanying distributor, which spawns a number of stations.*
Plan of work for the next week *Implement transitioning for WeatherSets, both procedural and manual* *Implement manual WeatherEvents transitioning* *Further extend transitioning including intensity driving internal values eg. visibility, precipitation etc* *User-test package*
Date(s) of supervisory meeting(s) since last Highlight 27/02/2018
Brief notes from supervisory meeting(s) since last Highlight *Discussed the inclusion of internal weather values being driven by a single intensity value, and Lerp-ing between WeatherEvents using intensity values*

B.2.5 - Highlight Report Stage 5

PRCO304: Highlight Report
Name: Sam Lord
Date: 08/03/2018
Review of work undertaken *Introduced weather properties and reliant weather properties. Updated procedural weather to use the same system. This allows for properties to be independently applied to each WeatherEvent.* *Imported a first person controller and volumetric lighting package. Also added Terrain from a previous project.* *Created a rain particle effect.* *Added a new WeatherStation distributor for placing weather stations on the terrain.* *Overhauled WeatherPropertys to use MonoBehaviours inheriting from IntensityDrivenBehaviour which are found at runtime.* *Added a new WeatherEvent editor for adding curves through to WeatherProperties objects at runtime.*
Plan of work for the next week *Complete manual weather mode implementation.* *Add MonoBehaviour controllers for visual and audio control from WeatherEvents.* *Add further visual and auditory implementations.* *Add getting temperature and humidity values from a WeatherEvent at a particular position.*
Date(s) of supervisory meeting(s) since last Highlight 08/03/2018
Brief notes from supervisory meeting(s) since last Highlight *Discussed progress this week. Also talked over adding visuals and audio alongside finishing up the implementation of manual mode. Finally, discussed starting writing the report within the next two weeks.*

B.2.6 - Highlight Report Stage 6

PRCO304: Highlight Report
Name: Sam Lord
Date: 15/03/2018
Review of work undertaken *Made substantial progress in completing the manual weather implementation.* *Almost all designed IntensityDrivenComponents added.* *Added further visual and auditory implementations.* *Investigated getting temperature and humidity values from a WeatherEvent at a particular position.*
Plan of work for the next week *Write up stages 1 to 6 for final report.* *Write up Legal section for final report.* *Complete temperature and humidity lookup for a weather event in manual weather mode implementation.* *Instance events implementation.*
Date(s) of supervisory meeting(s) since last Highlight 13/03/2018
Brief notes from supervisory meeting(s) since last Highlight *Brief meeting discussing what writing should be delivered by next week.*

B.2.7 - Highlight Report Stage 7

PRCO304: Highlight Report
Name: Sam Lord
Date: 22/03/2018
Review of work undertaken *Wrote up stages 1 to 6 for final report.* *Did research for Legal section for final report.* *Planned temperature and humidity reverse lookup for a weather event in manual weather mode implementation.*
Plan of work for the next week *Mind map for final report structure and content.* *Fix editor bug and snap transitioning between WeatherEvents* *Re-implement setting weather type enum for WeatherEvent in editor*
Date(s) of supervisory meeting(s) since last Highlight 21/03/2018
Brief notes from supervisory meeting(s) since last Highlight *Discussed planning and structuring final report including how mind mapping can be useful.*

Appendix C - Sprint Reviews

C.1 - Sprint Reviews

C.1.1 - Sprint 1 Review Document

Sprint Assessment

Sprint Number: 1	**Start:** 29/01/2018	**End:** 06/02/2018
Axosoft Stage Reference: https://samlord.axosoft.com *V1.0/Stage 1*		

Goal Completion

Goals	Time Allocated (hrs)	Time Used (hrs)	Complete (Y/N)
Initalise project repo with unity project	2	0.5	Y
Setup Axosoft	2	1	Y
Produce Requirements Document	2	1.5	Y
Throwaway prototype of procedural weather	16	15	Y
Compile real weather data in single document for later testing	2	2	Y

Total hours rolled over into next sprint: 0 hrs

Objectives

Objective	Achieved (Y/N)	Notes
Meet objective to be procedural	N	Prototype implementation works well and will be implemented into the final product soon
Meet objective to be extensible	N	
Meet objective to be queryable	N	
Meet objective to be editor friendly	N	

Tests

Test	Pass (Y/N)	Notes

Additional Comments:
No testing performed this sprint as throwaway prototype was the only code produced.

