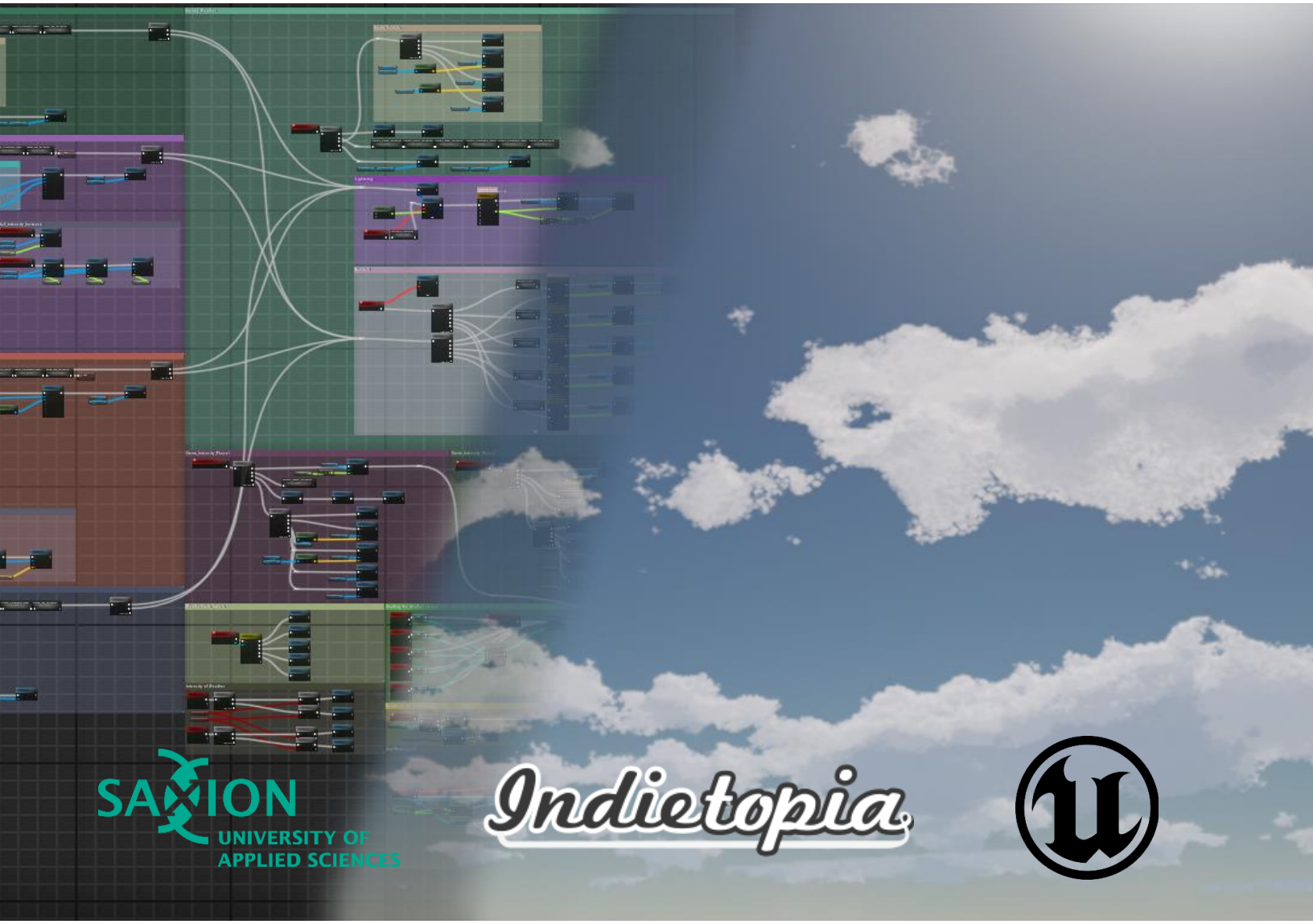
Graduation Report.

How can a dynamic weather system/ day-night cycles be implemented in an already existing VR application, to improve the immersion and complement the artistic view, while maintaining the desired performance capabilities and can be used in other projects, by developers, with minimal changes to reduce workload in the future?



Student: Tudor Ani -474971-

Coordinating Teacher: Iain Douglas

Table of Content:

Glossary of Terms:

Term	Definition
Blueprint	An asset that allows content creators to easily add functionality on top of existing gameplay classes.
Fluid Simulation	A special toolset that helps to mimic the effect of fluids in an environment.
FluidNinja Live	An independent tool developed to provide real-time, responsive fluid simulations.
Material Instance	A way to create a parent Material that you can use as a base to make a variety of different-looking children.
Material Parameter Collection	An asset that stores an arbitrary set of scalar and vector parameters that can be referenced by any material.
Niagara	Niagara is Unreal Engine's next-generation VFX system.
Optimization	A game's ability to run on multiple systems.
Real-Time	A computer system that reacts to events by performing tasks within a specific time interval in the order of milliseconds.
Scalar Parameter	Stores numerical values.
Unreal Engine	3D computer graphics game engine developed by Epic Games.
Vector Parameter	Stores color data RGBA.
VR	A simulated experience that employs pose tracking and 3D near-eye displays to give the user an immersive feel.

(Table 1. Glossary of Terms.)

Abstract:

In this report, the researcher details the development of a dynamic weather system with a day and night cycle, seamlessly integrating it into an existing Virtual Reality (VR) application crafted by Indietopia for a local museum.

To begin, the researcher delved into an exploration of various research methodologies, such as desk research and interviews, to achieve the defined goal, examining shaders, Niagara systems, and a third-party software—FluidNinja Live. After thorough consideration, the researcher determined that FluidNinja Live emerged as the optimal solution, consolidating all essential functionalities required for the system's creation in a unified framework.

Subsequently, the report outlines the process of implementing this system within the Unreal Blueprint system. The researcher elaborates on encountered challenges, detailing how each obstacle was successfully navigated, and provides a comprehensive walkthrough of specific steps crucial in the development process.

Moreover, the research includes a robust testing phase, wherein the developed system undergoes evaluation with multiple end users. This meticulous testing ensures that the system performs at the desired level of efficiency, aligning with the specified performance criteria.

In conclusion, FluidNinja Live proved to be the right tool for aiding the developer to deliver a complex final product. The final product is a dynamic weather state system with five different built-in states. The system is optimized for VR use and testing with other developers proved its effectiveness in this project and its help in future ones by cutting down development time.

1. Introduction:

1.1 Company:

Indietopia is a prominent non-profit foundation and award-winning studio with a mission to support aspiring game developers. They operate from Groningen, Netherlands, while striving for global recognition through social media, a game store, and participation in events. Additionally, they offer workshops on various aspects of game design and related topics. (*About - Indietopia*, 2023).

Today, Indietopia offers a diverse range of services encompassing development and publishing. A significant focus of their projects lies in the realm of 3D scans and virtual reality (VR) applications. At present, they are actively engaged in the development of a VR application designed for a local museum, with my own involvement. This immersive project is centered around the celebrated local musician Ede Staal, with each level seamlessly integrating one of his well-known songs.

1.2 Project:

The current project undertaken by the team involves the development of a VR application intended for installation within a local museum, as part of an exhibition dedicated to the local musician, Ede Staal. This immersive application comprises multiple levels, each intricately woven around a specific, well-known song associated with the artist. The primary objective of this project is to provide visitors with a profound connection to the life of Ede Staal.

Named "Explore Ede's World," this virtual reality experience offers users a nonlinear, exploratory adventure, allowing them to step into Ede Staal's world and delve into his memories, the surrounding environment, and his life achievements through the portrayal of portraits and paintings found within his residence. Central to this experience is Ede's house, serving as a pivotal hub where users can continually return. Within this hub are the paintings and portraits that serve as gateways to other levels within the experience.

The four distinct levels within the experience — "Ede Staal's House" symbolizing life, "Ede's English Classroom" representing language, "The RTV Noord Station" also representing language, and "The Grasslands" embodying the landscape — are deeply rooted in the themes of life, language, and the environment, all of which played significant roles in Ede Staal's life and work.

This captivating experience is designed for use with a virtual reality headset, allowing users to immerse themselves while seated. Simultaneously, the content is projected onto a large screen in the exhibit room, enabling other museum visitors to witness and engage with what the VR user is experiencing in real-time. (Source: "Explore Ede's World" Game Design Document, May 2022)

1.2.1 Problem:

In each level, player emotions are dynamically evoked primarily through the interplay of background music and the storyline. To enhance the immersion factor, it's crucial to align these emotional elements with the level's atmosphere. A pivotal aspect to be developed is a dynamic weather system capable of seamless transitions. This system should offer a range of settings to empower developers to effortlessly alter the atmosphere at predefined intervals. Key settings to be implemented include rainy, stormy, and

sunny conditions, while also considering secondary elements like lightning and rainfall. Additionally, the development must encompass dynamic water bodies, such as seas, lakes, or rivers, varying in scale, to harmonize with other atmospheric elements.

The challenge in this VR application lies in optimizing performance, as these systems can be resource intensive. While Unreal Engine offers multiple options for creating such systems, they can tax system resources significantly. Developers face the task of balancing the need for pleasing volumetric effects to ensure an enjoyable visual experience for users.

1.3 Research Questions:

1.3.1 Main Question:

How can a dynamic weather system/ day-night cycles be implemented in an already existing VR application, to improve the immersion and complement the artistic view, while maintaining the desired performance capabilities and can be used in other projects, by developers, with minimal changes to reduce workload in the future?

1.3.2 Sub Question:

1. What are the desired capabilities of the weather system?
2. How well-optimized does it need to be in order to be supported by VR?
3. What tools can be used to complement the atmosphere to improve immersion?

1.4 Blueprints:

Unreal Engine's Blueprint Visual Scripting is a robust gameplay scripting system that uses a node-based interface in Unreal Editor to create objects and classes. It bridges the gap between designers and programmers, offering designers access to powerful tools and enabling programmers to create foundational systems that designers can build upon. (Unreal Engine Documentation, *Blueprints Visual Scripting*, n.d.) Developers will employ this system to dynamically adjust various element parameters. This approach is favored because it allows for parameter changes to respond to in-game events, ensuring a tailored experience based on the player's progression.

1.5 Scope:

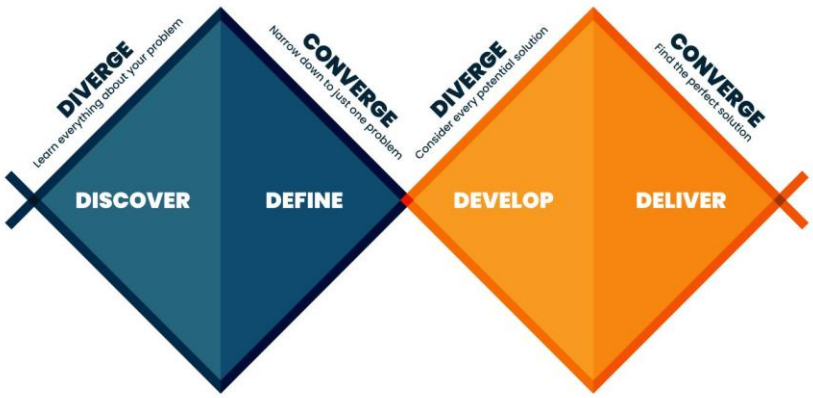
Determining the project's scope has been challenging, primarily because I lack experience in the field I'll be working in and with the tools I'll use. The complexity of the Blueprints topic further complicates estimating task durations. Balancing this, I must allocate sufficient time for optimization, knowing that debugging will be time-intensive during production.

Taking a realistic view, I believe I'll have enough time during this project to develop the core dynamic system, encompassing light settings, a day/night cycle, and dynamic weather (ranging from clear skies to stormy conditions). Additionally, I'll tackle secondary systems like dynamic, interactive water bodies and volumetric fog.

2. Design Process:

2.1 Double Diamond:

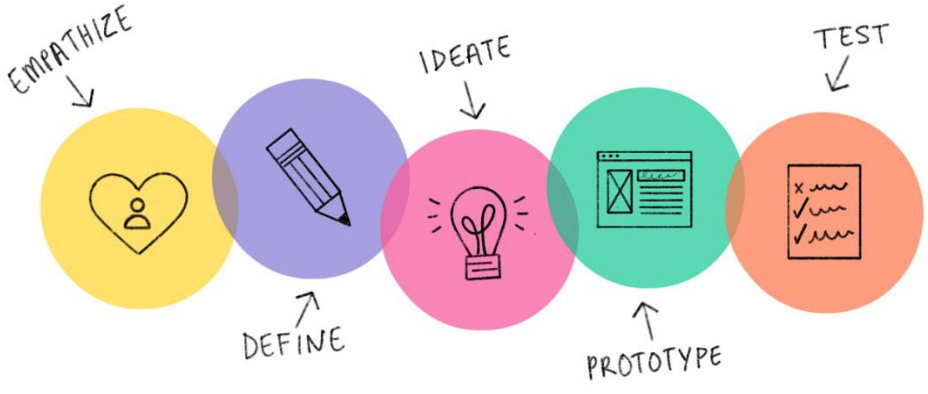
The Double Diamond design process is a structured framework for problem-solving and innovation, consisting of four phases: Discover, Define, Develop, and Deliver. In the "Discover" phase, designers research and understand user needs and context. "Define" narrows down the problem, "Develop" involves generating and prototyping solutions, and "Deliver" refines and tests the chosen solution. This iterative process ensures user-centric, effective solutions. (LogRocket Blog, V. Brown, 2023)



(Figure 1. Double Diamond visual representation. (Google Images))

2.2 Design Thinking:

Design thinking is a human-centered problem-solving approach with five stages: Empathize, Define, Ideate, Prototype, and Test. It focuses on understanding the user's needs, defining the problem, generating creative ideas, building prototypes, and testing solutions to create user-centric and innovative designs. (*What Is Design Thinking & Why Is It Important? | HBS Online*, 2022)



(Figure 2. Design Thinking visual representation. (Google Images))

I've determined that opting for the double diamond method is the more suitable choice in my situation. Several factors underlie this decision.

To begin with, the double diamond method aligns with the design process adopted by the team and the company for the entire project. By selecting this approach, I can enhance my collaboration with the team throughout the various project phases.

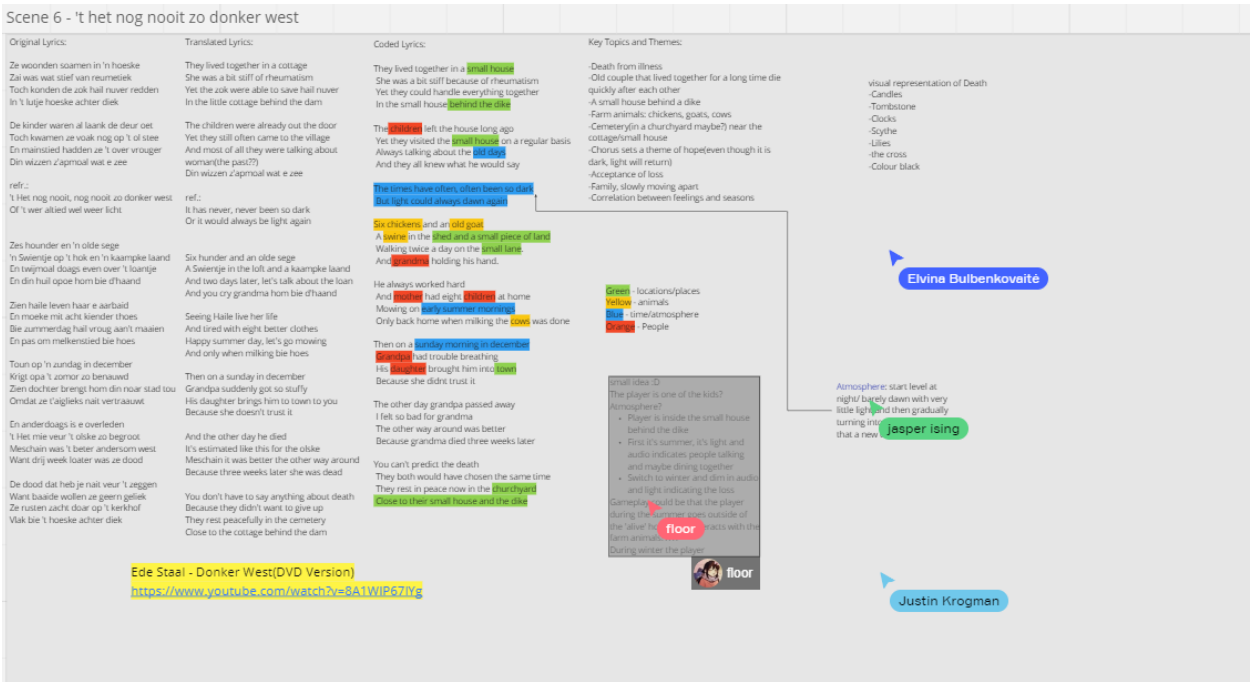
Furthermore, in this context, where I need to continuously experiment with different methods, assess them, and subsequently fine-tune my approach, the double diamond method affords me greater flexibility. This is particularly advantageous because I frequently find myself diverging and converging throughout all project stages.

2.3 Discover:

During this phase, I will explore various options related to the weather system and determine the most suitable solution for my scenario. The foundation of the project and its functionalities will be built upon the Unreal Blueprint system. As mentioned in the Blueprint section of the introduction(Blueprints), this system provides the dynamic adjustability essential for seamless implementation and grants developers complete freedom, even in future projects. To learn what the requirements and capabilities of this system should be I conducted interviews with the more experimented developers Kristiyan Petrov and Justin Krogman. I chose to talk with them because they have worked on this project from the beginning and know its needs and their experience adds verticity to the reason why certain decisions have been made.

In this interview with Kristiyan regarding the system's requirements, it became evident that using blueprints is crucial to enable communication with other pre-existing systems. You can read more in the Appendix section: Interview. The ability to invoke custom events and reference them across multiple blueprints allows for flexible adjustments to the logic or the addition of new elements. This reinforces the decision to adopt this method in Unreal for our project.

Next, I assessed the system's capabilities to align them with both the company's and the client's needs. Discussions with Merijn de Boer and Kristiyan, along with a meeting with the museum's board head for which we are developing the VR application, provided valuable insights into the system's requirements. Collaborating with individuals experienced in working with the museum and its target audience allowed me to understand their expectations and challenges. The primary goal is to enhance immersion and artistic fidelity in the levels, aligning harmoniously with the museum's objectives. With this in mind, to define the system's capabilities, I collaborated with the team during the design phase of the new level and reviewed designs for existing levels. I identified key aspects applicable to all levels, such as dynamic volumetric clouds, day and night cycles, dynamic water interaction, and changing material parameters. Additionally, I outlined level-specific elements, including distant lightning, rainfall, particle systems (e.g., snow, rain), and volumetric fog. These elements were linked to individual levels and their storylines. (Figure 3.)



(Figure 3. Miro Board design process of one of the levels I took part in. Underlying weather elements to tailor the dynamic system to the story of the level.)

Going over my outlined elements together with Justin and Kristiyan, both agreed that these elements complement the story of each level. The system, constructed in this manner, not only meets the current project's requirements but also provides the desired flexibility for future endeavors.

In the subsequent phase, I sought out tools to achieve my goal, conducting extensive research and consulting fellow developers for advice. After online exploration, I considered three potential solutions: relying solely on material shaders, incorporating Niagara, and utilizing a third-party plug-in, FluidNinja Live, constructed by a third-party developer, recommended by Kristiyan Petrov.

My initial thorough exploration focused on the use of shaders, acknowledging their potency and versatility in Unreal. Shaders, being a fundamental component of any engine, offer flexibility and performance efficiency when implemented correctly. To assess their suitability for the entire system, I delved into online tutorials, drawing insights from various resources. Notable among these were Ben Cloward's comprehensive water shader playlist (Ben Cloward, https://www.youtube.com/playlist?list=PL78XDi0TS4lGXKflD2Z5aY2sLuIln6-sD), PrismaticaDev's detailed explanation of the "SingleLayerWater" node (PrismaticaDev, https://www.youtube.com/watch?v=J2Qf5v9_uSY), and Michel Kinsey's insightful guide on volumetric clouds (The DiNusty Empire, https://www.youtube.com/watch?v=oWE_vYiQMoE). Additionally, William Foucher's video provided valuable insights into techniques for setting up and art directing volumetric clouds (William Faucher, https://www.youtube.com/watch?v=yolGEIrhu0s).

These resources, combined with Unreal Engine's official documentation, equipped me with a thorough understanding of the capabilities and limitations of using shaders. The key takeaways highlighted:

Nr.	Key Takeaway
1.	Shaders are the engine's backbone and will be used regardless of the system's development approach.
2.	They offer flexibility and performance benefits when implemented correctly.
3.	Limitations exist in dynamic interactivity with other objects, providing only limited options.
4.	The complexity of the shader increases significantly when adding all required elements, making future additions or changes complicated.
5.	Relying solely on shaders for adding depth is limited and encounters the same complexity issue.
6.	Clouds lack interaction with other objects without an additional driving system, such as fluid simulation.

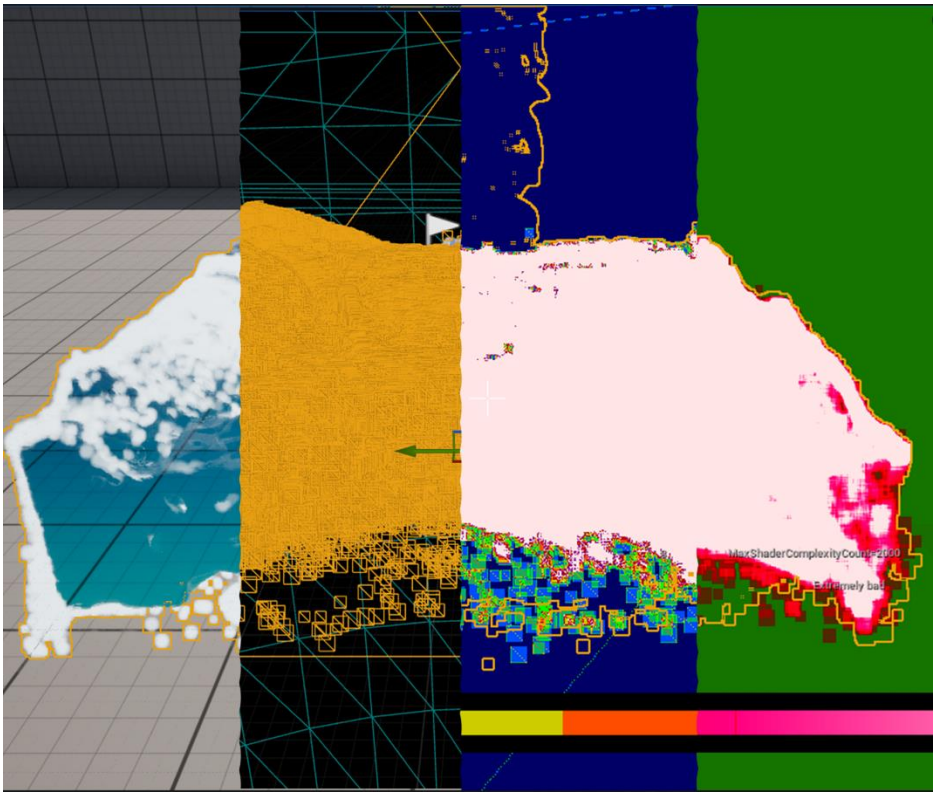
(Table 2. Shaders key takeaways.)

Recognizing these key takeaways, it became evident that relying solely on shaders would not suffice to meet all predefined requirements for this dynamic system. Acknowledging this, I recognized the necessity of integrating an alternative solution in conjunction with shaders to achieve optimal results.

Based on these conclusions, I initiated research into Niagara systems and their applicability in my case. Niagara, Unreal's next-generation VFX system, empowers technical artists to independently add elements while providing maximum flexibility. This system comprises four key elements: Systems, Emitters, Modules, and Parameters, seamlessly combining to produce intricate visual effects (Niagara Overview. (n.d.). Unreal Engine 4.27 Documentation).