C.1.2 - Sprint 2 Review Document

Sprint Assessment

Sprint Number: 2	**Start:** 07/02/2018	**End:** 14/02/2018
Axosoft Stage Reference: https://samlord.axosoft.com *V1.0/Stage 2*		

Goal Completion

Goals	Time Allocated (hrs)	Time Used (hrs)	Complete (Y/N)
New project repository for actual implementation	0.5	0.5	Y
Create base class implementations based on prototype	8	4	Y
Refresh editor code knowledge through implementation of first editors	32	28	Y
Create object for lookup using two values - “DoubleDictionary”	8	4	Y
Design user tests	8	0	N
Research common implementations of weather effects (eg shaders vs particle effects etc)	8	0	N

Total hours rolled over into next sprint: 16 hrs

Objectives

Objective	Achieved (Y/N)	Notes
Meet objective to be procedural	N	
Meet objective to be extensible	N	
Meet objective to be queryable	N	
Meet objective to be editor friendly	N	Custom editors first stage to ensure this is met

Tests

Test	Pass (Y/N)	Notes
DoubleDictionary can add a value	Y	
DoubleDictionary can retrieve a value with the correct lookup keys	Y	
DoubleDictionary can remove a value given its keys	Y	
DoubleDictionary returns true when Trying to get a value is successful and false otherwise	Y	
Editor data is correctly assigned to object	Y	
On restarting the editor data changes are still applied	N	ProceduralWeatherLookup does not correctly serialize the DoubleDictionary data
Switching modes in the WeatherManager editor displays only pertinent variables	Y	

Additional Comments:
Further work to be completed to fix the serialization issues in the editor

C.1.3 - Sprint 3 Review Document

Sprint Assessment

Sprint Number: 3	**Start:** 15/02/2018	**End:** 22/02/2018
Axosoft Stage Reference: https://samlord.axosoft.com *V1.0/Stage 3*		

Goal Completion

Goals	Time Allocated (hrs)	Time Used (hrs)	Complete (Y/N)
Design user tests	8	1	Y
Research common implementations of weather effects (eg shaders vs particle effects etc)	8	4	Y
Fix serialization issues in WeatherLookup editor	8	8	Y
Implement procedural noise generation with seeded values	16	16	Y
Develop tools to test/visualise procedural noise	8	8	Y

Total hours rolled over into next sprint: 0 hrs

Objectives

Objective	Achieved (Y/N)	Notes
Meet objective to be procedural	N	Not currently driving weather, but procedural generation core is largely complete
Meet objective to be extensible	N	Generation structure conforms to OO principles
Meet objective to be queryable	N	At a low level, this is met. However, not marked complete as a WeatherManager will act as an intermediary for weather queries
Meet objective to be editor friendly	N	

Tests

Test	Pass (Y/N)	Notes
Generated noise is deterministic with seed allowing changes to initial state	Y	This was tested with the noise visualisations
Values produced by noise change over time in a believable way	Y	
Noise is deterministic over time (i.e same seed always produces same sequences of values)	Y	
Values changed in custom editors are stored when opening and closing the editor	Y	Fixed the bug that failed this test last sprint

C.1.4 - Sprint 4 Review Document

Sprint Assessment

Sprint Number: 4	**Start:** 23/02/2018	**End:** 01/03/2018
Axosoft Stage Reference: https://samlord.axosoft.com *V1.0/Stage 4*		