Despite my efforts, I encountered challenges in finding examples of people using Niagara for scenarios akin to this project. Further investigation revealed that the Niagara fluid simulation is exclusively available in UE5 as a beta version plugin (Niagara Fluids Quick Start Guide. (n.d.). Unreal Engine 5.3 Documentation). However, this system, still in its infancy, proves to be highly performance-demanding. To illustrate, I created a test scene in a UE5.3 project featuring a default Niagara 3D simulation, utilizing the built-in optimization view mode option (Figure 4). The results confirmed the system's complexity, even at a small scale, rendering it impractical for large-scale options required by the team in this project. It is important to understand what all these Optimization viewports show, as I will refer to them further in the report multiple times. The first one, lit mode, shows the developers how the scene will look like in the end. It combines all the information provided like color, roughness, reflections, and more, into the final render, which the players will see. The second one, brush wireframe, shows the topology of the objects displayed on the screen. This is useful to developers to see if all the meshes have properly optimized topology. Here we can see that even though the mesh itself is only a plane there are a lot of them which add up to a large number of triangles. The third one, quad overdraw, shows the developers what parts of the objects are being rendered on top of each other. This can be heavy on the performance because the engine needs to calculate all those objects in a small area which can slow down the system. To visualize it easier the engine shows objects colored from green to white: in green there is no overlap - in white there is a lot of overlapping. The last viewport, shader complexity, shows how complex the shader(material) of meshes is. This shows how difficult is for the engine to compute that material. When the engine deals with transparency then the shader is the most complex as the

engine needs to render what is behind as well. To visualize it the engine shows the objects colored from green to white: in green the shader is not complex at all - in white the shader is very complex.



(Figure 4. UE5.3 Viewport using optimization view mode on a 3D Fluid Niagara system. From left to right: Lit Mode/ Brush Wireframe/ Quad Overdraw/ Shader Complexity)

It's crucial to note that the default system lacks interactions with different objects, and adding such functionality would only exacerbate its complexity and performance demands. Even if the team were to migrate the project to the latest Unreal version, this option remains unviable.

Online observations revealed that people commonly utilize the Niagara system for creating particle systems like rain or explosions, functionalities available in UE4.27—the version employed for this project. However, these systems can be resource-intensive, necessitating careful optimization and limited usage in small areas for practical implementation. In the exploration of fluid simulation, I came across the Water plugin native to Unreal (Water. (n.d.). Unreal Engine 4.27 Documentation). While offering various built-in options for water creation, such as oceans, lakes, and rivers, this plugin was introduced in UE4.26 and remains experimental. Its experimental status deems it unsuitable for commercial production due to potential stability issues, which is undesirable when developing an application for a client.

Further analysis by importing the plugin into a test project unveiled its use of complex shaders and materials, echoing the issues associated with shaders. The dense mesh density further disqualifies this plugin as a viable option, despite being a well-developed tool by Unreal developers.

As I mentioned Kristiyan, the project manager, suggested the use of a third-party software, FluidNinja Live. This is a plugin that integrates with the project and offers real-time, interactive fluid simulations tailored for PC and console developers within the Unreal Engine environment. With a straightforward 5-minute setup process, developers can swiftly introduce responsive fluid simulations into their projects. These simulations take the form of Actors, and a Live ActorComponent can be seamlessly incorporated into user-defined Actors to influence other Components like Niagara, VolumeSmoke, and Fog. FluidNinja Live is highly customizable, boasting a system for managing simulations based on presets, and it allows users to edit output materials according to their preferences. (FluidNinja Live Manual, July 2023.)

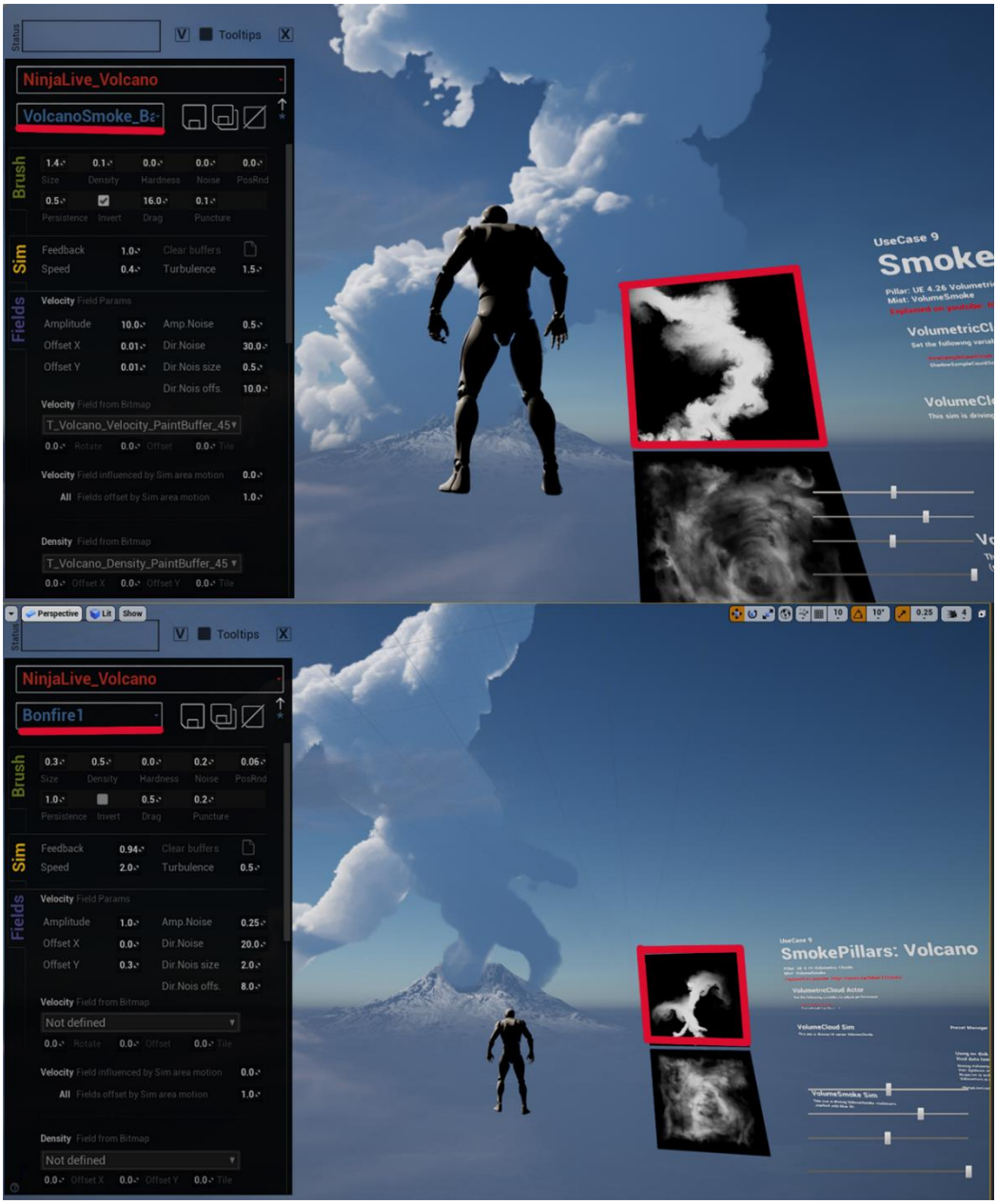
The manual and the documentation provided by the developers were the main source of researching this tool as it provides a deep dive into its all functionalities and capabilities. Together with these, I made use of the showcase videos on the YouTube channel of the developer, Andras Ketzer. (Andras Ketzer, https://www.youtube.com/playlist?list=PLVCUepYV6TvOrOfQVLMCxl_JoU_cIkK8P) In this playlist, he showcases all the methods FluidNinja Live can be used. These resources provided me with a good understanding of this tool capabilities. The key features of this tool are:

Nr.	Key Takeaway.
1.	Large scale: local solver combined with global pattern generators - to render infinite fields of sand, water, fog.
2.	Volumetrics: custom, lit 3D Smoke volume included + driving native UE Fog and Clouds.
3.	Scalable: could be optimized for low-end hardware or run a 4k sim container
4.	Simple-mode: track objects and draw trajectories without running fluidsim (eg. footsteps, wheeltracks).
5.	Modular: could be added as ActorComponent / could drive other components (niagara, volumes).
6.	Niagara: two-way data flow - drive particles using fluidsim / drive fluidsim using particles.
7.	Sim buffers (density, velocity, pressure) exposed to RenderTargets.

(Table 3. FluidNinja Live key takeaways.)

Recognizing these capabilities FluidNinja Live seems to be the right tool for this project. It offers a lot of functionalities under the same hood and all the elements of the dynamic weather system can leverage these functionalities. To illustrate this, I will use the use case levels provided with the plugin.

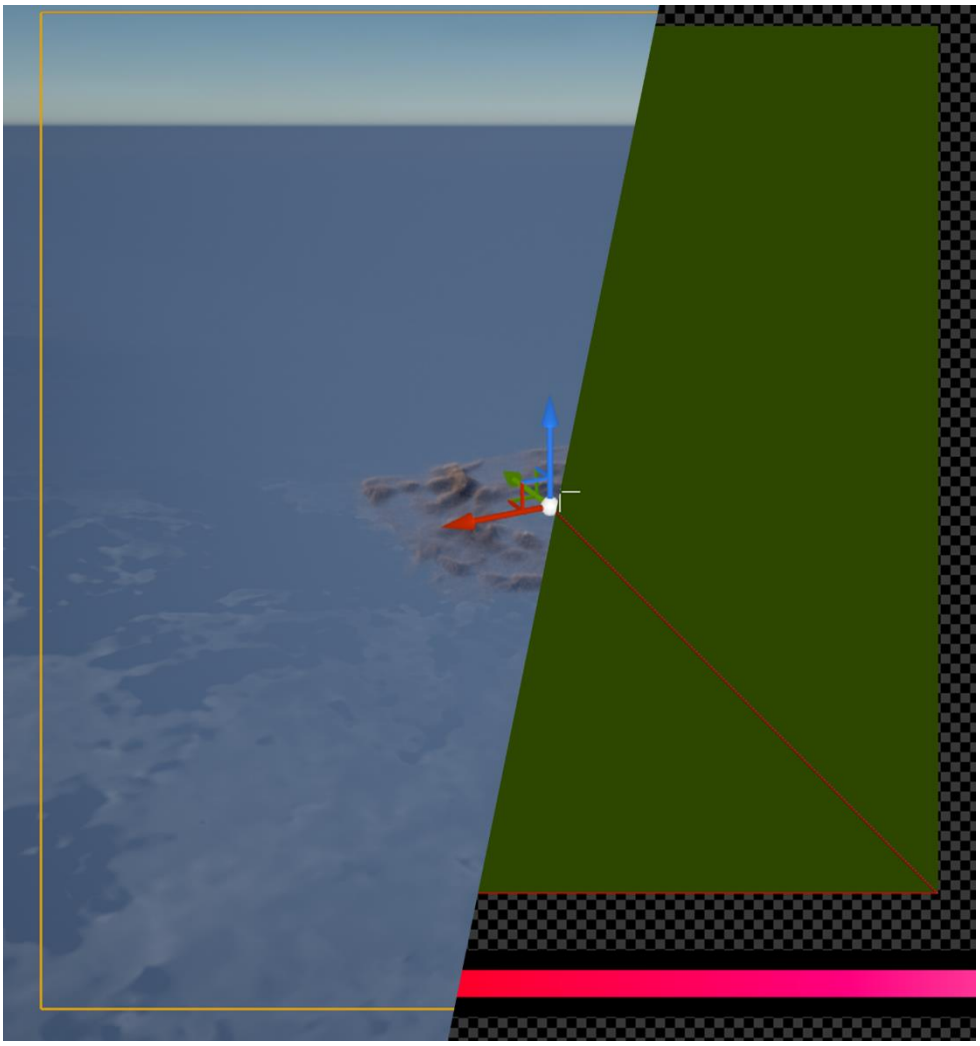
The first scenario I want to show is the use of FluidNinja Live to drive native UE volumetric clouds and exponential height fog. In this example, level multiple features of the plug-in are leveraged to create a pillar of smoke using the volumetric clouds. In this level, it's easy to see that developers can use the Ninja GUI to adjust parameters to get multiple types of looks and effects with the clouds enabling to use of the volumetric clouds in multiple instances to create different types of effects. In this example, they are used to visualize the smoke from a volcano but the same method can be used to create a hurricane cloud vortex or different common cloud coverages. (Figure 5.)



(Figure 5. Different cloud looks using a different preset for the sim and manually painting a different shape.)

This showcases that you have a lot of freedom in art directing these effects. For this example, I just changed the simulation preset (blue underlined text) and drew a different shape on the trace mesh (red box.) The shape that you draw can be saved and used later and have it load as default for that specific case. All these settings can be accessed and edited through blueprints which offers the developers another layer of dynamic control. Here is a list of all the UI parameters that FluidNinja offers: https://drive.google.com/file/d/1jpr3cuNvwVUNc1Fj2nPE_9jyLnh-JVmw/edit.

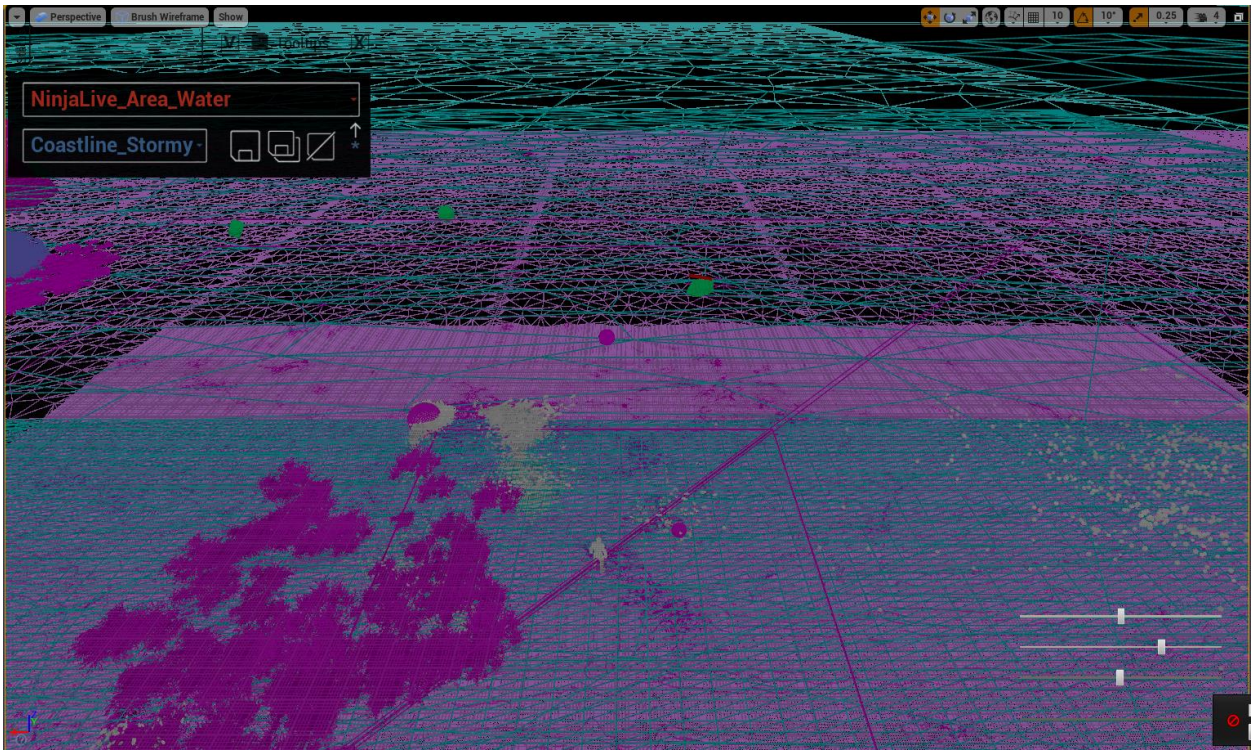
You can use the same method with fog as well, to create a variety of effects. Ninja uses for this a custom volume type labeled as “VolumeSmoke”. This volume employs a camera facing plane. The volume is calculated by a compute shader and the 3D result is mapped back to the scene on the camera facing plane. With this method we can achieve very realistic results while maintain performance cost at a low level. (Figure 6.)



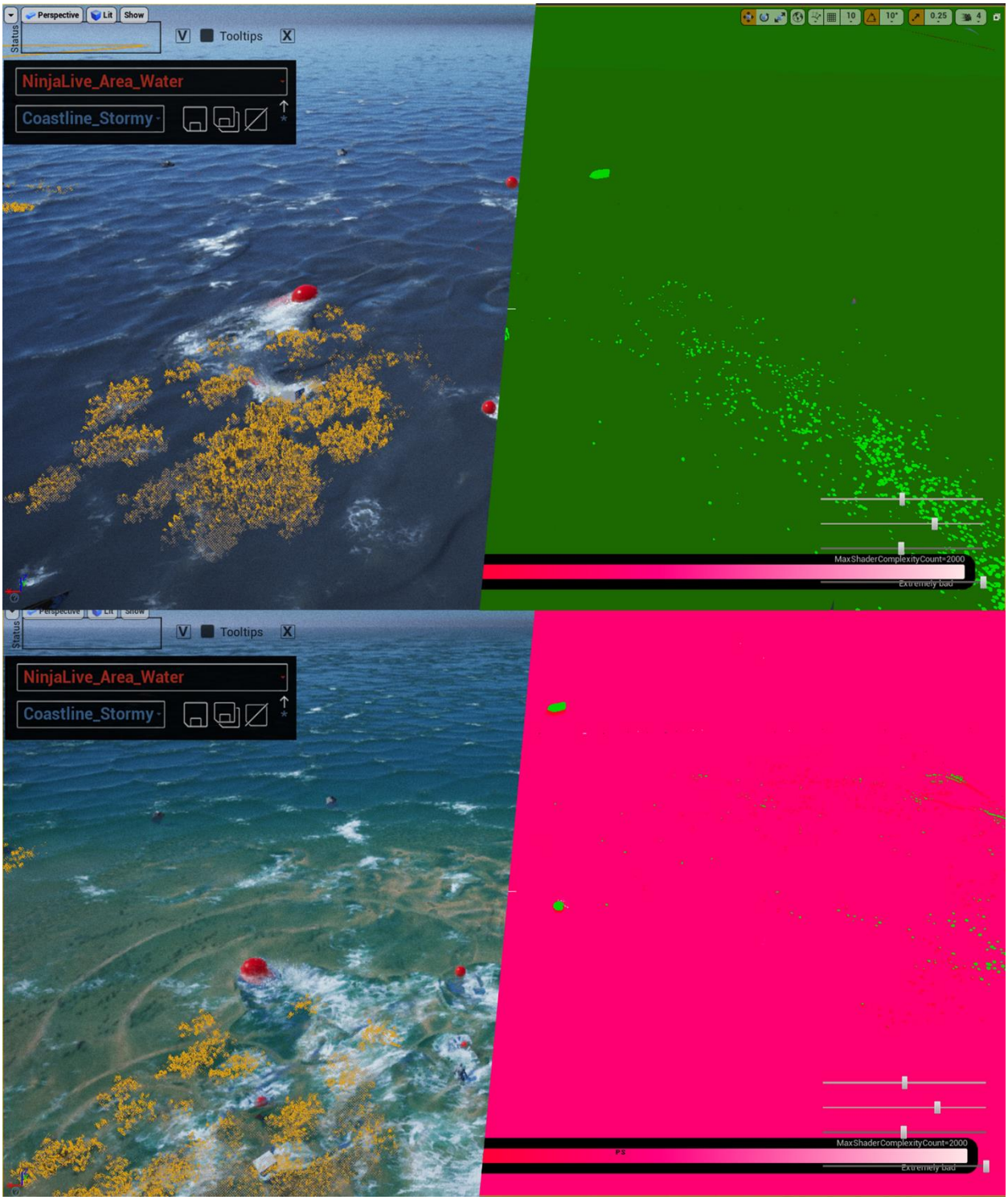
(Figure 6. Volume Smoke driving fog. Left: Lit Mode Right: ShaderComplexity+Quads.)

The next use case I want to showcase is the use of FluidNinja to drive water interaction. To make this system to work you need a FluidNinja Component which determines the area of interaction and the area on which these simulations can be drawn. It also determines the type of object with which it can/should interact and it can be attached to different actors. The next step is to add the material you use and use tags to let Ninja know on which surface it should draw this simulation and which material to use. This method can be used to create large water or small ones like rivers that have interaction with static or dynamic objects. To edit these effects the developers can use the same GUI to create the desire look and then save that to use it later. These effects being 2D simulations that are drawn on top of the material

don’t affect the performance. The complexity of this system is determined by the mesh density of the water planes (Figure 7.), which make use of LOD’s to optimize the experience, and the shader complexity of the material used. (Figure 8.)



(Figure 7. Mesh Density of the water planes with LOD’s.)



(Figure 8. Same water surface with different materials. Top - Opaque material/ Bottom - Translucent material. Left – Lit Mode/ Right - Shader Complexity.)

As you can see the material used makes a huge difference on the performance so this needs to be kept in mind while developing and test the performance of the scenes. Although the translucent material looks better it could be to expensive for a VR application.

2.4 SWOT Analysis:

2.4.1 Niagara:

Strengths • Great Flexibility. • A lot of sources online. • GPU Acceleration. **S**	**Weaknesses** • Steep learning curve. • The necessity of a specialized person for migrating and/or changes. • Resource-Intensive. **W**
Opportunities **O** • Ultra-Realistic Simulations. • Contiuned Advancements. • Advanced Particle Behavior.	**Threats** **T** • Technical Challenges. • Future Updates.

(Table 4. SWOT analysis for Niagara.)

2.4.2 Shaders:

Strengths

- Customization.
- Performance.
- Documentation & Sources.
- Mellow learning curve.

S

W

Weaknesses

- Debugging Challenge.
- Limited interactivity.
- Complexity.

Opportunities

- Combination with other Plug-ins.
- Exposed Parameters.

O

T

Threats

- Technical Challenges.
- Bugs.

(Table 5. SWOT analysis for Shaders.)

2.4.3 FluidNinja Live:

Strengths • Customization. • Performance. • Dynamic Interactivity. • Accesible through Blueprints. • User-friendly interface. **S**	**Weaknesses** • Steep learning curve. • Complexity. • Scares resources. **W**
Opportunities • 2D Simulation. • Drive Volumes. • Change Materials. • Extended Application. • Customization. **O**	**T** **Threats** • Small developing team. • Rapid technological changes.

(Table 6 SWOT analysis for FluidNinja.)

2.5 Define:

Considering these examples and the main takeaways from each, it's evident that using FluidNinja to drive all these elements is the optimal choice for this project. While the other solutions mentioned do possess their own advantages and are robust tools in their own right, they fall short in providing the comprehensive solution required for this specific project when compared to FluidNinja. It provides developers with everything needed under the same hood, integrating various elements to create an immersive experience. Additionally, it offers necessary optimization methods to ensure the system doesn't impact application performance. Presenting these findings and arguments to the team resulted in unanimous agreement that using FluidNinja as the backbone of the dynamic weather system is the best solution for this VR application.

3. Development:

The initial phase of creating this product involved acquainting myself with the primary tools, with FluidNinja Live being the most crucial. I commenced by delving into desk research by reading its documentation, recognizing it as the primary source of insights into the tool's functionality. It was imperative to dedicate ample time to thoroughly absorb this information, ensuring I possessed the requisite knowledge to embark on my journey with the tool. This entailed tasks ranging from integrating it with the Unreal project to comprehending its underlying mechanisms.