Goal Completion

Goals	Time Allocated (hrs)	Time Used (hrs)	Complete (Y/N)
Research unity curve system	4	4	Y
Implement WeatherEvent hierarchy	32	32	Y
Implement Weather transitions with lerping intensity	4	4	Y
Develop “weather stations” to display weather at various location	2	1.5	Y

Total hours rolled over into next sprint: 0 hrs

Objectives

Objective	Achieved (Y/N)	Notes
Meet objective to be procedural	Y	With the addition of Weatherevents, the procedural noise is now driving the selection and transition of weather events
Meet objective to be extensible	N	
Meet objective to be queryable	N	
Meet objective to be editor friendly	N	

Tests

Test	Pass (Y/N)	Notes
Intensity gradually transitions between values during transitions	Y	
Weather station values update over the course of weather changes	Y	
Intensity values filter down to the lowest hierarchy members	Y	
Intensity changes during transitions are affected by changes to the curve	Y	

Additional Comments:

C.1.5 - Sprint 5 Review Document

Sprint Assessment

Sprint Number: 5	**Start:** 02/03/2018	**End:** 08/03/2018
Axosoft Stage Reference: https://samlord.axosoft.com *V1.0/Stage 5*		

Goal Completion

Goals	Time Allocated (hrs)	Time Used (hrs)	Complete (Y/N)
Implement weather properties as a child in the Weatherevent hierarchy that are also driven by intensity values	8	15	Y
Test potential demo controllers and scenery	4	8	Y
Create a particle effect to be later driven by a weather property	4	3.5	Y
Add WeatherEvent editor for curve application via weather properties	4	2	Y

Total hours rolled over into next sprint: 0 hrs

Objectives

Objective	Achieved (Y/N)	Notes
Meet objective to be procedural	Y	
Meet objective to be extensible	Y	The addition of weather properties and weather events means that extensibility should be possible/reasonable from this point forward
Meet objective to be queryable	N	
Meet objective to be editor friendly	N	

Tests

Test	Pass (Y/N)	Notes
Weather properties should correctly be found at runtime by intensity monobehaviours	Y	
Weatherevent editor allows curves to be applied on a weather-properties basis	Y	
Weather event transition intensity is fed down to new lowest level of hierarchy	Y	

Additional Comments:

Sprint Assessment

Sprint Number: 6	**Start:** 09/03/2018	**End:** 15/03/2018
Axosoft Stage Reference: https://samlord.axosoft.com *V1.0/Stage 6*		

Goal Completion

Goals	Time Allocated (hrs)	Time Used (hrs)	Complete (Y/N)
Manual weather mode implementation	16	25	N
Add IntensityDrivenComponents for particle effects, shaders and audio	8	7.5	N
Add shaders, particle effects and audio to be driven by new components	8	8	N
Reverse lookup of weather and humidity values at a given point	16	2	N

Total hours rolled over into next sprint: 16 hrs

Objectives

Objective	Achieved (Y/N)	Notes
Meet objective to be procedural	Y	
Meet objective to be extensible	Y	Structure of new intensity components further this requirement
Meet objective to be queryable	N	Requirement furthered through the inclusion of manual weather intensities being queryable
Meet objective to be editor friendly	N	

Tests

Test	Pass (Y/N)	Notes
Manual weather timing matches that set in the editor	Y	
Adding new manual weather should not affect existing manual weather events	Y	
Intensity of manual weather events can be set using curves in the editor to vary over time	Y	
Intensity of manual weather should be queryable in the same manner as procedural weather	Y	
IntensityDrivenComponents vary the values of their respective objects correctly	Y	These could be further worked on to be more realistic but for testing are perfectly fine

Additional Comments:

Sprint Assessment

Sprint Number: 7	**Start:** 16/03/2018	**End:** 22/03/2018
Axosoft Stage Reference: https://samlord.axosoft.com *V1.0/Stage 7*		