Subsequently, I progressed to the next stage, which entailed immersing myself in the developer-provided tutorials and use-case scenarios. Here, I seamlessly connected the theoretical knowledge from the documentation with practical visual examples. These levels not only facilitated experimentation with various aspects of the tool but also expedited the learning process. Of particular significance in this stage was the "Intro LEVEL," which methodically covered the fundamental components of the tool in discreet example stages, each capable of functioning independently but ultimately coalescing to create intricate systems. This approach offered a clear understanding of how each element of the tool complemented the others, elucidating the intricacies of building complex systems with diverse components.

As a newcomer to Blueprints within Unreal Engine, I recognized that relying solely on official documentation would have its limitations. Therefore, I supplemented my learning by exploring online tutorials and forum posts where different individuals tackled specific problems. Since my preferred method of learning is hands-on and practical, these tutorials served as invaluable step-by-step guides, helping me acclimate to the workflow and thought process required. Naturally, I later adapted and implemented these learnings to align with my unique requirements and ensure seamless integration with my specific use case.

3.1 Prototyping:

Given the project's unique nature, my priority was to kickstart prototyping at the earliest opportunity, even if it meant a temporary absence of formal research. I made this choice with the understanding that I would continuously engage in research on highly specific subjects as I encountered various challenges and situations throughout the product's development.

I began the project's prototyping phase by focusing on the clouds, a crucial element. The significance of clouds in our system lies in their constant visibility and their dominance on the screen. Given the need for realism and dynamism, it was evident that Unreal's volumetric clouds were the ideal choice.To control the clouds, I integrated FluidNinja, a tool enabling developers to manage cloud movement, behavior, shape, and coverage. This involved linking the actor and material of the clouds with the FluidNinja simulation, allowing for intuitive real-time adjustments through the FluidNinja GUI.

Once satisfied with the cloud appearance and behavior(Figure 9), I turned my attention to customizing them further through material instance editing. This marked the conclusion of the artistic aspect of this phase. Subsequently, I began crafting a blueprint responsible for transitioning between stormy and sunny weather. Determining the necessary parameters for this transition was the initial step. Manual experimentation revealed that modifying the material itself could achieve the desired outcome. However, I encountered my first challenge when attempting to change parameters dynamically.

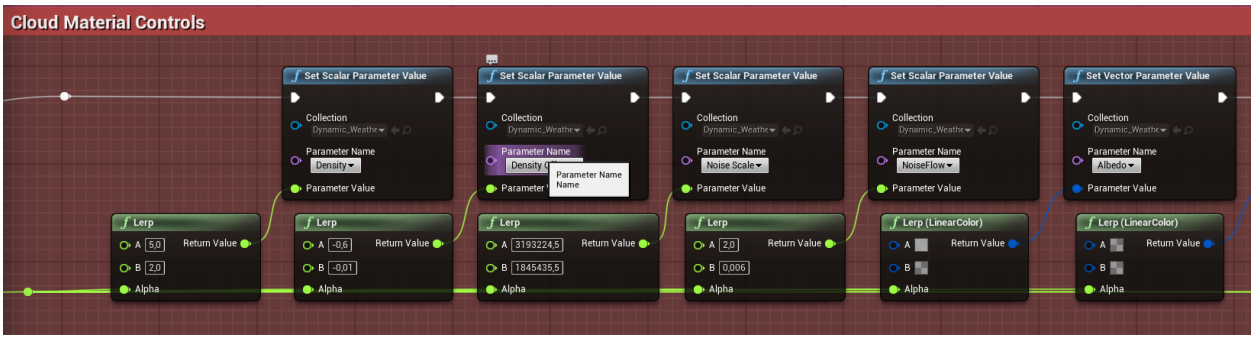
Referencing the material and clouds proved problematic, mainly due to the absence of a mesh in volumetric clouds. The "Create Dynamic Material Instance" node was explored, but it failed to deliver the required results, as it generated a copy of the material only in memory at playtime, not within the project files, and relied on a mesh target. After extensive research and consultation with experienced developers, I discovered a solution in a tutorial that elucidated "Material Parameters Collections" and their proper setup. (Figure 10.)

(Figure 9. Clouds in the sunny state.)

(Figure 10. Material Parameters Collection set-up with all the desired parameters.)

(Figure 11. Custom node inside the master material that references the desired parameter.)

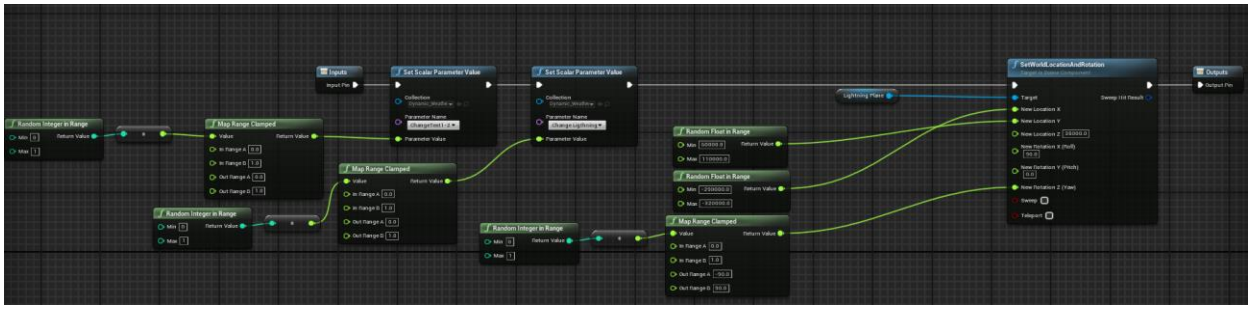
This method involved editing the master material with a custom node capable of referencing scalar or vector (Figure 11.) parameters predefined in the collection. Following this, a "Set Scalar/Vector Parameter Value" node was employed to reference the collection and the parameter name. (Figure 12.) Finally, a "Lerp" node facilitated the smooth transition between two values controlled by a timeline. Additionally, I implemented the functionality to adjust light intensity and sky luminance factor in the scene setup, a relatively straightforward process that merely required referencing and application within the blueprint, following the same methodology as before.



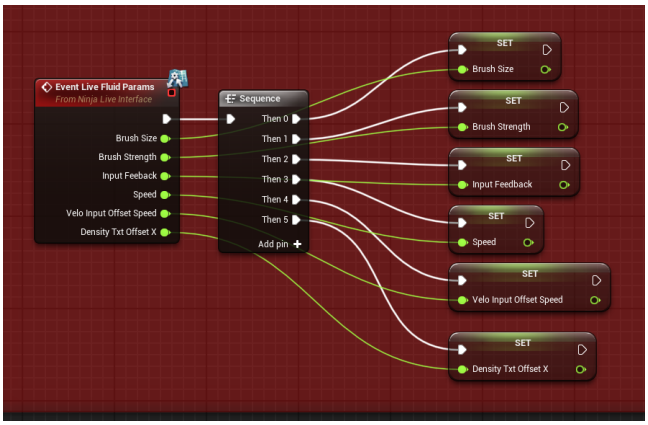
(Figure 12. Blueprint logic on how the clouds parameters change.)

After establishing the foundation for the clouds, my focus shifted to developing the expansive body of water crucial for the level. Ensuring a synchronized progression, I aligned both elements at the same level, enabling a systematic enhancement of functionality and system refinement. Leveraging insights gained from working on the clouds, I applied a similar technique to create blueprint functionality for the water. Configuring FluidNinja to interact seamlessly with static and movable objects required careful adjustment of settings through the GUI to achieve the desired interaction appearance. Once confirmed that the interactions functioned as intended, I integrated the necessary functionality into the blueprint.

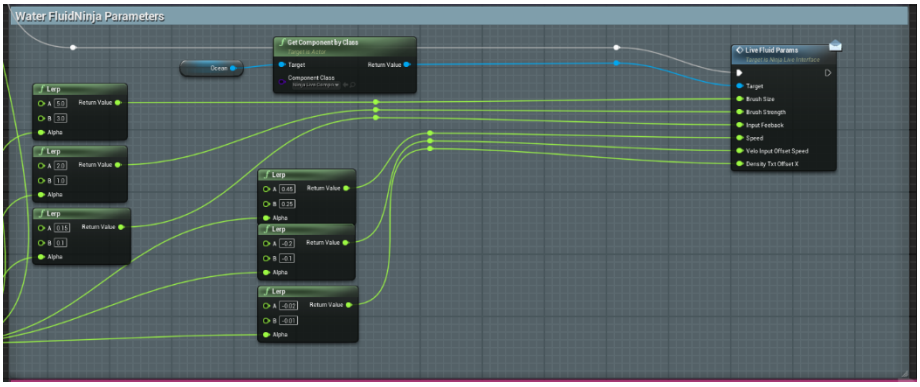
With both foundational systems operational, I turned my attention to expanding the weather system. Drawing inspiration from references, I envisioned distant rainfall and lightning as additional elements. Considering performance optimization and the distant nature of these elements, I opted for a straightforward approach—utilizing simple planes with textured materials. The rainfall texture, created in Photoshop using a soft brush and transformed into a black-and-white alpha image, was complemented by a gradient for gradual visibility adjustments. The blueprint logic for spawning these planes involved triggering a custom event through a timer and using the "SetWorldLocationAndRotation" node in a collapsed graph. Implementing lightning followed a similar process, employing random float values to spawn lightning at different locations. To enhance flexibility, I introduced three different textures, randomly switching between them for a more realistic effect. This variation was achieved through "Lerp" nodes in the material and random integer selections in the blueprint. (Figure 13.)


(Figure 13. Blueprint logic of spawning the lightning and changing the texture.)

Expanding the system further, I aimed to enable the editing of NinjaLive parameters via blueprints, crucial for controlling water interactions. Leveraging FluidNinja's existing functionality, I utilized a Blueprint Interface to facilitate communication between separate blueprints. This involved defining parameters as inputs within the interface, linking them to corresponding variables in the FluidNinja Component blueprint (Figure 14.), and integrating lerps with desired values in the main blueprint. (Figure 14.)



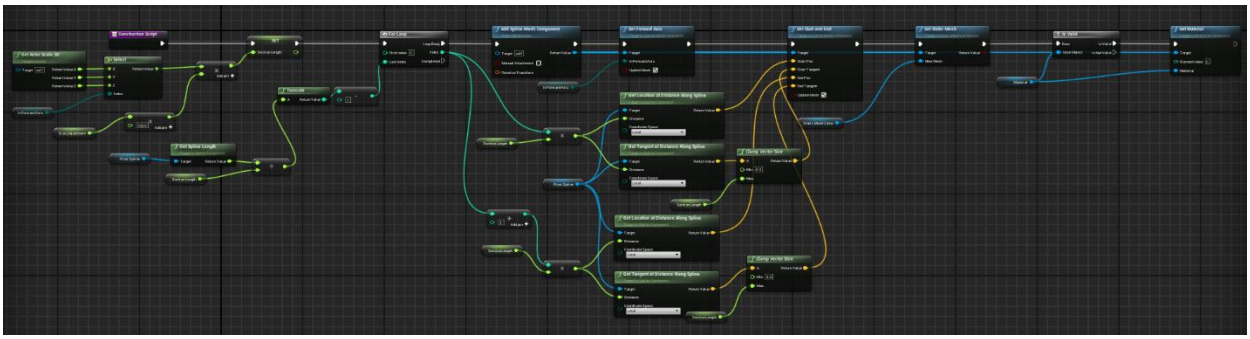
(Figure 14. Variables of the FluidNinja parameters in the main blueprint linked with the blueprint interface.)



(Figure 15. Blueprint interface integrated with the dynamic weather.)

This comprehensive approach ensured not only the seamless integration of weather elements but also the adaptability and control needed for a dynamic and realistic experience within the project.

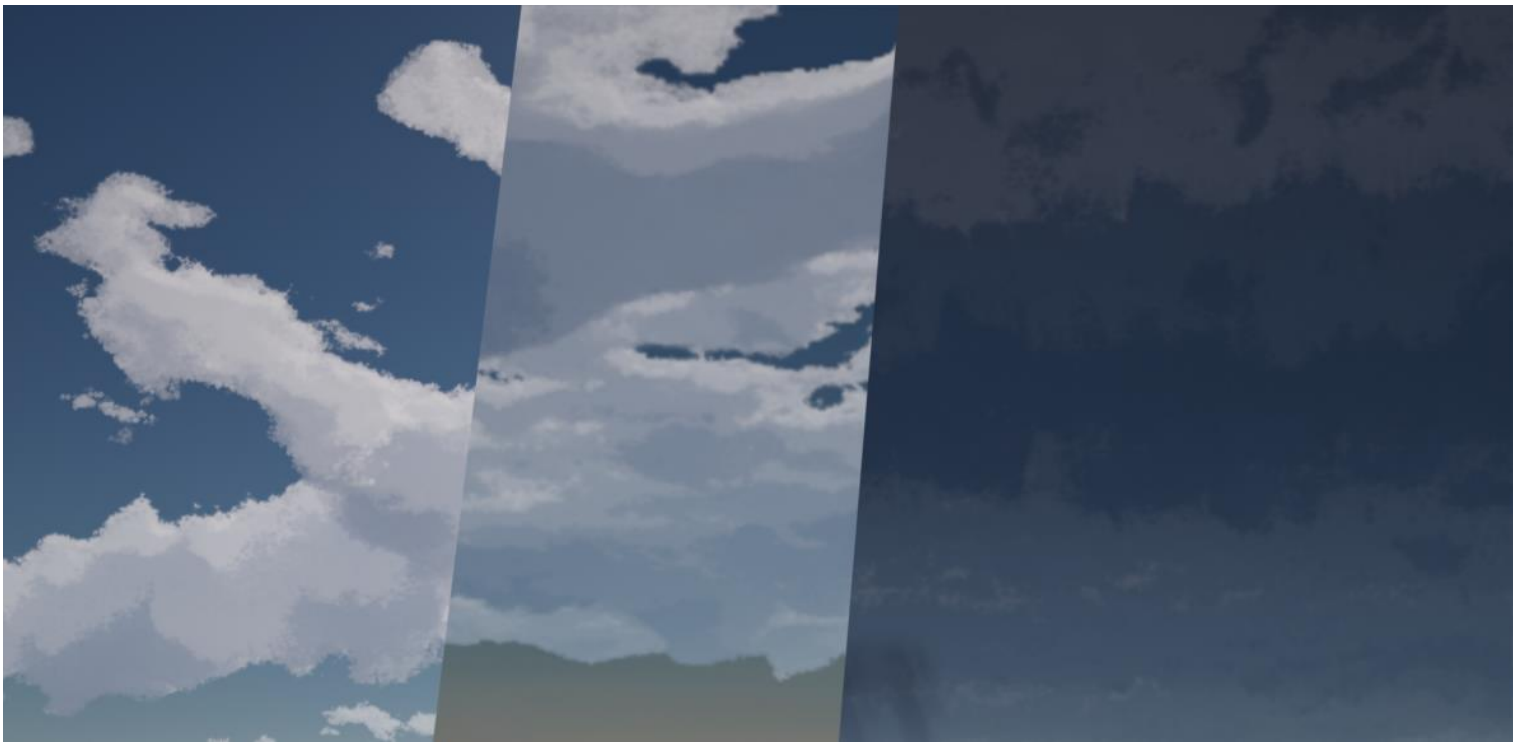
The subsequent phase involved establishing the canal system within the level. To facilitate implementation, I opted for the creation of a spline system, streamlining the addition of canals to the level. (Figure 16.) This straightforward system involves taking a pre-defined mesh and spawning it along a dynamically adjustable spline. Once this system was in place, my focus shifted to refining the material and enhancing object interaction using FluidNinja.



(Figure 16. Canal spline blueprint.)

Initially, my iteration depicted a small river with a somewhat rapid flow. Feedback highlighted that this appearance was unsuitable for our scenario, as canals typically exhibit a gentler flow, even in windy conditions, with only subtle ripples on the surface. Examining references closely, I modified the material to better suit our needs, consequently adjusting the object interaction.

The subsequent task involved implementing a transition from stormy to sunny weather in the blueprint(Figure 17.). During this process, I identified a flaw in how I manipulated material parameters. Both the ocean water and canal water utilized the same master material from FluidNinja, which I had edited to access parameters in the "Material Parameter Collection." To resolve this, I had to create another master material to incorporate distinct parameters controlling the canal material. While not optimal for optimization purposes, as master materials are intended for streamlined variation, it currently serve as the best solution until further research yields alternatives.



(Figure 17. From left to right -> sunny state, overcast state, stormy state.)

Addressing the next challenge, I aimed to enable the independent editing of parameters for two separate FluidNinja systems. In the previous setup, I utilized an array to determine the systems to edit, resulting in identical values for both. To overcome this, I introduced separate variables in the blueprint for each system, allowing direct targeting for parameter adjustments. This approach became necessary due to the uniform "parameter space" shared by all systems within the same base blueprint, requiring selective addressing.

In the subsequent phase, I focused on integrating sounds into the entire weather system, a pivotal step to enhance immersion. Sound plays a crucial role in connecting players fully with the visual elements of a level. To initiate this process, I began by delving into the sound setup within UE4, leveraging available documentation, and importing relevant royalty-free sounds discovered online.

After establishing sound cues for each desired audio element, I progressed to adding functionality within the blueprint. The initial task involved spawning sounds in the level, executed through two distinct methods: "SpawnSound2D" for sounds meant to permeate the entire map (e.g., wind), and "SpawnSoundAtLocation" for 3D sounds intended to be heard only in specific locations on the map (e.g., ocean). To ensure a robust implementation, I created two separate custom events for the sounds corresponding to the two weather states. This approach facilitates the initiation of sounds as the weather changes and provides flexibility for use in other blueprints if required.

You can see the whole Blueprint in the Appendix here: 9.4.1. Blueprint versions showcase or in detail here: https://blueprintue.com/blueprint/ws-srofq/ .

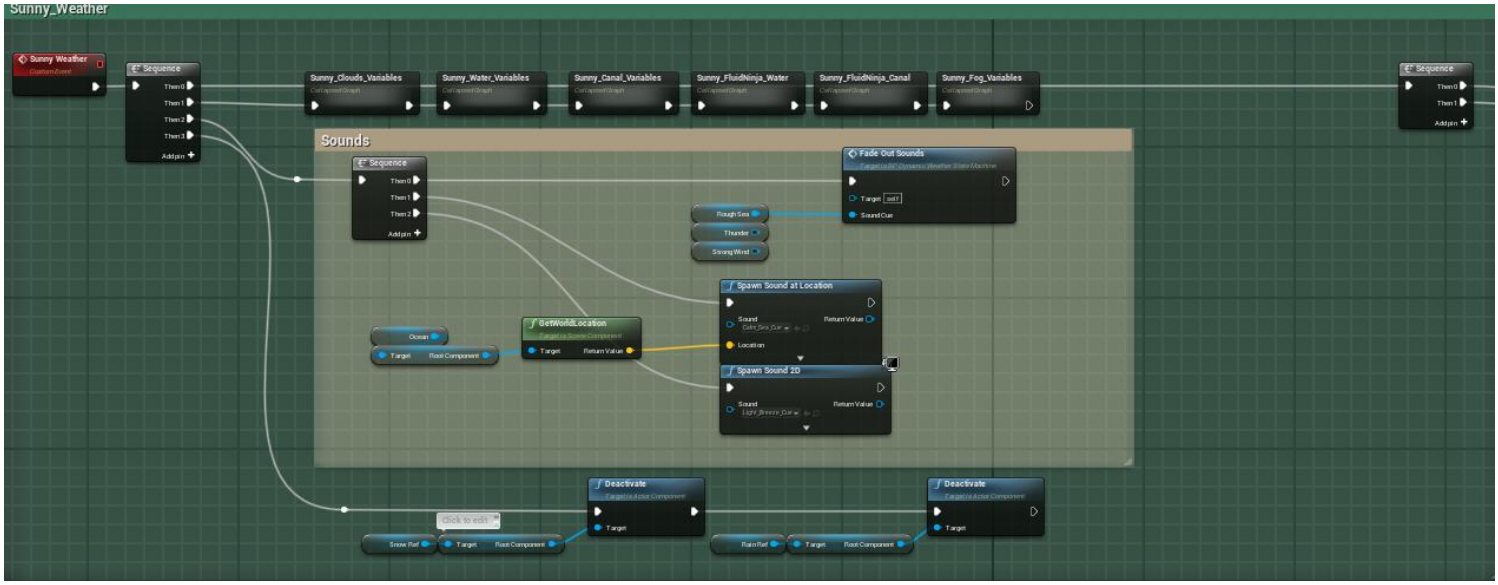
The next hurdle I encountered involved enhancing the blueprint to accommodate more than just two states, marking the most intricate functionality I had to implement thus far. Following a code review session with Justin Krogman, it became evident that I needed to transform the system into a more systematic structure rather than relying on hardcoded values. In essence, my previous approach involved interpolating between two values, "A" and "B," and then reversing the transition, which served well for two states but lacked the flexibility to accommodate a third or fourth state.

To overcome this limitation, I recognized the need for two sets of variables—namely "DEFAULT" and "NEW"—for each aspect of the system undergoing change. Another set of variables corresponding to each weather state was then created, and these were utilized to set the "NEW" values. Justin's guidance was instrumental, particularly in introducing the "Switch on Int" node, forming the backbone of this state machine system, and providing insights into the logical sequence of events.

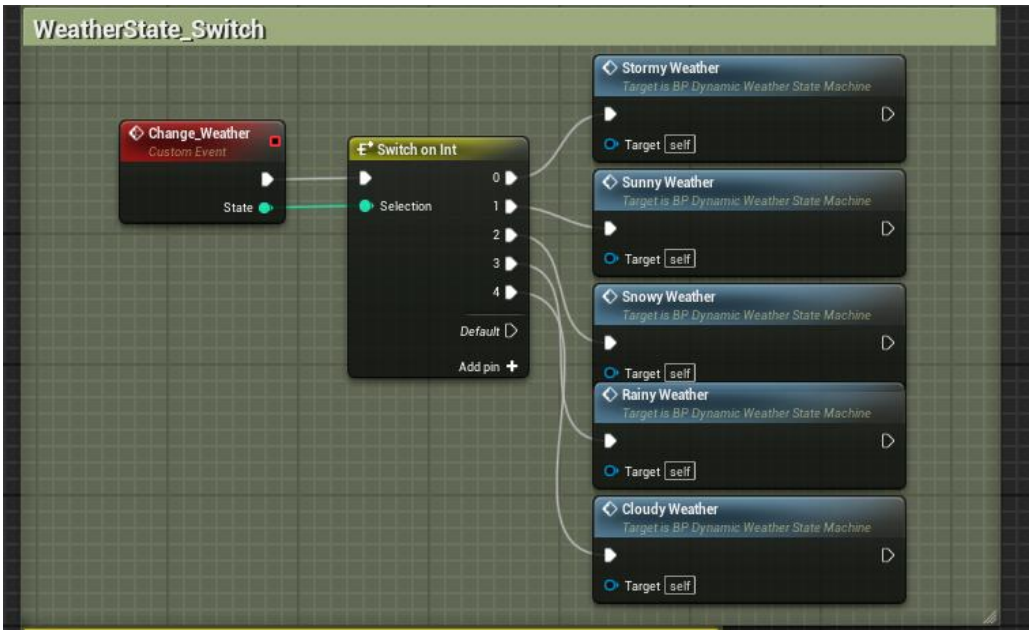
The following modifications were crucial for the new weather state machine to function effectively:

1. Each weather state became its own custom event, triggering its requisite functionality. (Figure 18.)
2. Utilizing the "Switch on Int" node, a single custom event was employed to trigger all the weather states. (Figure 19.)
3. The timeline now solely used its "Play from Start" input, ensuring changes consistently transitioned from "A" to "B." (Figure 20.)
4. A custom function was introduced to set all the "NEW" variables to the new "DEFAULT" immediately after the timeline completion. (Figure 21.)
5. Custom variables were used in each state to set the "NEW" variables. (Figure 22.)

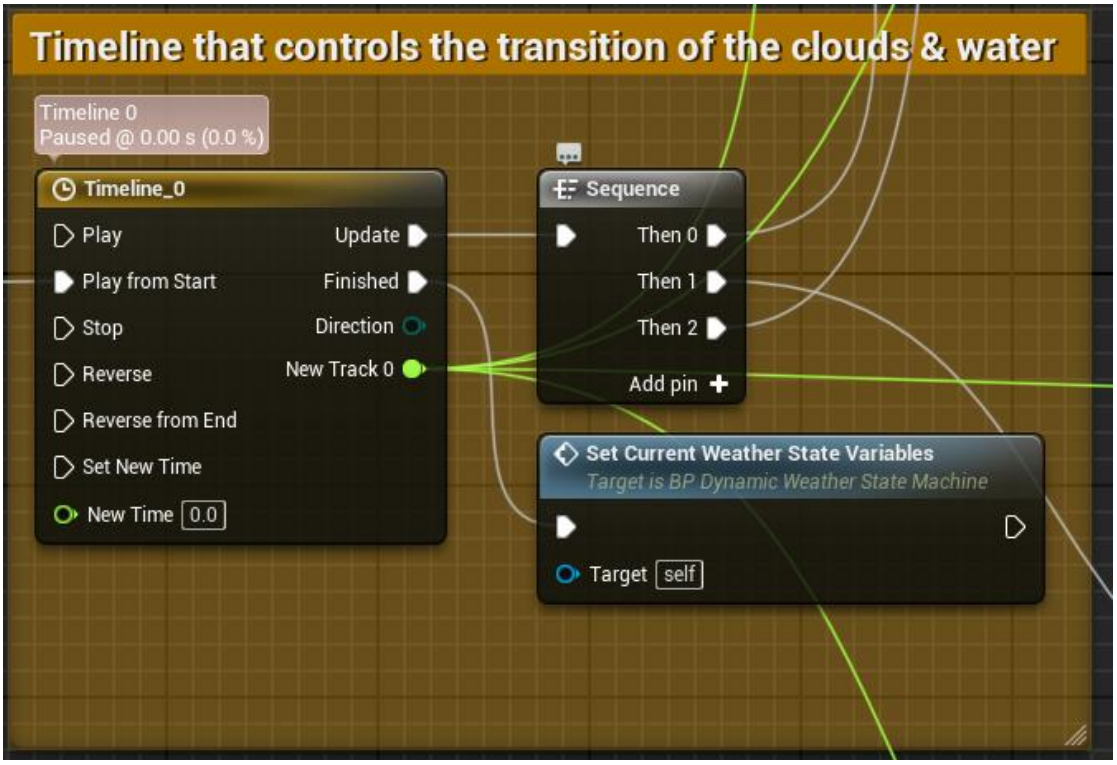
6. For testing purposes, the custom event "Change_Weather" was connected to key inputs, each corresponding to a different weather state. In the actual level, this event would be triggered by other blueprints. (Figure 23.)



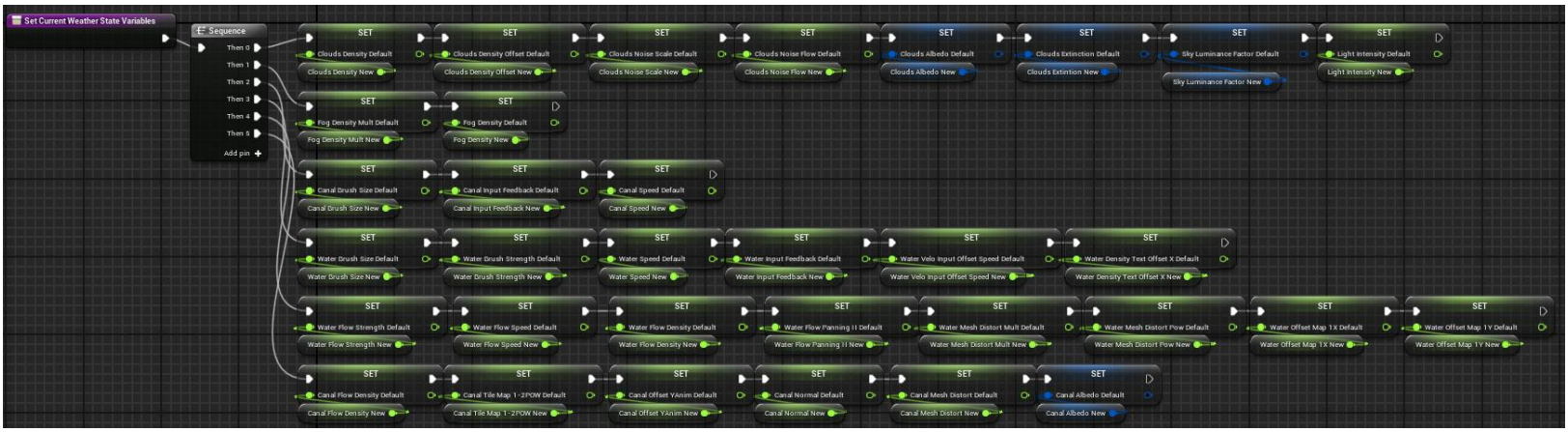
(Figure 18. Sunny weather state with its functionality.)



(Figure 19. Switch on Int node hooked up to the “Change_Weather” custom event to trigger the correct weather state.)

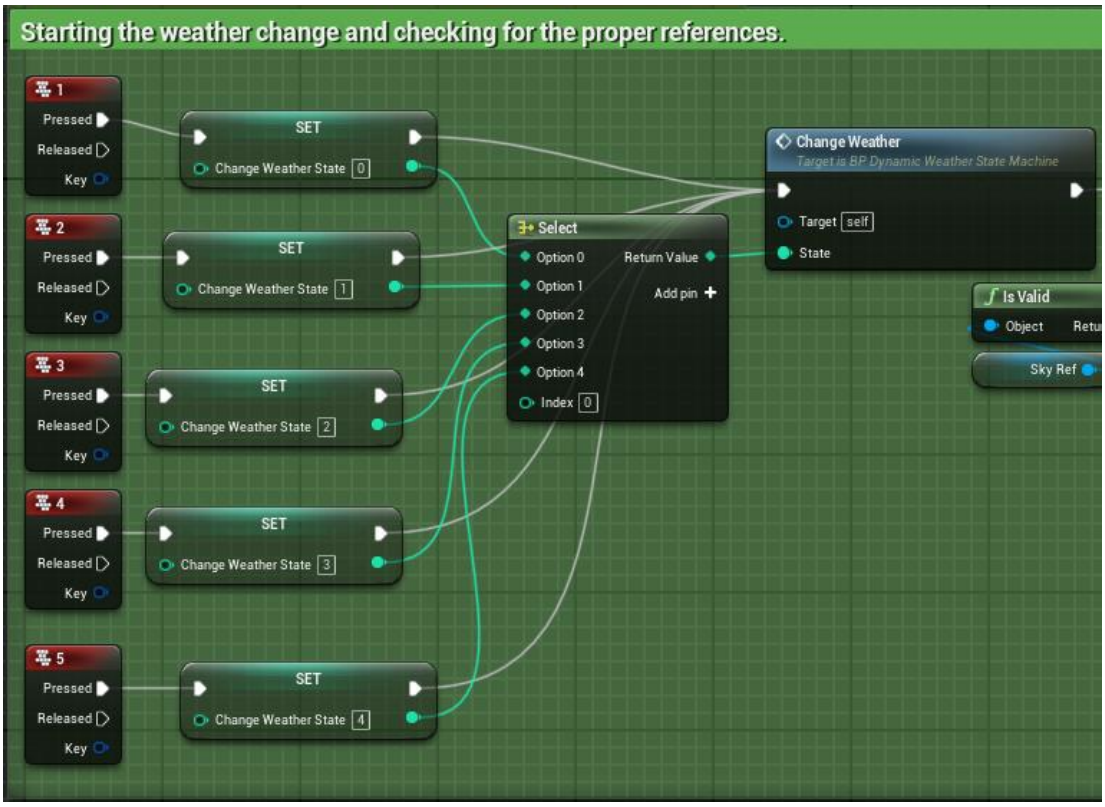


(Figure 20. Timeline set-up.)



(Figure 21. Custom function to set the “NEW” values to be “DEFAULT”.)

(Figure 22. Custom variables are used to set the “NEW” variables inside the weather state.)



(Figure 23. Selecting the desired weather state and initializing the change.)

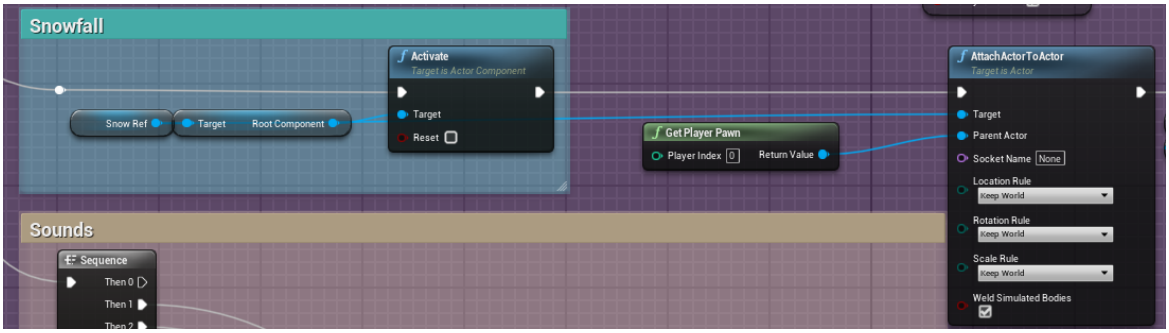
These steps not only rendered the system more robust and developer-friendly but also facilitated the addition of more weather states and the incorporation of complexity into each state in an organized manner. Moreover, making each weather state variable public allowed developers to modify specific aspects of each weather state without accessing the blueprint itself.

After addressing the previous challenges, I took on the task of expanding the range of weather states for use in various scenes within the VR application. I incorporated three additional states suitable for our project: a cloudy state, a rainy state, and a snowy state.

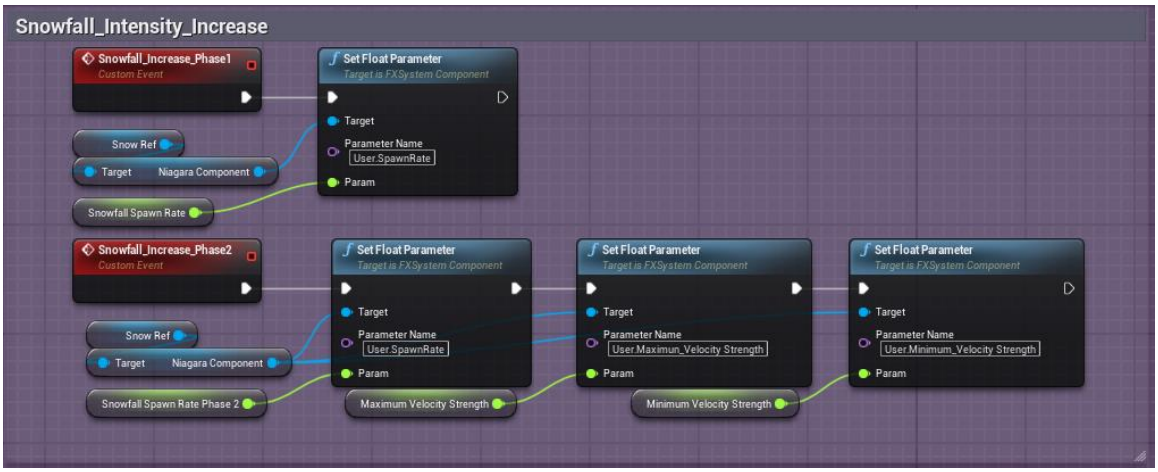
You can see the whole blueprint in the Appendix here: 9.4.1. Blueprint versions showcase or in detail here: https://blueprintue.com/blueprint/38dtzbfo

For the rainy and snowy states, I introduced two Niagara particle systems to simulate rain and snowfall around the player. This introduced new challenges and necessitated the integration of additional functionality into the blueprint. The initial step involved creating these systems while ensuring minimal performance impact. To achieve this, I kept the spawn radius small and attached the systems to the player, providing the impression that the effects were present throughout the entire map while optimizing performance. Exposing relevant parameters allowed for easy adjustment via blueprint, enabling developers to increase the intensity of these systems if desired.

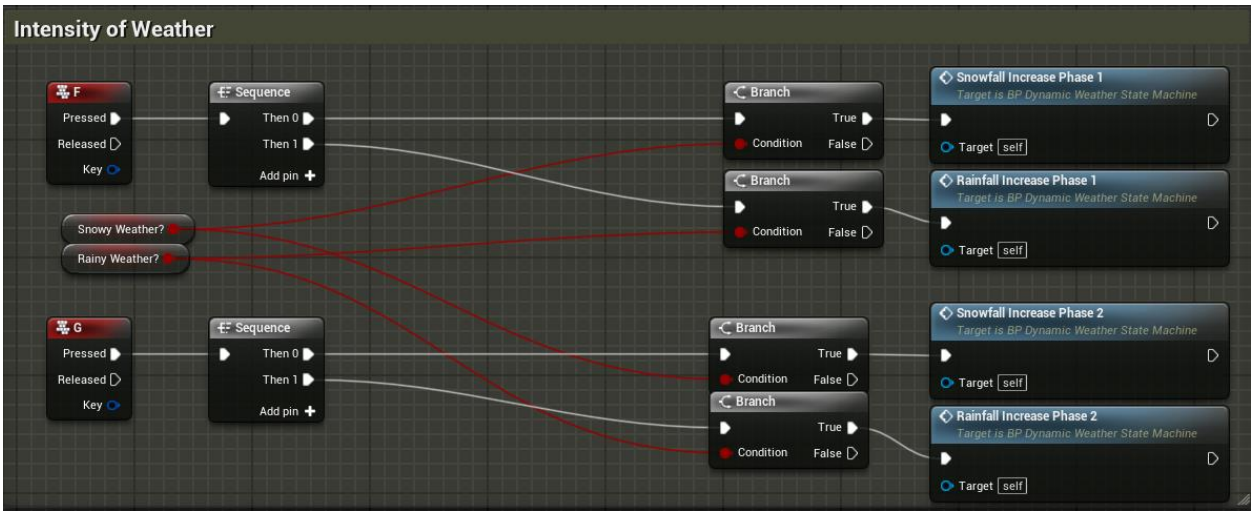
In terms of blueprint functionality, the first step was to spawn the system in the desired state and attach it to the player. (Figure 24). Subsequently, I implemented functionality to increase the intensity of these effects independently of the state switching. Two custom events were created for each particle system, allowing developers to enhance intensity in two phases. (Figure 25). Each event modified the previously exposed parameters of the Niagara system using public variables, providing developers with flexibility. To activate these systems, I linked them to key inputs, allowing each phase to be triggered separately. (Figure 26). Before it also checks if the complementary state has been triggered. In the actual level, these events can be triggered through another blueprint or box triggers as needed by developers.



(Figure 24. Spawning the Niagara system in the desired state.)

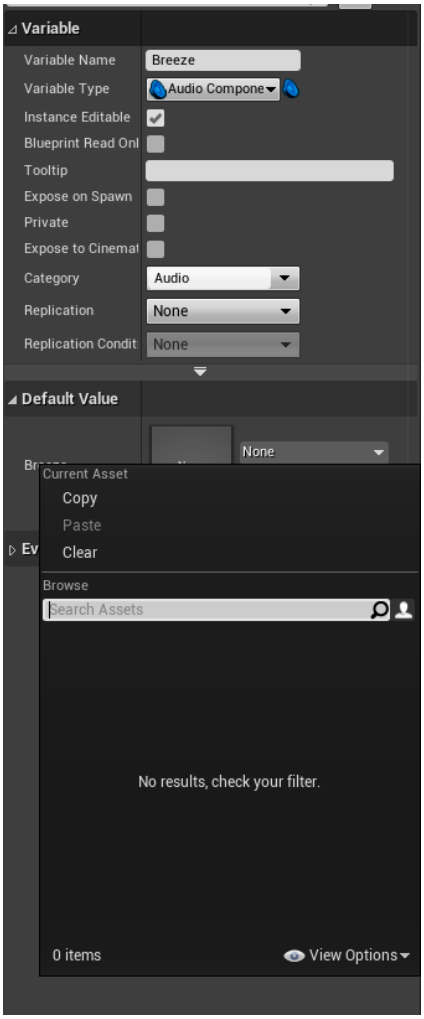


(Figure 25. Intensity increases in two phases.)



(Figure 26. Initializing the increase in intensity.)

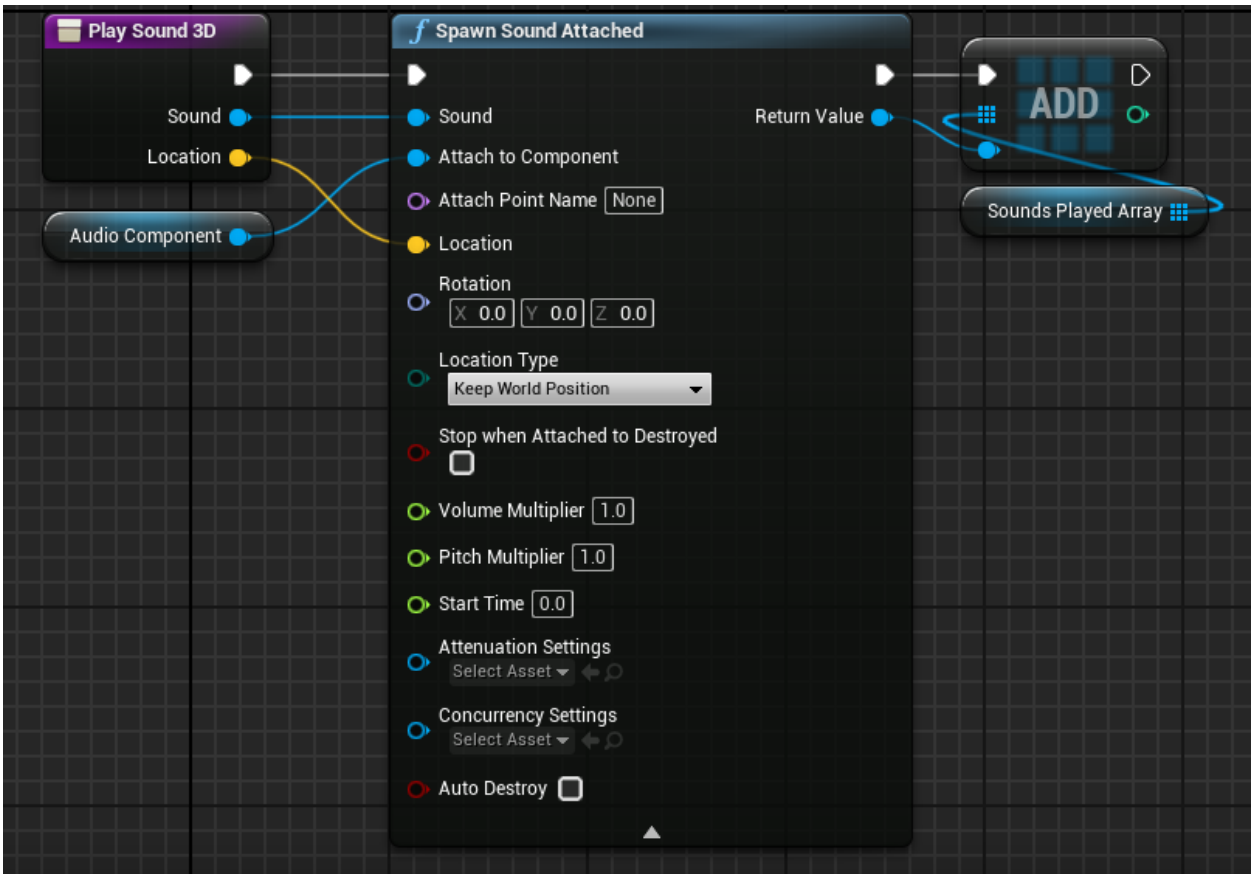
The subsequent challenge involved addressing a system error that occurred after completing a game level. The issue stemmed from the audio setup, manifesting as an "accessed none" error in the message log. This arose because the audio components did not recognize the newly created sound cues as default values (Figure 27).



(Figure 27. Audio Component details panel inside the blueprint.)

To rectify this, I modified these variables to accept "sound cues," resolving the error but introducing new complexities. The nodes responsible for spawning and fading out sounds expected audio components as targets. Through an iterative process involving trial and error, coupled with input from fellow developers, I successfully identified a solution, ensuring the system operated as intended.

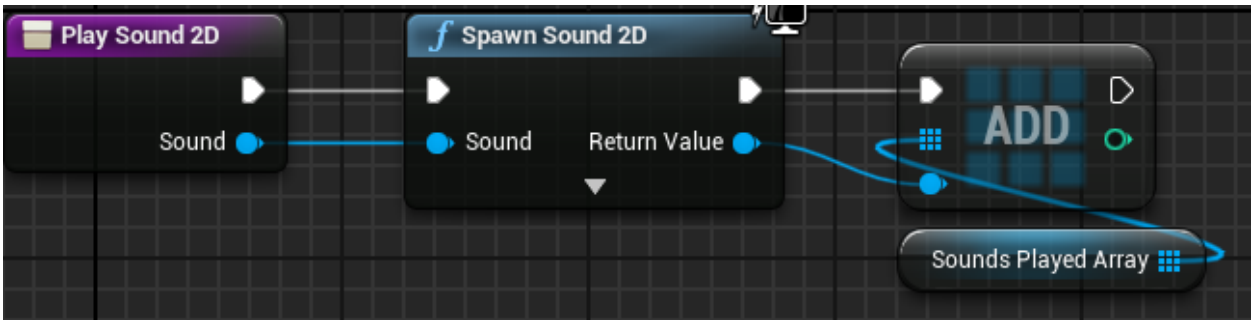
The solution involved crafting three custom functions: one for spawning 3D sounds, another for 2D sounds, and a third for fading them out. The "Play_Sound_3D" function featured two custom inputs directing the target and location of the "PlaySoundAttached" node. The latter's execution pin connected to an "ADD" node linked to an array variable (Figure 28).



(Figure 28. Play Sound 3D custom function.)

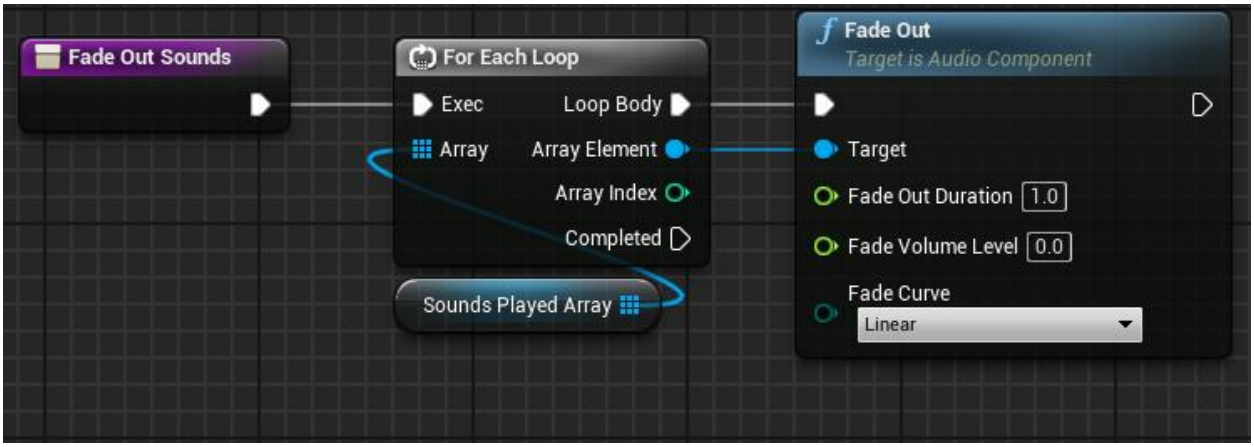
This configuration was chosen for its compatibility with sound cues and the ability to specify spawn locations, maximizing the utilization of the attenuation feature.