Goal Completion

Goals	Time Allocated (hrs)	Time Used (hrs)	Complete (Y/N)
Manual weather mode implementation	5	2	Y
Add IntensityDrivenComponents for particle effects, shaders and audio	2	1	Y
Add shaders, particle effects and audio to be driven by new components	2	1.5	Y
Reverse lookup of weather and humidity values at a given point	14	2	N
Stages write up for final report	4	3	N
Legal section research for final report	2	1	Y

Total hours rolled over into next sprint: 10 hrs

Objectives

Objective	Achieved (Y/N)	Notes
Meet objective to be procedural	Y	
Meet objective to be extensible	Y	Structure of new intensity components further this requirement
Meet objective to be queryable	N	Requirement furthered through the inclusion of manual weather intensities being queryable
Meet objective to be editor friendly	N	

Tests

Test	Pass (Y/N)	Notes
Manual weather timing matches that set in the editor	Y	
Adding new manual weather should not affect existing manual weather events	Y	
Intensity of manual weather events can be set using curves in the editor to vary over time	Y	
Intensity of manual weather should be queryable in the same manner as procedural weather	Y	

Additional Comments:

C.1.8 - Sprint 8 Review Document

Sprint Assessment

Sprint Number: 8	**Start:** 23/03/2018	**End:** 29/03/2018
Axosoft Stage Reference: https://samlord.axosoft.com *V1.0/Stage 8*		

Goal Completion

Goals	Time Allocated (hrs)	Time Used (hrs)	Complete (Y/N)
Reverse lookup of weather and humidity values at a given point	14	8	Y
Stages write up for final report	1	1	Y
Tree generation	4	6	Y
Time Extension	4	3	Y
Consolidate editor functionality to parent class	2	3	Y

Total hours rolled over into next sprint: 0 hrs

Objectives

Objective	Achieved (Y/N)	Notes
Meet objective to be procedural	Y	
Meet objective to be extensible	Y	
Meet objective to be queryable	N	
Meet objective to be editor friendly	Y	OO-compliant structure of editor scripts and bug fixes

Tests

Test	Pass (Y/N)	Notes
Editor scripts functionality should be unchanged	Y	
Weather functionality should be unchanged in relation to time extension implementation	Y	
Reverse lookup of values in the DoubleDictionary should behave in the same way as a normal Dictionary	Y	Tested retrieving existant and non existant values with expected results.

Additional Comments:

Sprint Assessment

Sprint Number: 9	**Start:** 30/04/2018	**End:** 12/04/2018
Axosoft Stage Reference: https://samlord.axosoft.com *V1.0/Stage 9*		

Goal Completion

Goals	Time Allocated (hrs)	Time Used (hrs)	Complete (Y/N)
Weather transition callback delegates	4	4	Y
More data passed through intensity hierarchy	16	14.5	Y
Snow particle system	4	1.5	Y
Snow shader	8	7.5	Y
Conditional intensity driven behaviours using new hierarchy structure	4	3.5	~Y

Total hours rolled over into next sprint: 0 hrs

Objectives

Objective	Achieved (Y/N)	Notes
Meet objective to be procedural	Y	
Meet objective to be extensible	Y	
Meet objective to be queryable	Y	Call back delegates alongside previously implemented methods on Weather manager
Meet objective to be editor friendly	Y	

Tests

Test	Pass (Y/N)	Notes
Callback delegates pass through correct values	Y	
Callback delegates do not cause null ref exceptions when no callbacks registered	Y	
Example “Weather Announcer” script using delegates to test	Y	
Intensity data hierarchy changes results in same behaviour as previously	Y	
Conditional behaviour implementation (snow/rain) using new data only occurs when condition met	N	Largely this works, some restructuring needs to be done to ensure that disabling works as expected. May also affect other behaviours - more testing required
Snow particle effect and	Y	

shader controllable by intensity driven behaviour		

Additional Comments:

Sprint Assessment

Sprint Number: 10	**Start:** 13/04/2018	**End:** 03/05/2018
Axosoft Stage Reference: https://samlord.axosoft.com *V1.0/Stage 10*		