For 2D sounds, the setup mirrored the 3D counterpart, utilizing the "SpawnSound2D" node followed by the same array configuration (Figure 29).



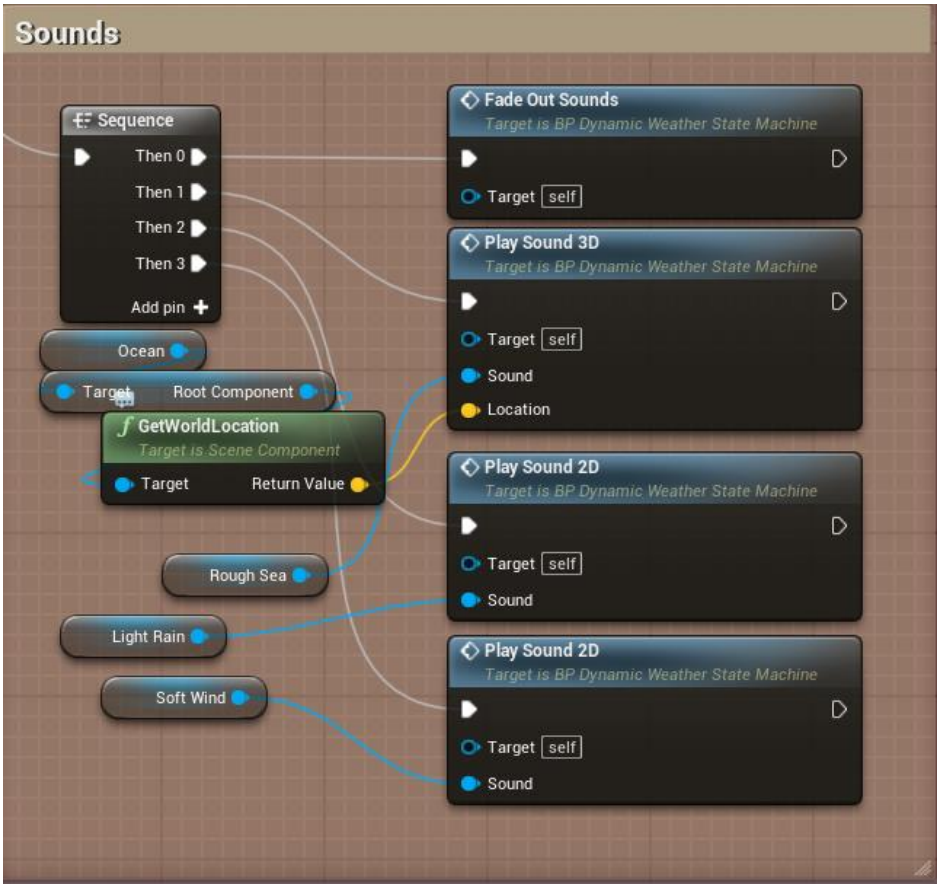
(Figure 29. Play Sound 2D custom function.)

The array proved essential when fading out sounds since the "FadeOut" node did not seamlessly work with sound cue variables. This array stored every played sound across all states, and through a "ForEachLoop" node, systematically faded out each sound (Figure 30).



(Figure 30. Fade Out custom function.)

In each weather state, these functions were employed to first fade out active sounds and subsequently spawn the required sounds. Public variables were introduced for each sound, providing developers the flexibility to easily modify specific sounds (Figure 31).

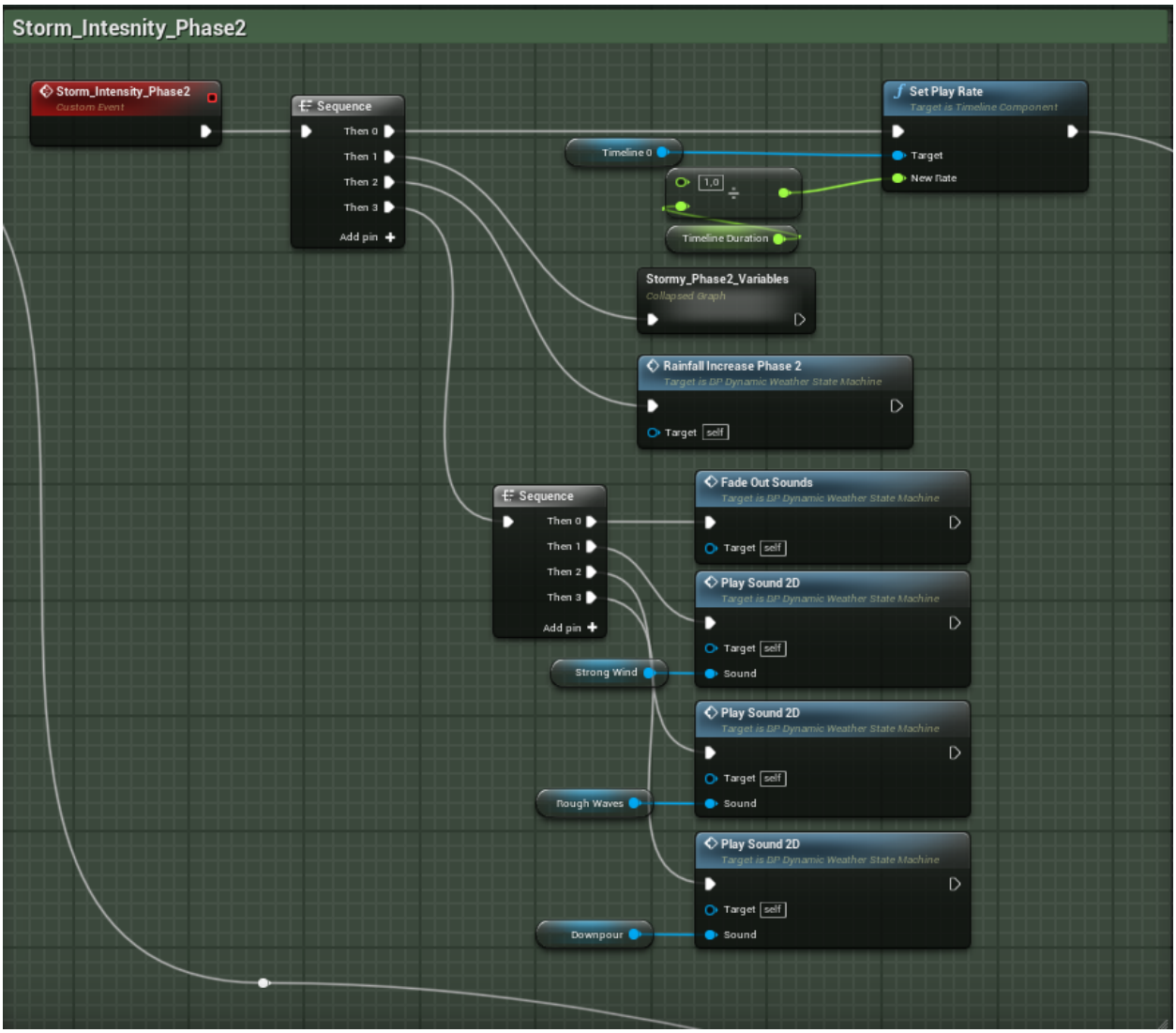


(Figure 31. Sound set-up in the weather state.)

Although the process demanded substantial time to establish the correct configuration, the end result eliminated the error and established a more robust system. This not only optimized the code but also enhanced its accessibility for future developers.

You can see the whole blueprint in the Appendix here: 9.4.1. Blueprint versions showcase or in detail here: https://blueprintue.com/blueprint/4o808ph9/

The next stage in the development process involved enhancing the stormy state's intensity by modifying parameters, adjusting sounds, and introducing new elements. This particular aspect of the system capitalizes on intensity augmentation in two phases of the Niagara systems, I created before. To implement this, my initial task was to devise a method that utilized the same timeline but in a shorter duration for altering the desired parameters. I aimed to achieve this without reworking a portion of the logic, thereby optimizing the blueprint. To accomplish this optimization, I employed a "SetPlayRate" node with a variable influencing the timeline. This adjustment imparts a shorter duration to the timeline, accelerating the transition. Subsequently, I replicated this setup to modify parameters, trigger sound spawns, and invoke a custom event designed to escalate the Niagara system. (Figure 32.)



(Figure 32. Storm intensity increase functionality.)

The next complex feature I aimed to integrate into my dynamic weather system was a day and night cycle. Although players might not frequently experience nighttime in their playthrough, altering the sun's position introduces an additional layer of realism and enhances scenes with time transitions. To initiate the necessary functionalities, I implemented a custom event. The initial step involved adjusting the sun's position, specifically its rotation on the "Y" axis, accomplished using an "AddActorLocalRotation" node. The rotation speed was controlled by a variable, multiplied by the world delta seconds.

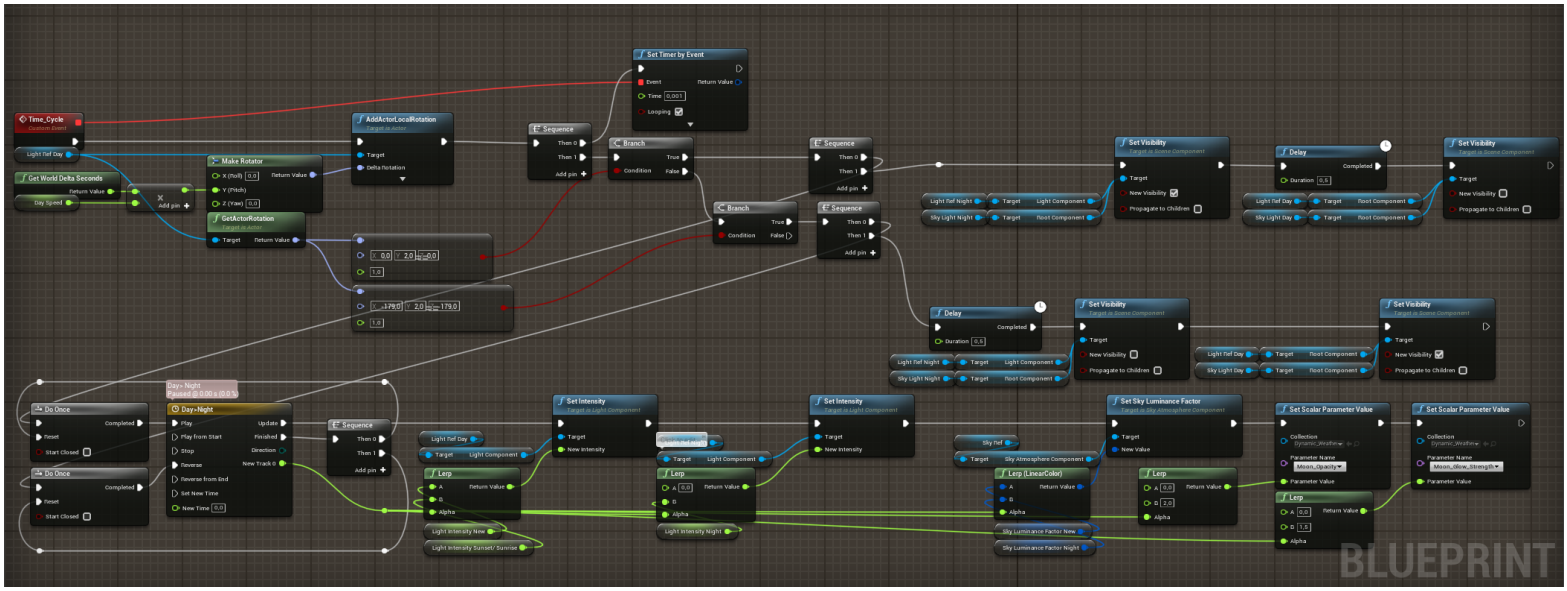
Creating a smooth transition between day and night, especially during sunset and sunrise, required the use of a timeline. Initiating this timeline at a specific sun rotation presented challenges due to the light's rotation nature. After experimentation and exploration of various conditional setups using ">" and "<" nodes, I found a solution using an "==" rotator node. This node compares the light's rotation to a specific value, outputting a Boolean. Two such nodes, each with distinct values, fed into corresponding "branch" nodes. Depending on the branch triggered by the sun, the timeline played or reversed. A "DoOnce" node ensured the timeline did not repeatedly trigger, with its "finish" execution pin resetting the node.

The subsequent step involved incorporating a moon to establish a nighttime atmosphere. Following a tutorial by Underscore on YouTube (underscore. (2018, May 9). *UE4 Tutorial: Night Scene with Moon*), I created a simple blueprint with a sphere and a plane, both oriented toward the world's origin. This blueprint served to complement the night atmosphere, enhanced by an additional directional light and skylight. To avoid complications with the existing directional light, I opted to add new lights and manage their visibility through blueprints. Although initially exploring a second lightning scenario, I found this approach impractical due to the project's structure.

The first method I explored was the creation of a second lightning scenario and loading that at nighttime via blueprint functionality. While this method proved effective, it posed challenges within the project's existing structure. Given that the team utilized a custom engine built from AFCore, the project was configured to utilize one persistent level with individual levels acting as sub-levels.

However, adopting the lightning scenario method would necessitate the addition of two more sub-levels for each existing sub-level, essentially doubling the number of levels in the project. To implement this, modifications to the default AFCore functionality would be essential, potentially leading to complications and unforeseen issues. Moreover, it would not have been the most efficient solution for the project.

Due to these challenges and potential disruptions to the established project structure, the lightning scenario method was ruled out. Instead, I opted for a different approach, managing the visibility of elements via blueprints. This setup ensured that only one directional light was active at any given time, facilitating a smooth transition between day and night. To grant developers flexibility, I added a public Boolean to enable or disable the day/night cycle based on project requirements. (Figure 33.)



(Figure 33. Day/Night Cycle functionality.)

You can see the whole Blueprint in the Appendix here: 9.4.1. Blueprint versions showcase or in detail here: https://blueprintue.com/blueprint/448d11ci/

3.2 Implementation:

Integrating these systems into the test level proved to be a straightforward process. I initiated the task by copy-pasting the requisite elements, ensuring meticulous attention to setting up accurate references. Once the foundational elements were in place, I methodically positioned everything and conducted comprehensive tests to confirm seamless functionality. Satisfied with the initial results, I progressed to the subsequent phase, focusing on fine-tuning the materials and refining the interaction dynamics with FluidNinja.

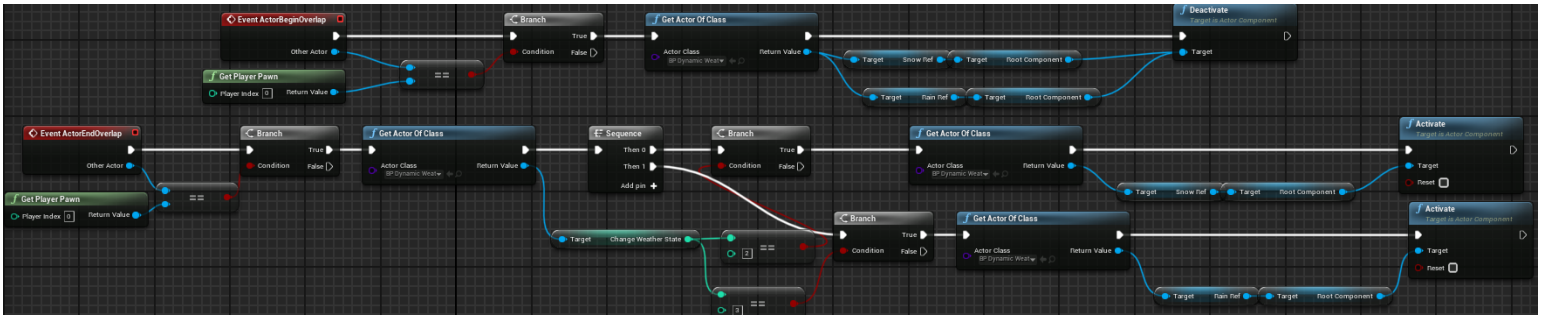
These refinements encompassed subtle adjustments such as altering the tiling of specific maps, refining the direction and speed of panning, and scaling the interaction volumes to better align with the level's requirements. Following these material enhancements, I proceeded to implement the canal system utilizing the previously created spline framework, meticulously tracing the path of the canals.

Negotiating corners proved challenging due to their unique shapes and angles. I maximized coverage by carefully extending the splines and ensured that the intersections of two splines remained inconspicuous to players. Addressing problematic areas that persisted, I strategically introduced small planes, conducting thorough playthrough tests to ascertain their imperceptibility to players. The goal was to create a seamless integration of the systems while maintaining a visually cohesive and functionally sound environment.

After developing the updated version of the dynamic weather blueprint, which incorporates a state machine, the next step was to implement it into the various levels. The introduction of new functionalities in the blueprint required specific adjustments to ensure the correct spawning of sounds and the proper functioning of particle systems. This new weather system was integrated into all three scenes currently under development.

While there were minor challenges encountered during this process, they were more of an annoyance than a serious issue. Through trial and error, every aspect of the blueprint was eventually functioning as intended. The addition of new elements, such as particle systems, necessitated extra functionality within the levels. Given that players would be inside during certain parts of the playthrough, it was crucial to prevent rainfall and snowfall particles from being present in those areas.

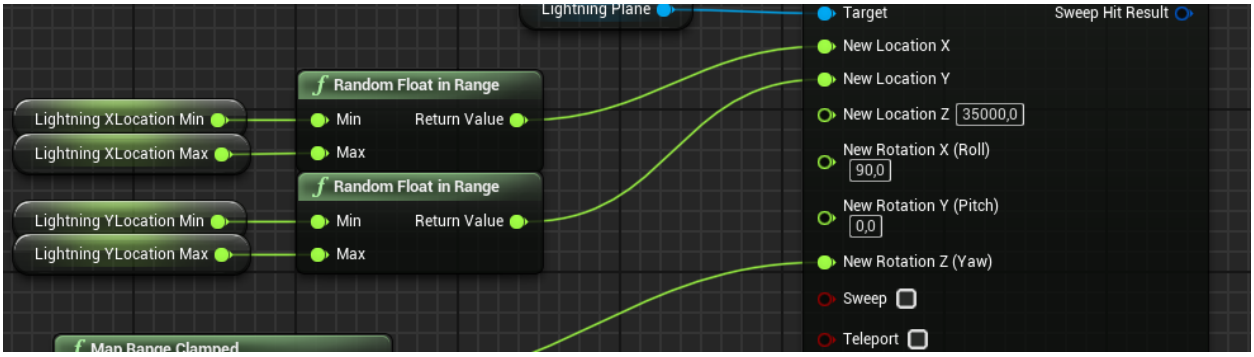
To achieve this, two potential methods were considered. The first involved enabling collision on the particles to interact with meshes in the world. However, this approach was dismissed due to concerns about its impact on performance. Implementing collision would require spawning the particles at a higher position, thereby increasing their life duration, resulting in a potential performance hit, especially in a VR application. Instead, a more optimized solution was pursued—creating a box trigger blueprint that deactivates particles when the player enters it and respawns them upon exit. (Figure 34.) This approach avoided the need to modify the particles directly, providing developers the flexibility to place triggers as needed.



(Figure 34. Trigger box functionality.)

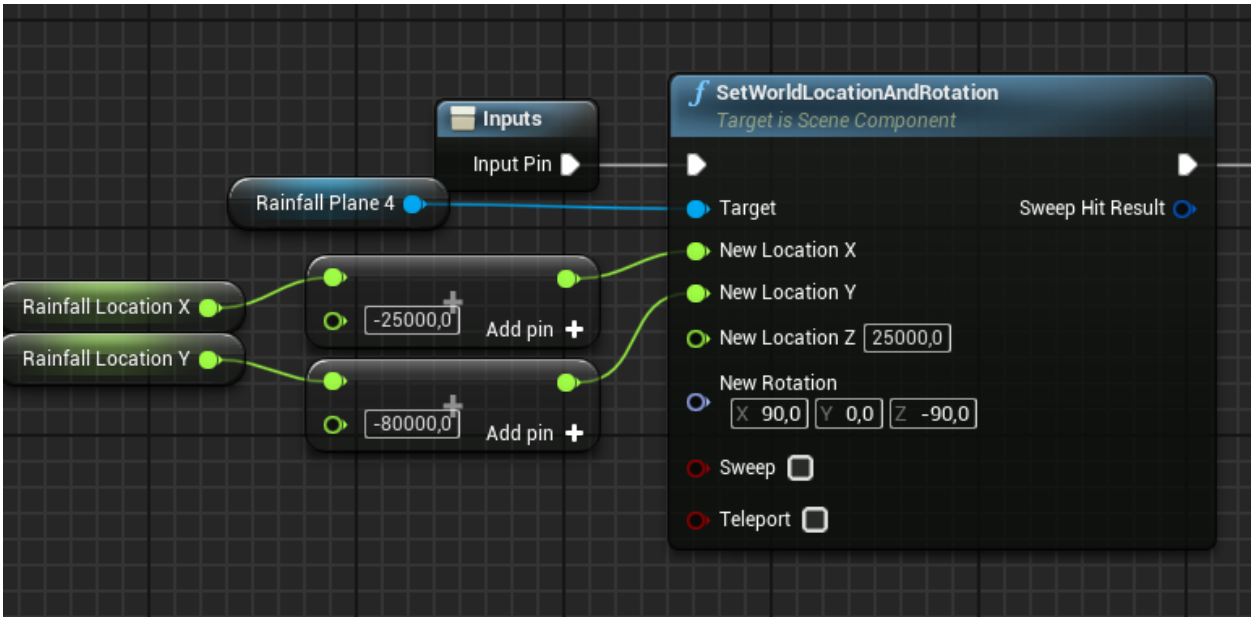
The subsequent task involved fine-tuning the variable values in each level to achieve the desired visual appearance for each weather state present during the player's experience.

Implementing the blueprint in new levels this brought up new necessities to its functionality. Firstly, I have noticed the necessity of spawning the lightning and rainfall at different locations. This can be easily implemented with a set of new public variables. (Figure 35.)



(Figure 35. Lightning location variables.)

The same process was used for the rainfall but because I have more than one plane, I decided to use the same variables and to change the location of the others so they're not on top of each other to multiple those variables with a value. (Figure 36.)



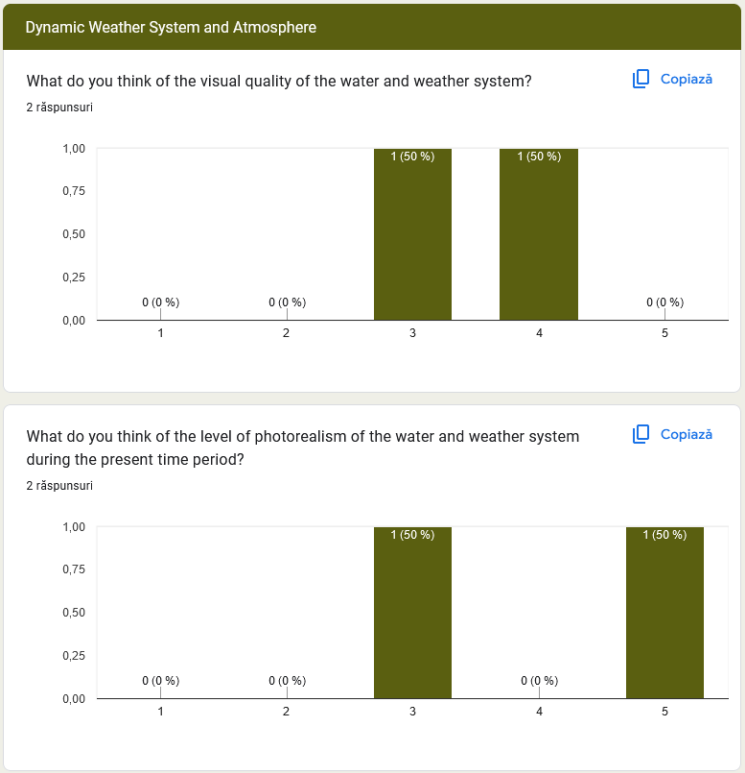
(Figure 36. Rainfall location variables.)