Goal Completion

Goals	Time Allocated (hrs)	Time Used (hrs)	Complete (Y/N)
Fix snapping weather bug	8	32	Y
Update hierarchy curves	4	3.5	Y
Conditional intensity driven behaviours disable behaviour fix	4	4	Y

Total hours rolled over into next sprint: 0 hrs

Objectives

Objective	Achieved (Y/N)	Notes
Meet objective to be procedural	Y	
Meet objective to be extensible	Y	
Meet objective to be queryable	Y	
Meet objective to be editor friendly	Y	

Tests

Test	Pass (Y/N)	Notes
Weather transitions should be gradual	Y	
Previously active weather should reactivate in a non-sudden way	Y	
Weather curves should result in believable weather visualisations	Y	
Conditional behaviour implementation disable should result in the behaviour fading out to zero	Y	

Additional Comments:

Sprint Assessment

Sprint Number: 11	**Start:** 04/05/2018	**End:** 17/05/2018
Axosoft Stage Reference: https://samlord.axosoft.com *V1.0/Stage 11*		

Goal Completion

Goals	Time Allocated (hrs)	Time Used (hrs)	Complete (Y/N)
Instance events implementation - use noise generator to ensure deterministic	4	3	Y
Further refinement of snapping issue	8	9	Y
Volumetric lighting snapping fix	8	6	Y
Visual and audio for all weather events	16	14	Y
Integration with Unity WindZones	8	7	Y
Queryability of wind and intensity values	2	1	Y
Weather info UI display	2	1	Y
Seed setting through weather manager	2	0.5	Y
Update documentation	4	2	Y

Total hours rolled over into next sprint: 0 hrs

Objectives

Objective	Achieved (Y/N)	Notes
Meet objective to be procedural	Y	
Meet objective to be extensible	Y	
Meet objective to be queryable	Y	
Meet objective to be editor friendly	Y	

Tests

Test	Pass (Y/N)	Notes
Gradual transitions of all weather events	Y	
No snapping for volumetric light controller	Y	
All weather events have audio and visual representation	Y	
Wind zone direction and intensity matches generated wind value	Y	
Wind and intensity values available through WeatherManager	Y	
Random and set seeds through weather manager	Y	
Live weather data shown on screen as text	Y	
All classes and methods XML	Y	Could be extended with

commented		namespace XML documentation

Additional Comments:

Appendix D - Requirements

D.1 - Weather System Requirements Document

Weather System Requirements

Target Release	1.0
Document Status	Draft
Product Owner	Sam Lord
QA	See testing design document

Background and Strategic Fit

This project aims to offer an open-source alternative to the complex and expensive or the cheap but overly simplified weather systems for Unity. The project will be made public from a github repo once the target release has been met.

Requirements

#	User story title	User story description	Priority	Notes
1	Customisable weather assets	As a developer I want to be able to customise the visual elements of the weather system to match my game	Must	Drag-and-drop assets in the editor
2	Believable weather	As a player I want immersive, believable weather in the games I play	Must	Procedurally generated weather will achieve this
3	Extensible package	As a developer I want to customise and extend the packages I use	Must	OO Design and good documentation
4	Queryable weather	As a developer I want the game worlds I create to react to the weather that is occurring and therefore want to be able to query various weather factors at specific locations	Must	The procedural algorithm should take location as an input to allow location-based related weather
5	Accessible package	As a developer I want a weather system which is free to use and distribute in my game	Must	MIT licence meets these requirements

Not Delivering

- Photorealistic assets within the package
- Simulation-level realism

D.2 - Weather System Requirements and Deliverables Met Document

Weather System Requirements and Deliverables Met

Target Release	1.0
Document Status	Draft
Product Owner	Sam Lord
QA	See testing design document

Background and Strategic Fit

This project aims to offer an open-source alternative to the complex and expensive or the cheap but overly simplified weather systems for Unity. The project will be made public from a github repo once the target release has been met.