4. Testing:

As previously mentioned in the course of this project, I maintain a rigorous testing regimen for the features I integrate. While desktop mode serves well for swift iterations and troubleshooting within the blueprint and FluidNinja systems, it falls short of providing an authentic assessment of the application's performance in VR, the actual environment of its intended use. To bridge this gap, I borrowed a VR headset from a colleague for comprehensive testing, both in a controlled test scene and within the current state of the actual level. To make sure I have a structure and don't miss anything important I have created a plan to follow. (Appendix 1 - Performance Test 1) This scrutiny brought to light several issues.

Foremost among these were visual artifacts involving volumetric clouds and water. Specifically, in the right eye, volumetric clouds were rendering atop everything, including solid meshes, and peculiar reflections marred the water's surface. In my quest for solutions, I stumbled upon a post on the official UE Forum where fellow developers reported analogous errors (https://forums.unrealengine.com/t/volumetric-clouds-in-ue-4-26-do-not-render-in-vr/153838/8). The post disclosed that Unreal Engine developers had rectified the cloud problem in UE 5. To address it in version 4.27, the post suggested copying two lines of code and creating a custom engine build. Upon consultation with the team, I discovered that this problem had been resolved on the office PC, with the custom engine build, and the clouds render accurately, but the water not so another solution should be found.

The team in our office recently conducted another test session involving actual users, focusing on various aspects outlined in a comprehensive test plan crafted by Justin Krogman (Appendix 2 - Indietopia Test Procedure). During this test, I took the opportunity to solicit feedback specifically regarding the visual presentation of weather elements such as water and clouds. To collect feedback efficiently, the team devised a survey, incorporating two questions related to the aspects I had been developing. Preliminary results indicate a positive reception of the visual representation of these elements. Two testers assigned a score of 3 out of 5 (1 being the lowest, 5 the highest) to both questions, while another tester rated them at 4 and 5, respectively (Figures 37 & 38). These scores suggest that even in this early blockout stage, the system's elements are identifiable and align with the intended visual aesthetic.



Legend: 1 – Poor Quality.

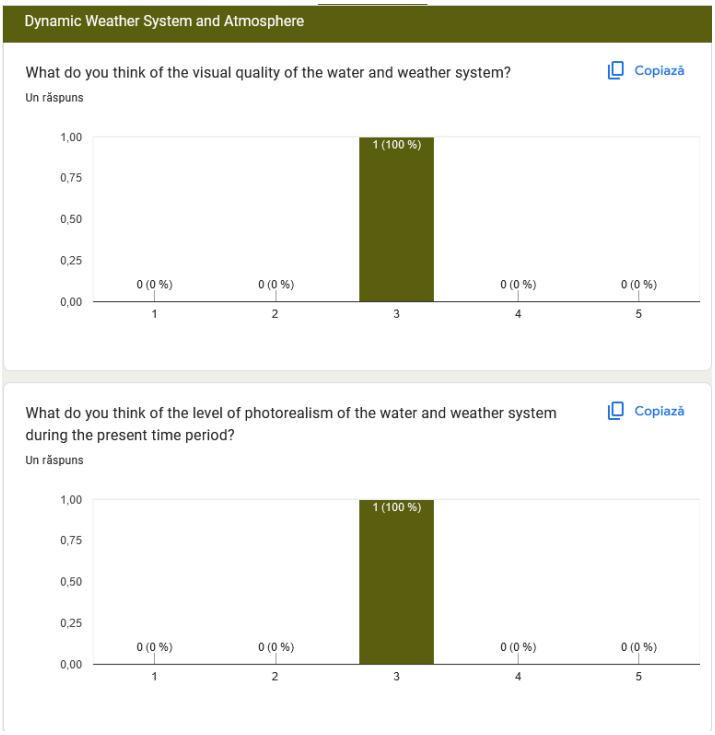
2 – Neutral.

3 - Decent Quality.

4 – Good Quality.

5 – Excellent Quality.

(Figure 37. Results from the first session – 2 Testers.)



Legend: 1 – Poor Quality.

2 – Neutral.

3 – Decent Quality.

4 – Good Quality.

5 – Excellent Quality.

(Figure 38. Result from the second session – 1 Tester.)

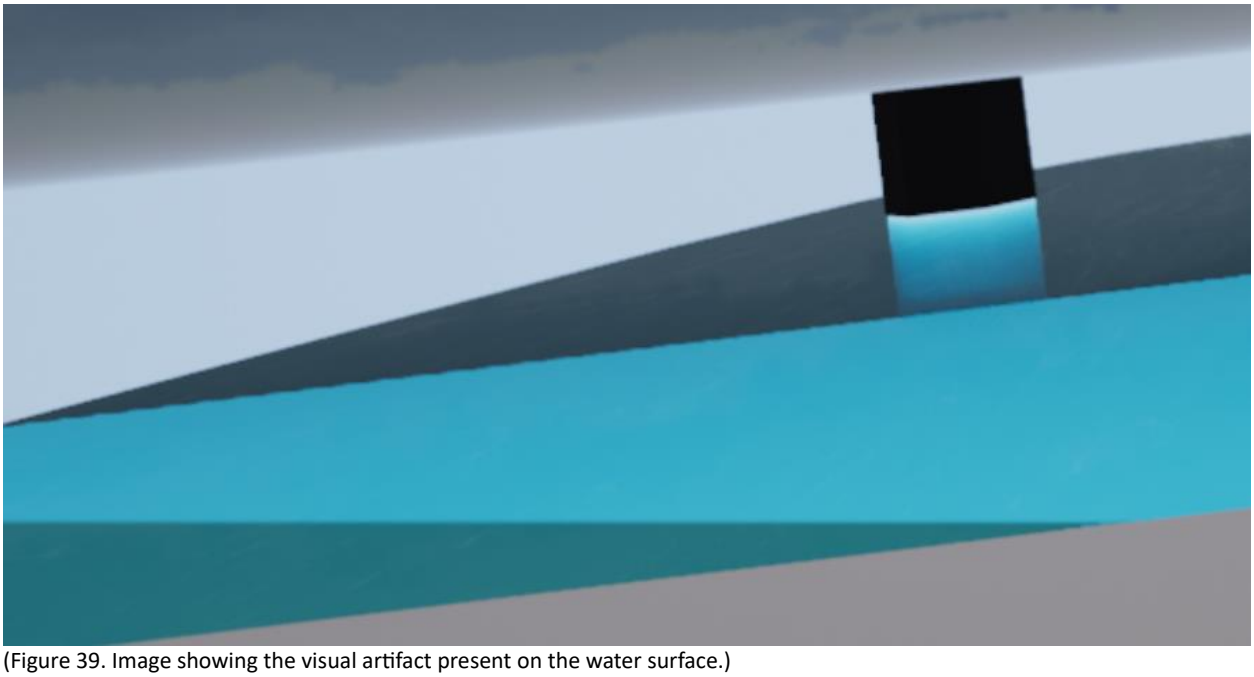
4.1 Performance:

Understanding the hardware specifications of the PC intended for running the application is crucial at this stage. The designated PC boasts an AMD Ryzen 7 5800X processor paired with an Nvidia RTX 3080Ti, which is noteworthy as it closely resembles the configuration of my own testing machine. While my personal setup utilizes an AMD Ryzen 7 3800X and an Nvidia RTX 2070, a comparison using benchmark websites clearly indicates that the PC set to run the application is superior in performance. (Appendix - PC Hardware comparison)

This alignment is promising because optimizing the application to run smoothly on my PC suggests that it should perform equally well, if not better, on the museum's PC due to its superior hardware specifications.

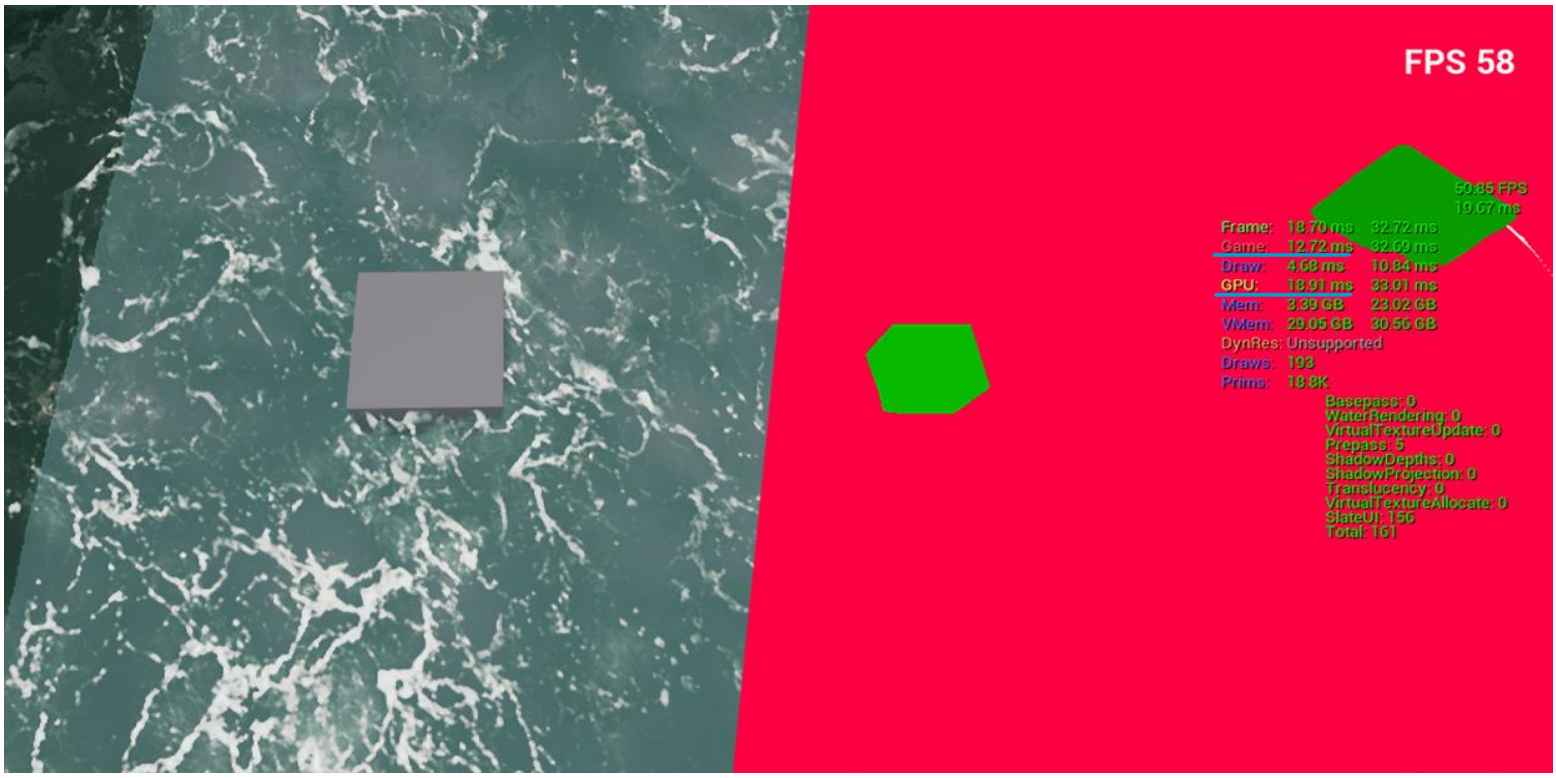
To test the performance of the system I made use of a multitude of tools which Unreal offers to developers. I used the GPU Visualizer, which shows how much the process of the GPU take to compute. I used the profiler which shows the performance over a period of time which I record at playtime. I used the detail stats which show the FPS, how much time it takes to render a frame, it shows the draw calls, memory usage, game thread and rendering thread. I will showcase all these tools as I explain what I did during the testing session.

The first step was to address the known problem I have with the water. For some reason in the right eye of the VR headset, there is a weird reflection going on. It was hard to get an image of it, but I managed to. (Figure 39) In this image, we can see a white reflection and a very harsh line between a black and white area.

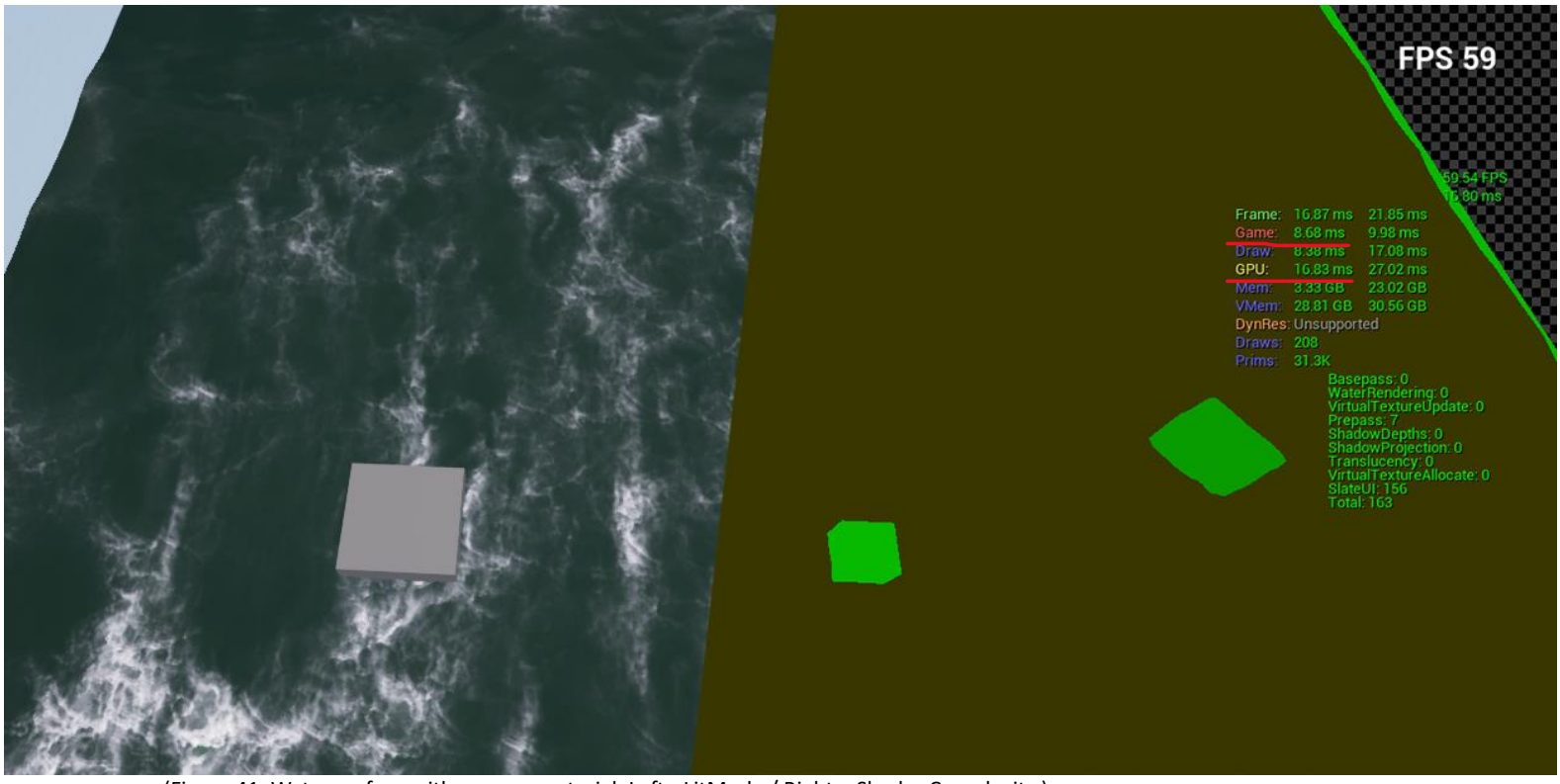


(Figure 39. Image showing the visual artifact present on the water surface.)

This is screenshotted but in the eye of the VR headset is even worse. To fix this issue I replaced this translucent material with an opaque one. Doing so I managed to achieve two things at the same time. Firstly, I fixed this bug and secondly, I improved the shader complexity. (Figures 40 & 41).



(Figure 40. Water surface with translucent material. Left - LitMode / Right – Shader Complexity.)

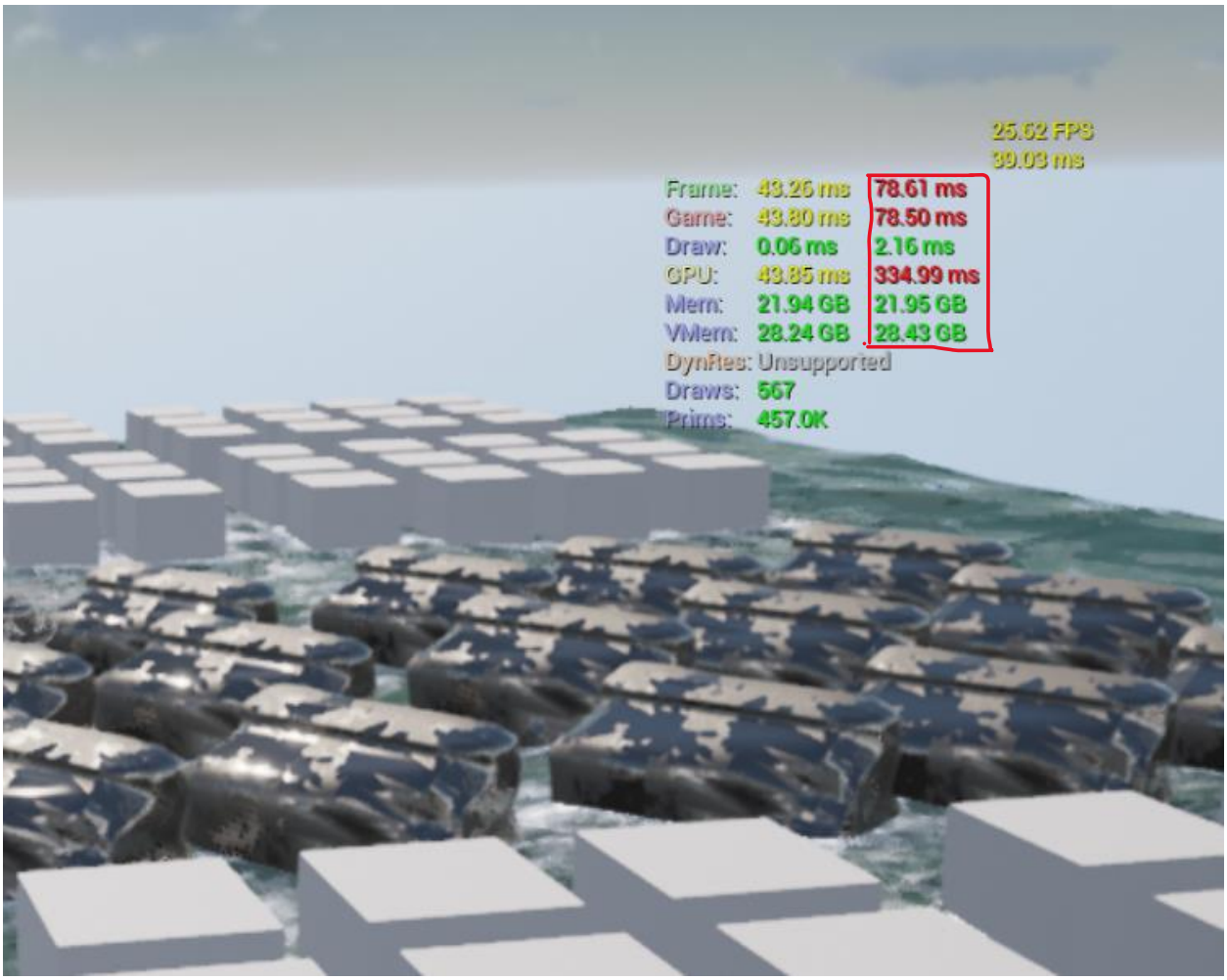


(Figure 41. Water surface with opaque material. Left - LitMode / Right – Shader Complexity.)

Although this change vastly improves the shader it is not visible in the FPS we can notice a decrease in the CPU of about 2ms decrease in the GPU processes and about 3ms in the Game Thread. I underlined

the two values in the above figures. These are small changes that may seem insignificant but when present in the levels all these small changes can add up and influence the performance greatly so that is why it is recommended to optimize whenever is possible.

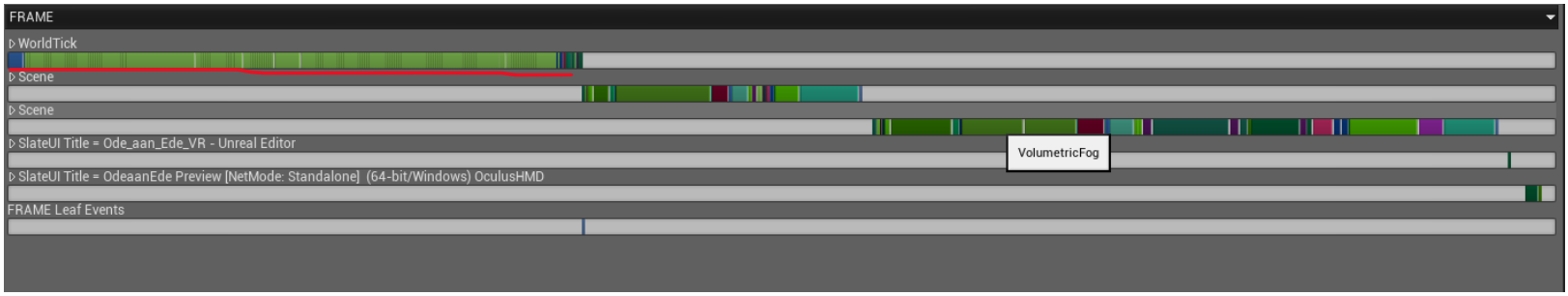
The next part which I wanted to test is the FluidNinja simulation input on performance. For this I duplicated the objects which interact with the water about 50 times. The hit on performance was quite drastic. (Figure 42) Of course, the scenarios in which the developers would need so many simulations are few but even if they need multiple simulations in one level, the developers of FluidNinja have thought of this and added a feature that stops any simulation from running when not visible.



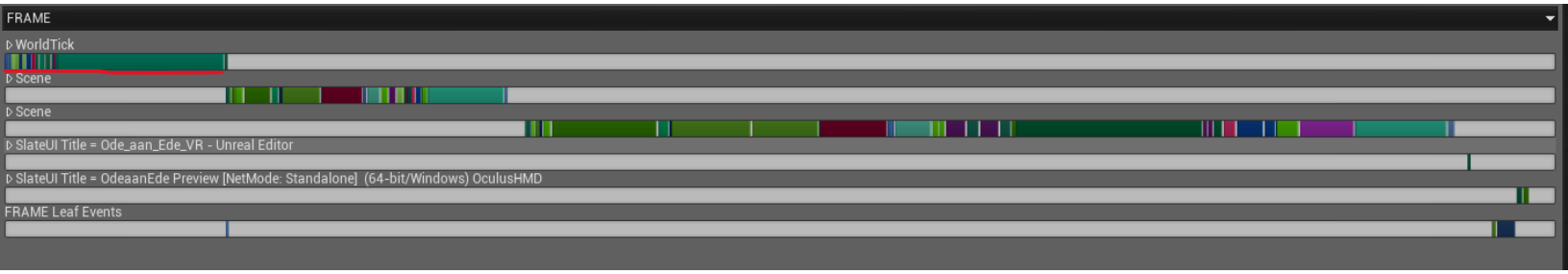
(Figure 42. Performance with 50 FluidNinja simulations running at the same time in a small area. CPU processes boxed in.)

We can see that the CPU processes have the biggest impact, represented in the second collum, which I boxed in. This simulations are ran by the FluidNinja Live Blueprint and the blueprints are processed by the CPU so that is why we see the biggest impact on performance of the CPU.

Another indicator of this performance impact is the GPU visualizer which shows all the processes of the GPU. The main part I am looking at is the WorldTick which shows the events that the GPU has to draw at every frame over and over. (Figure 43) We can see that when it runs only a few simulations that parts reduces more than 50%. (Figure 44)



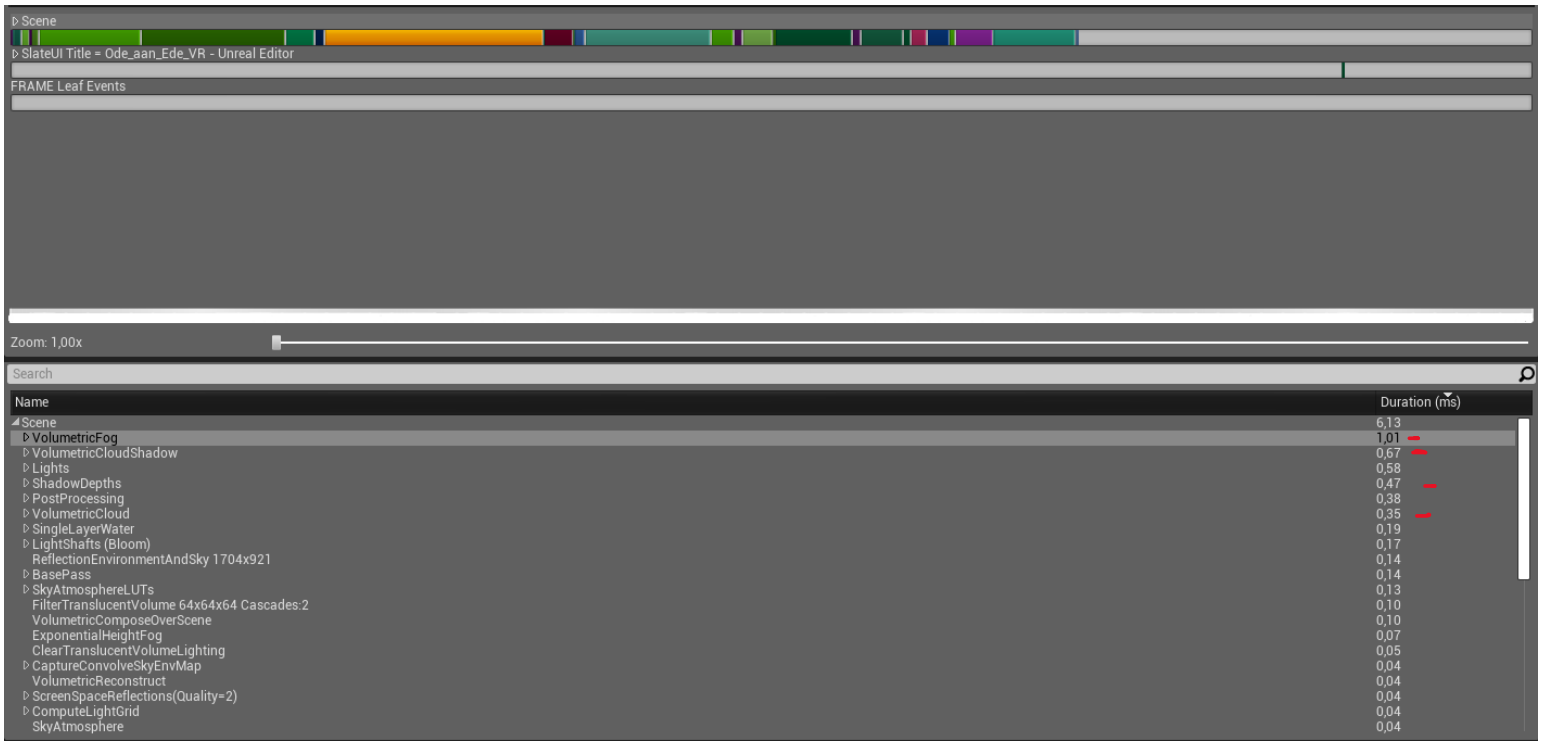
(Figure 43. GPU Visualizer when FluidNinja runs 50 simulations in a small area.)



(Figure 44. GPU Visualizer when FluidNinja runs 3 simualtions.)

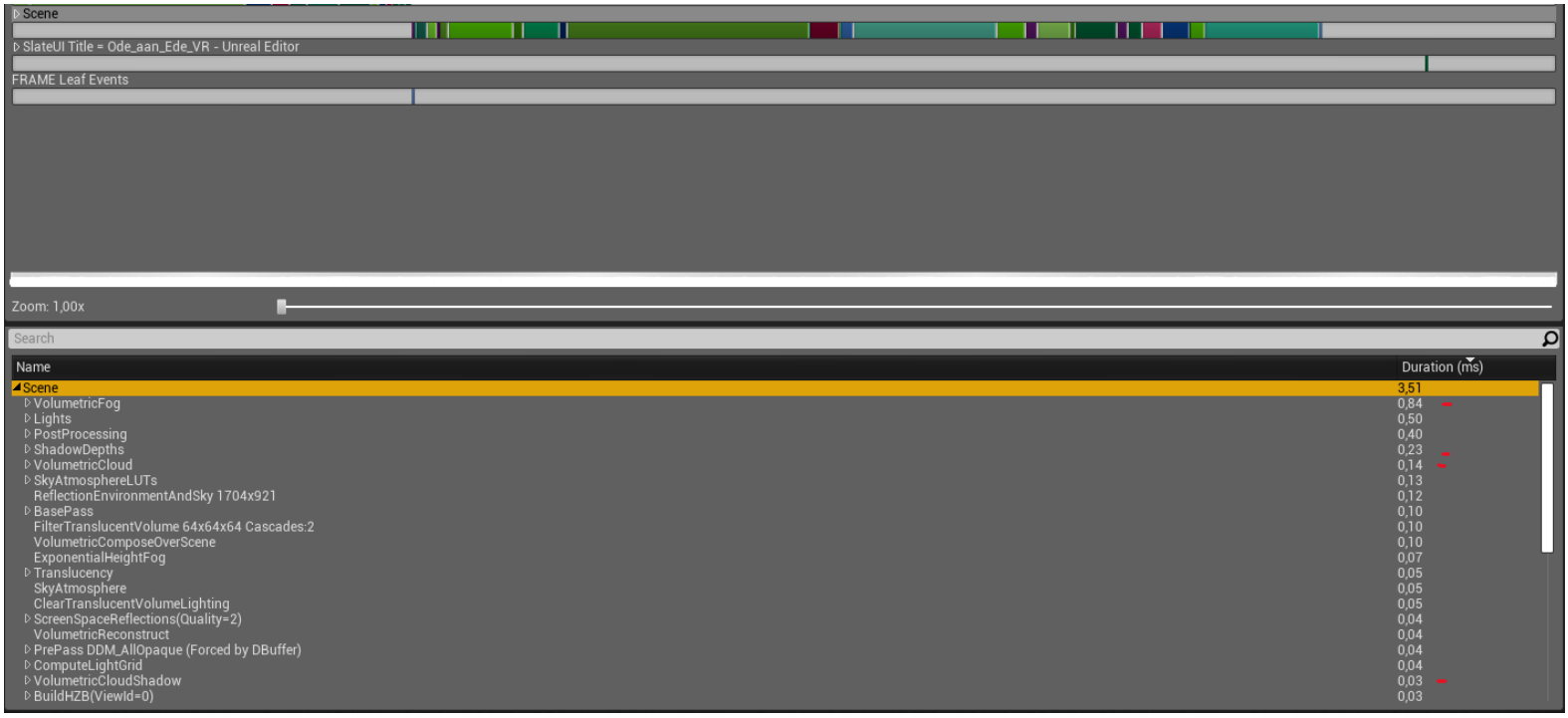
These two aspects combined result in the performance hit we see in figure 40, where the application can't run above 25 FPS.

Next, I wanted to take a closer look at the GPU visualizer and all the processes going on in the scene to determine which one is the heaviest on performance and if I can change something to improve this. This tool is neat as it shows the developer which processes exactly takes the longer and arranges them for easy reading. (Figure 45)



(Figure 45. GPU visualizer. In detail look at the scene processes.)

We can see clearer that the most “expensive” elements are the volumetric fog (1.01ms), followed by the volumetric cloud shadows (0.67ms). Other expensive elements are the ShadowDepths (0.47ms) and the Volumetric Clouds(0.35ms). To address these, I took a look at the settings of these elements to see what changes can be made to improve them. I managed to find some settings that control the quality of these elements, made small changes, and retested until I reached a better result. For the volumetric cloud shadows, I decreased the quality of the shadow map by 50%. This setting is self-explanatory. For the fog, I have decreased the scattering distribution. This means that the light will scatter less through the fog so it takes less to calculate it. For the shows, I decreased the resolution scale by 50% which means that the shadows will have less quality. For the clouds, I decreased a series of settings like view sample counts, reflections, and shadows sample. All of these settings control the quality of the clouds. These changes decreased the time these elements took to render. We see a drastic improvement when we look at the volumetric shadows and volumetric clouds and a slight change in the fog and shadows but still noticeable. (Figure 46)



(Figure 46. GPU visualizer. In detail look at the scene processes after the mentioned changes.)

It is visible that the VolumetricCloudShdows went from 0.67ms to 0.03ms which is a huge improvement, the Volumetric Fog went from 1.01ms to 0.84ms, the ShadowDepths went from 0.47ms to 0.23ms and the Volumetric Clouds went from 0.35ms to 0.14ms. These are noticeable improvements, and I am happy I managed to address these for slightly better render times.

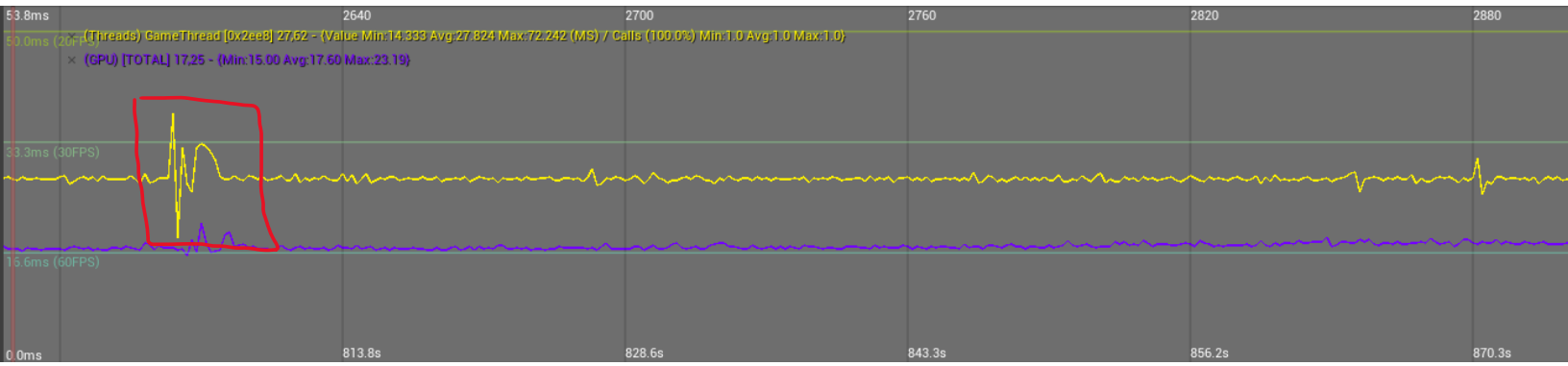
I chose not to change aspects like lights or post processing because these elements are different in each scene and should be edited separately in each scene. For this testing I am looking at the elements of my dynamic system.

The next step was to look at the Profiler and address the overall performance of the system and see if there are any spikes in the performance, what elements cause them and to address them.

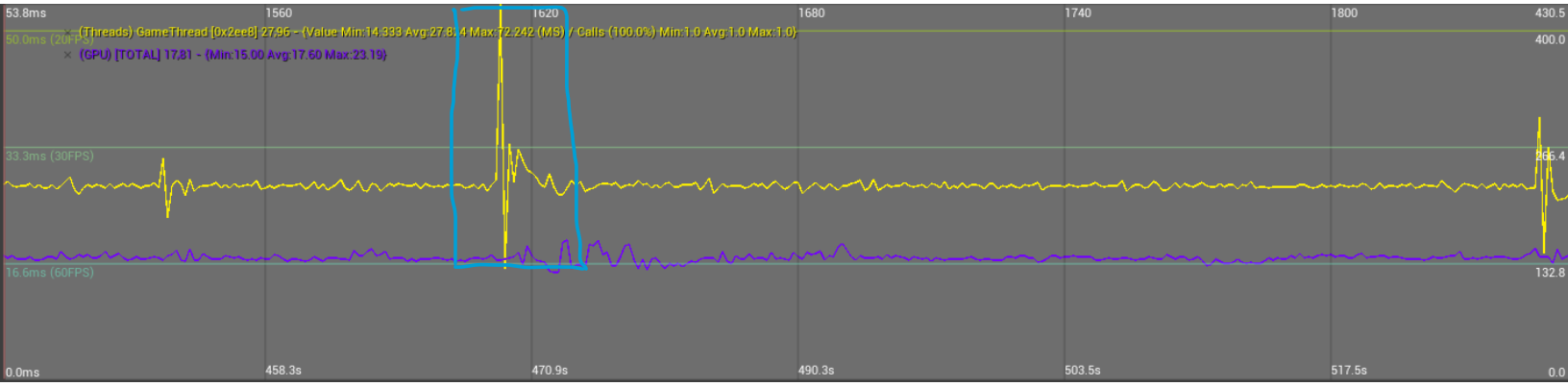
What I noticed from the get-go is that the scene has quite low FPS running constantly around 35-37 (Figure 47.). To visualize what could cause this I used the console commands “stat StartFile” and “stat StopFile” to record what is happening under the hood in my scene over a period of time. After this, I loaded it into the visualizer to see what processes caused this weak performance.

(Figure 47. Detail stats at run time. Top Right -> FPS and how much it takes to render one frame (red box). Bellow separate processes detailed individually.)

I have noticed that it runs quite stable without any big spikes in performance. (Figures 48 & 49.) We can notice that the GPU processes (purple line) are very stable and stay just below 60 FPS which is good but the Game Thread, which is the CPU processes (yellow line) mainly is stable as well but runs at around 35 FPS. While recording this I went through all weather states and all elements of the dynamic system to see if any of them would cause problems. I have noticed two spikes. One occurred when the snowy weather was loaded (red box in figure 48) because then I reinitialized FluidNinja to change the river material. Another is when the light changes from day to night (blue box in Figure 49). This is because the engine relights the entire scene. I don’t think these spikes are a big problem because they are not so big to freeze the screen so the player notices them and the bigger one, regarding the light, will never occur in this experience as all levels are shorter than the duration of the day, so the player will only play during the day.

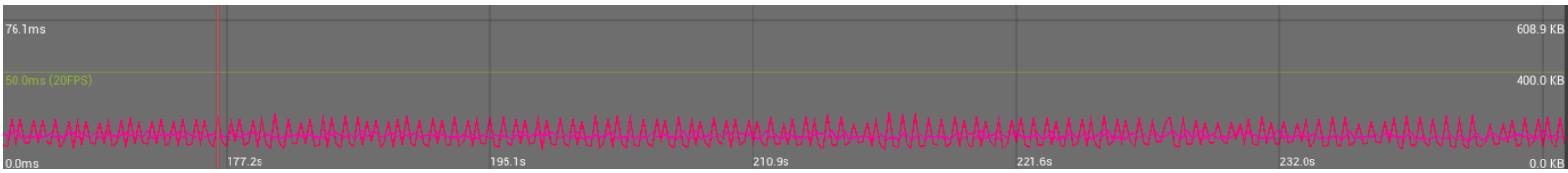


(Figure 48. The profiler inside UE4 shows the performance. Red box -> spike in performance when reinitializing FluidNinja)



(Figure 49. The profiler inside UE4 shows the performance. Blue box -> spike in performance when switching from day to night.)

What concerned me the most was the overall low performance. Because I saw that the CPU processes were responsible for this problem, and it showed that the scene was stable I deducted that not my system was at fault. To check this, I created an empty scene, and the performance was the same. This told me that it was the project itself that caused this problem, so I shifted my attention to the project settings. I came across Underscore's YouTube tutorial about VR optimization in which he discusses the project settings that can be adjusted to get a better performance. (underscore. (2022, March 1). UE4 Tutorial: Optimization (VR) [Video]) After I changed all the settings mentioned, one by one, and retested the scene I identified two settings which significantly improved the performance. I turned on the Forward rendering option. This changes the way the engine renders everything to a new and improved method. (Unreal Engine Documentation, Forward Shading Renderer. (n.d.)) The other setting is the default RHI (Render Hardware Interface.). "This is the C++ interface which the high-level platform-independent rendering code in UE4 uses to communicate with the several platform-dependent implementations that exist for Direct3D, OpenGL, etc." (What is Default RHI? Should I set it to DirectX12 or Default? (2020, September 28). Epic Developer Community Forums.) These two changes improved the performance from 35 to 60 -70 FPS. (Figure 50.)



(Figure 50. Profiler showing performance after the changes to the project.)

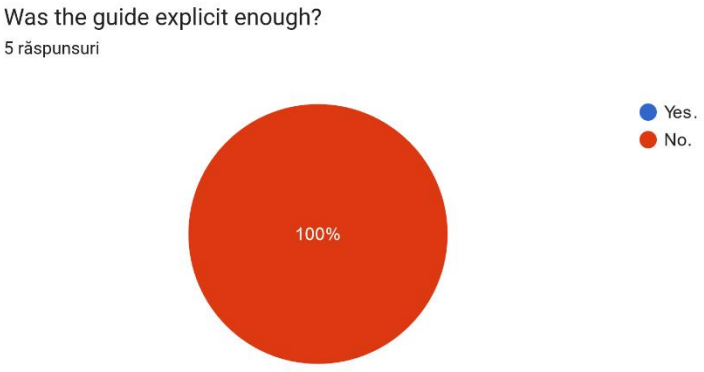
Although it fluctuates more, I rather would have the application fluctuate between 60 -70 Fps than be consistent at 35. To make sure these changes integrate properly with the project I will keep them as a separate push in the source control GitHub that we as a team use. If during further testing notice any problems, we can revert these changes easily.

This concludes the performance testing of the dynamic weather system. Through all these changes I managed to optimize the system to be at its best version and to make sure that it will not impact the performance of any scene in any project in any drastic mode.

4.2 Usability:

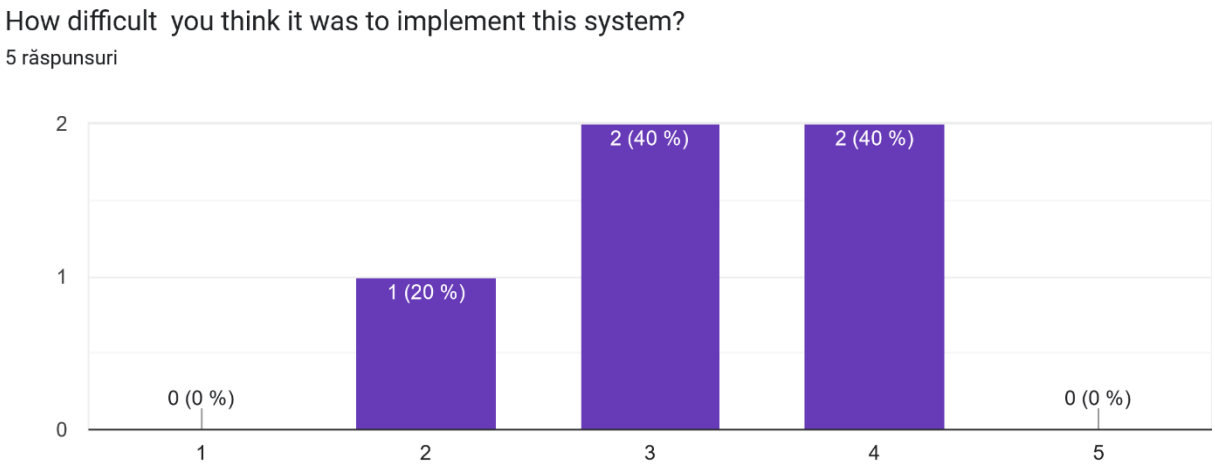
To assess if my system indeed would help developers, by cutting down production time and being implementable in future projects as well I created a test for my fellow teammates to do. (You can read the details of this test in the Appendix section: Usability Test) This text showed valuable insights into what needs to be changed to make sure future developers will be able to use my system. This was a qualitative usability test and therefore I aimed to get five testers. Research showed that for this type of testing five are enough as they will show 85% of the issues. (Budiu, R. (2021, December 3). Why 5 participants are okay in a qualitative study, but not in a quantitative one.) During the testing I also used the 'fly on the wall" method to observe the testers and see where they need help.

The first result I want to talk about is that the guide was not explicit enough (Figure 50.). There was some missing information and ambiguity, so I had to step in to help the testers. This proves that I need to write a detailed document that explains how the system works to make sure future developers feel comfortable implementing it. But it my guideness all of the testers were able to complete the tasks.

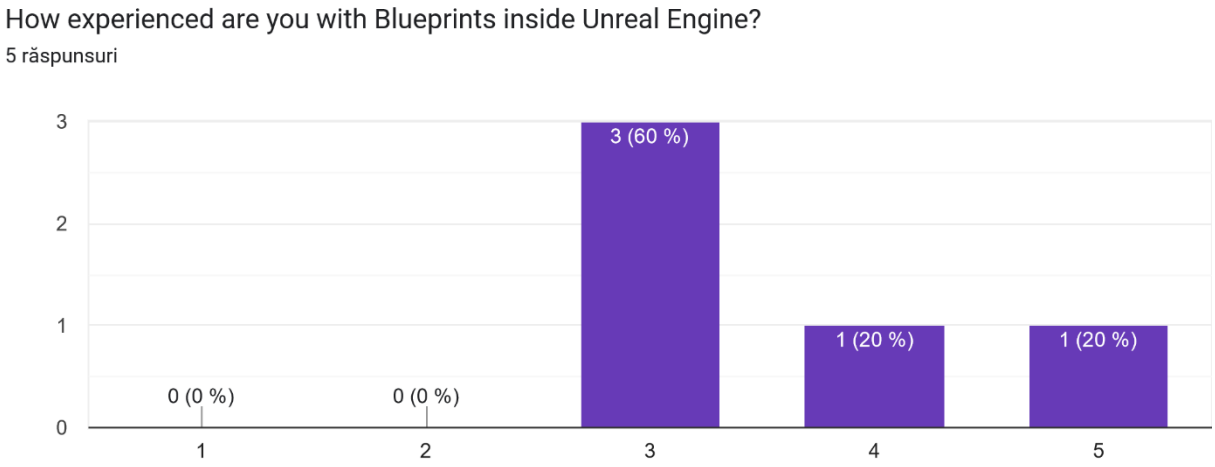


(Figure 50. Question in my survey – Was the guide explicit enough?)

The responses on how easy is to implement this system the responses are mixed (Figure 51). I think that with the right guide, the feedback would have been much more positive. Another factor for this is that the testers have different levels of experience with unreal and blueprints (Figure 52). I noticed watching them that some of them didn't find some default UE components very easy.