Requirements

- ✓ Be free, open-source and reusable without limitation
- ✓ Be procedural
  - ✓ Will provide a platform from which weather patterns can be new and interesting on each playthrough of any game using the system.
  - ✓ Will be seeded and therefore will generate reproducible weather patterns at the discretion of the developer, and enable easier testing.
  - ✓ Weather will vary across the game world at any given moment, adding more realism. This is in contrast to the researched products which unanimously had ubiquitous weather throughout the game world.
- ✓ Be extensible
  - ✓ Will be designed so as to be extensible by developers using the platform.
  - ✓ Will be well documented so as to be easy to use for other developers.
- ✓ Be queryable
  - ✓ Will allow developers to request information about the weather at specific locations and times. This allow for games which use information such as temperature to, for example, change the look of the player or environment in different conditions or provide information to survival-like games.
- ✓ Be Unity Editor friendly
  - ✓ Changes to initialization values will not require editing code and will be available as sliders/editable text boxes in the Editor.
  - ✓ As and when required, custom editors may be written to automate repetitive tasks.

Deliverables

- ✓ The following weather processes will be included in the platform: Clear sky, rain, storms, snow, overcast and integration with Unity's wind zones. The weather system will be

produced in C# as a package for Unity Game Engine. The weather system will control particle and audio effects alongside simple shaders and lighting.

- ✓ The product will not be a simulation, however it will take inspiration from weather simulation and historical data. For example, historical weather data may be analysed to determine the chance of one type of weather leading to another and this may inform the balancing of the system. This will enable an immersive experience to players through realistic transitions between weather states.
- ✓ The product will be cross platform (Windows/OSX) compatible, both for editing/development and gameplay.
- ✓ An example demo implementation utilising the package will be produced in Unity Game Engine (Unity, 2018) for Windows. The demo will allow a first person perspective view of a game world that has the produced weather package integrated, showing weather information at the players current position on screen whilst visual weather effects can also be seen.

Appendix E - Weather Data

E.1 - All Weather Data

Available online at
https://docs.google.com/spreadsheets/d/101wARLrJ-Nrzy9eb3_HxxPacZr8-_j3YiMVtAjFCViU/edit?usp=sharing

E.2 - Procedurally Generated Weather Data Summary Tables

E.2.1 - Procedurally Generated Weather, Wind Strength 1

Weather Type	Count	Percentage
Overcast	22	61.11111111
Rain	4	11.11111111
Clear	3	8.333333333
Snow	0	0
Storm	2	5.555555556
Hail	5	13.88888889

E.2.2 - Procedurally Generated Weather, Wind Strength 5

Weather Type	Count	Percentage
Overcast	30	48.38709677
Rain	5	8.064516129
Clear	8	12.90322581
Snow	1	1.612903226
Storm	1	1.612903226
Hail	17	27.41935484

E.2.3 - Procedurally Generated Weather, Wind Strength 10

Weather Type	Count	Percentage
Overcast	44	34.92063492
Rain	29	23.01587302

Clear	20	15.87301587
Snow	12	9.523809524
Storm	2	1.587301587
Hail	19	15.07936508

E.2.4 - Procedurally Generated Weather, Wind Strength 10

Weather Type	Count	Percentage
Overcast	44	34.92063492
Rain	29	23.01587302
Clear	20	15.87301587
Snow	12	9.523809524
Storm	2	1.587301587
Hail	19	15.07936508

E.2.5 - Real-World Weather

Weather Type	Count	Percentage
Cloud	646	32.3
Rain	459	22.95
Fair	656	32.8
Snow	15	0.75
Storm	1	0.05
Hail	0	0

E.2.6 - Weather Comparison Table

Weather Type	Real World Data	Simulation (Wind strength 1)	Simulation (Wind strength 5)	Simulation (Wind strength 10)	Simulation Average	Difference
Cloud	32.3	61.11111111	48.38709677	34.92063492	48.1	15.8
Rain	22.95	11.11111111	8.064516129	23.01587302	14.1	8.9
Fair	32.8	8.333333333	12.90322581	15.87301587	12.4	20.4
Snow	0.75	0	1.612903226	9.523809524	3.7	3.0
Storm	0.05	5.555555556	1.612903226	1.587301587	2.9	2.9
Hail	0	13.88888889	27.41935484	15.07936508	18.8	18.8
Other	11.15	0	0	0	0.0	30.2

Appendix F - Usability Questionnaire Data

F.1 - Responses Summary Table

Respondent	How navigable do you find the documentation?	How queryable is the WeatherManager in your opinion?	How believable is the produced weather?	To what extent is the system procedural in your opinion?	How extensible in the system in your opinion?	How editor-friendly is the system?
1	9	8	7	10	8	7
2	7	7	8	10	8	7
2	5	7	6	10	8	5
3	6	6	8	7	5	7
4	7	7	4	8	6	9
Average	6.8	7	6.6	9	7	7
Percentage	68.00%	70.00%	66.00%	90.00%	70.00%	70.00%

Appendix G - Best Practice Documents

G.1 - Coding Best Practice Document

Coding Best Practices

PascalCase
For *methods*, *functions*, *classes* and *namespaces*.

```
static void Main(string[] args)
{
    Console.WriteLine(SumOfTwoInts(5, 7));
}

private int SumOfTwoInts(int a, int b)
{
    return a + b;
}
```

camelCase
For variables.

Namespaces
Use them! Sensible naming, PascalCase.

If statements
Single line if statements must use '{ }'

French/Curly Braces { }
As above in the image, '{ }' brackets should be on a line of their own - this applies for classes, methods, for loops, while loops, foreach loops etc

OO Techniques
Basically - if you can abstract something that is *likely* to be used in a similar way elsewhere, abstract it.

Encapsulate classes. This provides specific entry points for classes and protects the inner workings (including data) of a class.

Design
Do not dive straight into coding a large system - take some time to *at least* do a class diagram and think through how the system will interact with other systems. See *OO Techniques*.

Class diagrams should be annotated where appropriate.

Descriptive variable names
Refrain from using one character variable names with the exception of variables used to iterate through a collection (i = 0 etc). Variable names should adequately describe their purpose.

Documentation

All classes and methods should have XML-documentation (type '///' above what you're documenting). Also, any code that may be unclear to others, does something especially complicated or would benefit from a comment(s) - comment it! See below for example (note the difference in readability between up to line 18 and after). Also see variable name specification.

```
using System;
namespace OperatorsAppl
{
    /// <summary>
    /// Program to demonstrate bitwise operations and good commenting practices
    /// </summary>
   class Program
   {
        /// <summary>
        /// Program entry points
        /// </summary>
        /// <param name="args">Arguments passed through if started from command line</param>
      static void Main(string[] args)
      {
         int a = 60; // 60 = 0011 1100
         int b = 13; // 13 = 0000 1101
         int c = 0;

         c = a & b; //12 = 0000 1100
         Console.WriteLine("Line 1 - Value of c is {0}", c );

         c = a | b; //61 = 0011 1101
         Console.WriteLine("Line 2 - Value of c is {0}", c);

         c = a ^ b; //49 = 0011 0001
         Console.WriteLine("Line 3 - Value of c is {0}", c);

         c = ~a;
         Console.WriteLine("Line 4 - Value of c is {0}", c);

         c = a << 2;
         Console.WriteLine("Line 5 - Value of c is {0}", c);

         c = a >> 2;
         Console.WriteLine("Line 6 - Value of c is {0}", c);
         Console.ReadLine();
      }
   }
}
```

See also

Microsoft's best practices (This document you're reading right now takes priority in the case of contradiction).

Appendix H - Licences

H.1 - MIT Licence Template

Copyright (c) <year> <copyright holders>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.