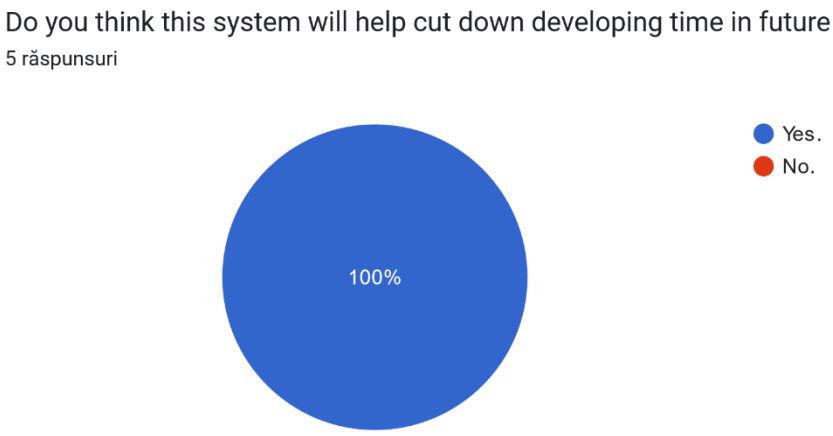


(Figure 51. Question in my survey - How difficult do you think it was to implement this system?) (1 -> very difficult /2 -> difficult / 3 -> neutral/ 4-> easy/ 5 -> very easy)

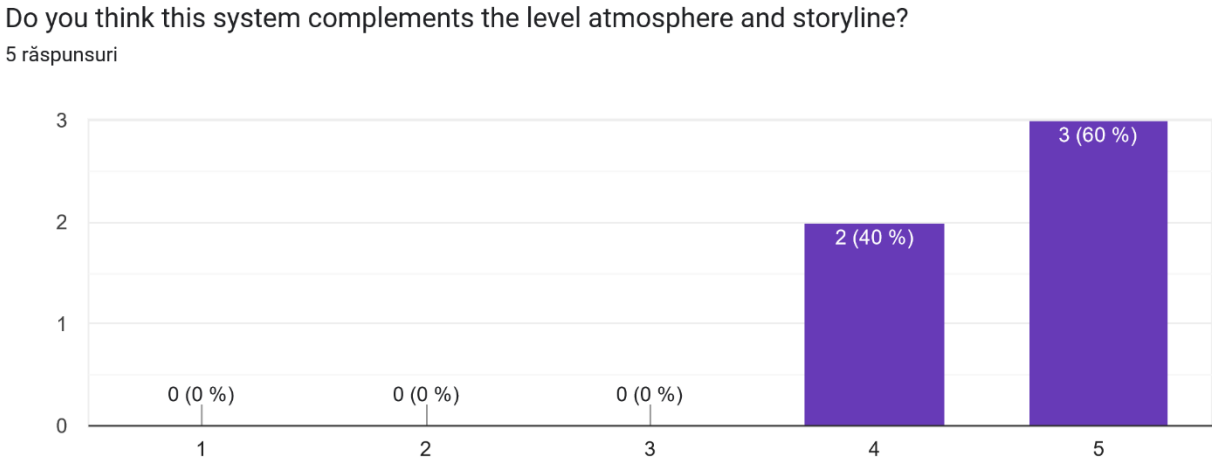


(Figure 52. Question in my survey – How experience are you with blueprints inside Unreal Engine?) (1 -> no experience at all/ 2 -> little experience/ 3 -> some experience/ 4-> experienced/ 5 -> very experienced)

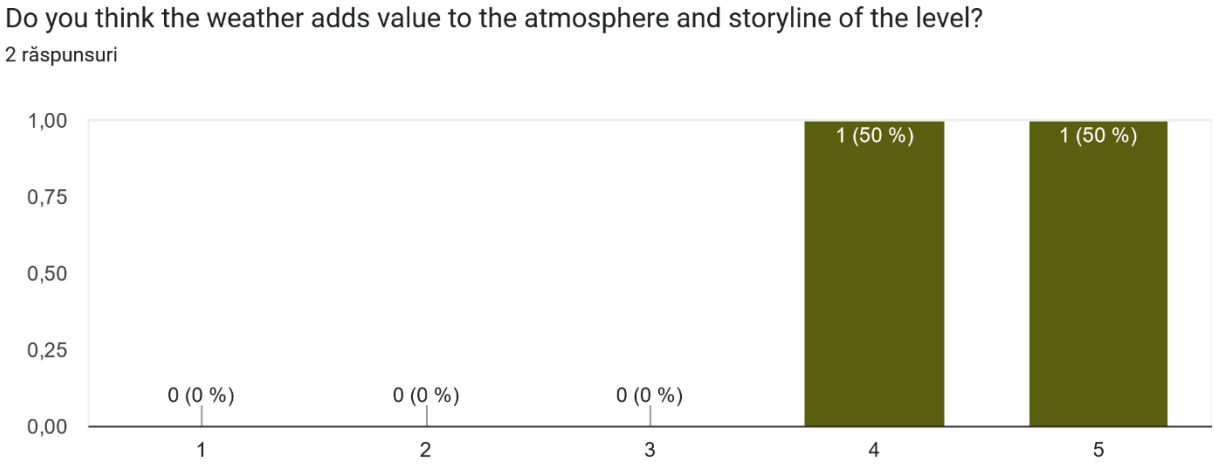
The system received very positive answers when the testers were asked about its ability to cut the development time (Figure 53) and if it complements the atmosphere and storyline of the levels (Figure 54). The fact that the system complements the atmosphere, and the storyline was proved by end users testers who tested the application in the last test did by the team in the office. (Figure 55.)



(Figure 53. Question in my survey – Do you think this system will help cut down developing time in future projects?)



(Figure 54. Question in my survey – Do you think this system complements the level atmosphere and storyline?) (1 -> Not at all/ 2 -> a little/ 3 -> neutral/4 -> it does/ 5 -> it's imperative)



(Figure 55. Question in Indietopia's survey – Do you think the weather adds value to the atmosphere and storyline of the level?)
(1 -> Not at all/ 2 -> a little/ 3 -> neutral/4 -> it does/ 5 -> it's imperative)

5. Conclusion:

In conclusion, the project I contributed to has not only turned out well but also holds significance for the community, successfully achieving its goal of attracting visitors to the museum and educating them about Ede Stall through an engaging, interactive experience. Having completed this process, I am confident in asserting that I have delivered a valuable product for both the company and the client. The dynamic weather system seamlessly integrates with the atmosphere and narrative of each level, enhancing the overall immersion of the VR application. It successfully fulfills the client's vision for the product, creating a sense of immersion through its realistic art style and profound connection to Ede Stall's songs. This system adds substantial value for Indietopia, providing an optimized VR solution that is easily implementable, reducing production time. This, in turn, positions the company with a competitive advantage for future projects, allowing developers the freedom to focus on artistic tasks and deliver more polished projects. Additionally, the success of FluidNinja Live as a plugin demonstrates its value, offering a comprehensive solution to multiple aspects of the project. Using this 3rd party plugin was the right choice as it really enabled me to add an extra layer of realism and interaction to the system. The testing I did

6. Recommendations:

I have a couple of recommendations for the company regarding this project and future endeavors.

Firstly, I advocate for exploring additional optimization measures for the project. This optimization process should ideally take place after the completion of all scenes, ensuring that everything is running in a thoroughly efficient way on the required hardware. The proposed solutions that I identified and suggested should be rigorously tested and implemented if they indeed enhance the overall user experience.

Furthermore, I highly recommend leveraging FluidNinja more extensively, as I believe I've only scratched the surface of its capabilities in this project. While I utilized it to drive volumetrics and facilitate interactions with fluids, FluidNinja has the potential to offer much more in terms of visual effects. Its versatility extends to a wide range of use cases that I wasn't able to fully explore in the context of this particular project.

7. Reflection:

Reflecting on the project, I am pleased with how it unfolded. I successfully achieved my set goals, completing all assigned tasks within the team-organized sprints, which usually spanned two weeks. Additionally, I occasionally added extra tasks during the sprint. This was made possible by maintaining a rigorous discipline throughout the week, planning all my tasks in ClickUp, and updating them as I progressed. This approach ensured that I always knew what needed to be done, allowing me to estimate the time required for each task and prioritize effectively to avoid falling behind.

Balancing work and free time was crucial, especially as I worked entirely remotely during this period. I utilized Friday afternoons for my personal project, aligning with the team's agreed-upon time for individual growth. This practice helped me disconnect from project challenges and return after the weekend with a fresh mind and view. It was satisfying to prove to myself that I could maintain such discipline for an extended period to achieve my goals.

Reviewing the delivered outcomes, I am very satisfied. I successfully learned and implemented two new tools, Unreal Blueprints and FluidNinja Live, integrating them into a cohesive system that adds value not only to this project but also to others. I successfully implemented the system into existing scenes and those initiated by the team at the start of the internship. This was accomplished by making use of every resource available, including various research methods such as desk research and interviews. Assistance from fellow developers like Justin and Kristiyan from my team, along with guidance from Andras Ketzner, the developer of FluidNinja, was invaluable and significantly eased the development process.

While content with the overall project, I acknowledge areas for improvement. Conducting testing, especially performance and usability tests, earlier in the development process could have been more beneficial. This would have provided a baseline for comparison, allowing for early identification of potential performance-heavy elements and necessary adjustments. Involving fellow developers in usability testing at an earlier stage could have highlighted the need for a detailed setup document, ensuring readiness for subsequent testing sessions. Incorporating more long-term milestones and checkpoints throughout the project would have allowed me to achieve this. These insights contribute to my ongoing learning and will be incorporated into my approach for future projects.

An area I aspire to enhance in future projects involves dedicating more time to the early stages for a systematic and thorough initial research phase. To achieve this, I recognize the importance of investing additional time in researching methods to achieve the desired outcomes and delving deeper into understanding the tools I plan to employ. While the early development of prototypes was beneficial, given the scale and complexity of the project, I believe I could have allocated one or two more weeks to research. This would have allowed me to be better prepared for various situations, eliminating the need to revisit documentation mid-development. Such an approach would have facilitated a more streamlined and linear development process.

8. Reference List:

Before you continue to YouTube. (n.d.-a). https://www.youtube.com/@AndrasKetzer

Before you continue to YouTube. (n.d.-b). https://www.youtube.com/playlist?list=PL78XDi0TS4lGXKflD2Z5aY2sLuIln6-sD

Before you continue to YouTube. (n.d.-c). https://www.youtube.com/playlist?list=PLVCUepYV6TvOrOfQVLMCxl_JoU_cIkK8P

Ben Cloward. (2020, April 23). Water Ripples Shader - UE4 Materials 101 - Episode 23 [Video]. YouTube. https://www.youtube.com/watch?v=r68DnTMeFFQ

Blueprint Editor reference. (n.d.). Unreal Engine 4.27 Documentation. https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Blueprints/Editor/

Blueprint Overview. (n.d.). Unreal Engine 4.27 Documentation. https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Blueprints/Overview/

Blueprint Splines. (n.d.). Unreal Engine 4.27 Documentation. https://docs.unrealengine.com/4.27/en-US/BuildingWorlds/BlueprintSplines/

Blueprint Visual Scripting. (n.d.). Unreal Engine 4.27 Documentation. https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Blueprints/

Brown, V. (2023, February 9). Modern Double Diamond design: Rethinking a classic design process - LogRocket Blog. LogRocket Blog. https://blog.logrocket.com/ux-design/modern-double-diamond-design/

Budiu, R. (2021, December 3). Why 5 participants are okay in a qualitative study, but not in a quantitative one. Nielsen Norman Group. https://www.nngroup.com/articles/5-test-users-qual-quant/

Dean Ashford. (2019, February 27). UE4 - tutorial - Dynamic Weather! (Request) [Video]. YouTube. https://www.youtube.com/watch?v=uFJmiLpfpEg

FluidNinja LIVE in Visual Effects - UE Marketplace. (n.d.). Unreal Engine. https://unrealengine.com/marketplace/en-US/product/fluidninja-live

FluidNinjaLiveIssuesFAQ.pdf. (n.d.). Google Docs. https://drive.google.com/file/d/17oVPVEoaW6Y6YKNISr4S0uUJY4_Yx_FM/edit

FluidNinjaLiveManual.pdf. (n.d.). Google Docs. https://drive.google.com/file/d/1I4dglPjeXLcNkSGxGok8sQCy59qgYcF9/edit

FluidNinjaLiveSim.pdf. (n.d.). Google Docs. https://drive.google.com/file/d/1jeICoiCKhnAbl2_LBkNrcW43YfbUmIKt/edit

FluidNinjaLiveUI.pdf. (n.d.). Google Docs. https://drive.google.com/file/d/1jpr3cuNvwVUNc1Fj2nPE_9jyLnh-JVmw/edit

Forward Shading Renderer. (n.d.). https://docs.unrealengine.com/5.3/en-US/forward-shading-renderer-in-unreal-engine/

Home - Indietopia | indie games, accelerator, game development. (2023, November 17). Indietopia. https://indietopia.org/

Material Parameter collections. (n.d.). Unreal Engine 4.27 Documentation. https://docs.unrealengine.com/4.27/en-US/RenderingAndGraphics/Materials/ParameterCollections/

Mathew Wadstein. (2015, November 15). WTF is? a dynamic material instance in Unreal Engine 4 [Video]. YouTube. https://www.youtube.com/watch?v=IVKVXjaT6QM

McCombes, S. (2023, July 18). How to write an abstract | Steps & Examples. Scribbr. https://www.scribbr.com/dissertation/abstract/

Niagara Fluids Quick Start Guide. (n.d.). https://docs.unrealengine.com/5.3/en-US/niagara-fluids-quick-start-guide-for-unreal-engine/

Niagara Overview. (n.d.). Unreal Engine 4.27 Documentation. https://docs.unrealengine.com/4.27/en-US/RenderingAndGraphics/Niagara/Overview/

Performance and profiling. (n.d.). Unreal Engine 4.27 Documentation. https://docs.unrealengine.com/4.27/en-US/TestingAndOptimization/PerformanceAndProfiling/

PrismaticaDev. (2021, September 19). Single Layer water | 5-Minute Materials [UE4/UE5] [Video]. YouTube. https://www.youtube.com/watch?v=KLl3PZeupFM

PrismaticaDev. (2023, February 8). Material Parameter Collection | 5-Minute Materials [UE5] [Video]. YouTube. https://www.youtube.com/watch?v=J2Qf5v9_uSY

Sound attenuation. (n.d.). Unreal Engine 4.27 Documentation. https://docs.unrealengine.com/4.27/en-US/WorkingWithAudio/DistanceModelAttenuation/

Sound Cue editor. (n.d.). Unreal Engine 4.27 Documentation. https://docs.unrealengine.com/4.27/en-US/WorkingWithAudio/SoundCues/Editor/

Sound cue reference. (n.d.). Unreal Engine 4.27 Documentation. https://docs.unrealengine.com/4.27/en-US/WorkingWithAudio/SoundCues/NodeReference/

The DiNusty Empire. (2020, October 10). Making Volumetric Clouds in Unreal engine 4.26 [Video]. YouTube. https://www.youtube.com/watch?v=oWE_vYiQMoE

underscore. (2018, May 9). UE4 Tutorial: Night Scene with Moon [Video]. YouTube. https://www.youtube.com/watch?v=thF80XvnqMo

underscore. (2022, March 1). UE4 Tutorial: Optimization (VR) [Video]. YouTube. https://www.youtube.com/watch?v=Jvze2nnR_-Q

Volumetric clouds in UE 4.26 do not render in VR. (2021, March 9). Epic Developer Community Forums. https://forums.unrealengine.com/t/volumetric-clouds-in-ue-4-26-do-not-render-in-vr/153838/8

Water. (n.d.). Unreal Engine 4.27 Documentation. https://docs.unrealengine.com/4.27/en-US/BuildingWorlds/Water/

What is Default RHI? Should I set it to DirectX12 or Default? (2020, September 28). Epic Developer Community Forums. https://forums.unrealengine.com/t/what-is-default-rhi-should-i-set-it-to-directx12-or-default/472055

What is design thinking & why is it important? | HBS Online. (2022, January 18). Business Insights Blog. https://online.hbs.edu/blog/post/what-is-design-thinking

William Faucher. (2021, May 25). Volumetric Cloud Secrets [Unreal Engine 4 & 5] works in UE5! [Video]. YouTube. https://www.youtube.com/watch?v=yolGEIrhu0s

9. Appendix:

9.1 Testing Plans:

9.1.1 Indietopia Test Procedure:

This is a testing plan made by Justin Krogman to use to test with the end users.

1. Before the arrival of the testing participants, ensure that there is a stable build of the project ready and that a single Oculus Quest 2 headset is prepared with an AirLink connection established so that the testing can begin without issue or further setup being required.

2. Upon the arrival of the testing participants to the office we will welcome them and thank them for their time and interest in our project. We will offer tea and coffee to participants and x begin with small talk to break the ice and establish a relaxed, friendly, and open environment.

3. We will have the participants take a seat and begin with a general introduction of Indietopia, the project, and how to use the Oculus Quest 2 VR headset.

    a. **For Indietopia:** Indietopia is a hub, accelerator, and publisher located here in Groningen. We love games and innovative technologies. We pave the way for the northern digital creative industry to blossom, bridging the gap between studies and the industry.
    b. **For Ode aan Ede:** "*Ode aan Ede*" features a virtual reality experience where a user can step into a world inspired by the works of local musician Ede Staal, and explore his impressions of the Dutch landscape, life, and language. A traditional Dutch farm workers house acts as a central hub point in the experience. Users can navigate between different levels from this location and explore three different experiences. Each level contains different types of gameplay, locations, and various songs from Ede himself.
    c. **For the Oculus Quest 2 Headset:** The Oculus Quest 2 VR headset is very easy to set up and begin using. For this test we will assist you with putting on and taking off the headset, as well as the controllers.

        i. Place headset on head, adjust top and rear straps to achieve a comfortable fit.
        ii. Adjust side brackets so that the center of each lens is in line with the center of your pupil to ensure you have clear and unblurred image quality.
        iii. Receive each controller, putting your hand through the safety lanyard first before gripping.
        iv. You're all set for virtual reality!

4. Ask all participants if there are any questions or concerns before the testing process begins, if there are none begin the testing process and inform all participants that testing will now begin.

5. Separate participants so that only one user is testing the application and no other participant is observing their progress or the testing procedure.
    a. **Special Note:** At this point, make a request to all participants to not discuss the testing process or details about the application with participants that have not tested the

application yet. Have participants that have completed testing sit in a different area from those that have not tested yet to prevent discussion.

6. Ensure the experience is running in UE and there is an Airlink connection to the headset and that the sound is turned on in the Oculus headset, assist the participant with equipping their VR headset. Once the testing participant has the VR headset equipped and the experience is loaded onto the headset begin with testing user scenarios in order.

7. User Tasks:
    a. **Task 1 (Onboarding, Navigation, Wayfinding, Branched Spline Movement)**
        i. Please follow the path towards the house to become familiar with the basic controls of the game. (Observe the user, if they are struggling with progressing give them hints and assistance with spline movement and gazeview controls)
        ii. When the user approaches the gray cube, please ask them to gazeview at it to transition to a new spline path.
        iii. Once the user has transitioned to a new spline path, please ask them to navigate along it to a different position.
        iv. Once the user is at a new position on the spline path, please ask them to look at the gray cube located on the original main spline path to transition back towards it.
        v. Please ask the user to return to the branched spline path and navigate along it to a new position one more time, and then to return to the main spline path again.
        vi. Once the user is on the main spline path, please ask them to continue on their journey forwards through the level.

    b. **Task 2 (Progression Systems, Navigation, Critical Path - Scene 6)**
        i. Please explore the level and follow any instructions or clues you encounter, there are several different objectives to complete within this level. Observe the player, if the player is struggling with making progress provide hints towards gameplay objectives.
            1. Hint: Gazeview at the toy ducky in the living room of the house.
            2. Hint: Gazeview at the bell on the first floor of the barn.
            3. Hint: Gazeview at the cane on the small bridge beside the church.
            4. Hint: Gazeview at the flower bouquet in the church graveyard.
            5. Hint: Gazeview at the grandparents sitting on the bench by the dike.
        ii. Congratulations on completing the level! You will be automatically transported back to the HUB house where we can begin with the next task scenario.

    c. **Task 3 (Progression Systems, Optional Objective Items - Scene 6)**
        i. During the regular testing of the level, observe the user and see if they interact with any optional objective items. If the user has not collected any optional objective items before key item3 (Cane), please give them some hints to interact with the items that are still available to them.
            1. Hint: Please Gazeview at the highlighted object to gain additional story information and dialogue.

d. Task 4 (Dynamic Weather System, Environmental Storytelling - Scene 6)

i. Once the user has reached the small bridge where Key Item 3 (Cane) is located, please ask them to pay attention to the canal water and sky while the time change effect is happening and speak their thoughts and opinions on it using the think-aloud method.

4. Task 5 (Accessibility Settings, Colour Blindness, Localization - Scene 6)

i. Once the user has completed all previous task scenarios and are content with their exploration of the level, please ask the user to open the journal by pressing the A button and to navigate through the menu options, and explore its contents.
ii. Please choose the menu option to “Reset Level”.
iii. Please open the journal again and change the language settings to a different language.
iv. Please open the journal again and change the color blindness settings to a different setting.

8. Once all tasks have been completed, inform the user that the testing session is now over, once they are ready to end the experience you may begin the question phase of the process.

9. Questions to ask: Please direct the participant to fill out this Google Docs form in its entirety (**In English**) and to the best of their ability.
Link:

10. Once all questions have been asked, ask the participant if they have any additional comments or remarks to make regarding the application or overall testing process.

11. Once the participant is done with their closing remarks, thank them for their time and interest again. Explain that the testing process is now over, and request that they move towards a seating area where all participants that have completed testing will be seated.

12. Get the next participant and begin the testing procedure from the beginning of “Step 5”.

13. Once all participants have completed the testing process they should all be seated together, we will thank the group for their time and interest once again and ask if there are any final remarks, comments, or pieces of feedback that they would like to give about Indietopia, the project, or the usage of the Oculus Quest 2 headset.

14. Once all participants have given their final remarks, the meeting is concluded. All information and feedback should be collected into a central location for future analysis and discussion within the team.

9.1.2 Performance Test 1:

This self-conducted test was executed using a borrowed VR headset in my personal space. The key details of the test are outlined as follows:

Test Setting:

Location: My personal space

Tester: Myself

Hardware Requirements: PC, VR headset: Oculus Quest 2

Objective: The primary aim of this test is to assess the performance of the developed systems in a VR environment and identify potential bugs.

Engine Preparations: Duplicate each system ten times for stress testing. Implement a simple FPS counter as a widget to monitor performance.

Hardware Preparations: Connect the VR headset to the PC using the Oculus app.

Upon completing the necessary preparations, a structured list of actions was created to guide the testing process:

1. Walk around the test scene adjacent to each system, monitoring the FPS.
2. Utilize Unreal's optimization buffers to assess shader complexity.
3. Examine each eye individually to identify possible bugs.
4. Load the blockout level and conduct two walkthroughs to identify bugs or performance issues in the systems.
5. Document any noteworthy findings.

Post-test procedures involve reviewing the findings. If any bugs are identified, efforts will be made to find and implement a solution. If resolution proves elusive, the issue will be documented and promptly discussed with the team. A comprehensive description and discussion of all findings will be presented in the testing section of this document. (Refer to Testing Section - Performance)

9.1.3 Performance Test 2:

This self-conducted test was executed using a borrowed VR headset. The key details of the test are outlined as follows:

Test Settings:

Location: Own room.

Tester: Myself.

Hardware requirements: PC, VR headset: Oculus Quest 2

Objective: The primary aim of this test is to assess the performance of the developed systems in a VR environment and identify potential bugs. To use profile tools to assess which areas of the system may cause performance issues.

Engine Preparations: Make sure that the testing scene is working as intended without any errors in the message log. Make sure that the levels 4,5 and 6 also run without errors.

Upon completing the necessary preparations, a structured list of actions was created to guide the testing process:

1. Playthrough all 3 levels as it is intended for the end-users.
2. Record the performance of each scene using the Unreal Engine tools.
3. Use the optimization view modes to detect any performance-heavy elements.
4. Make small changes to individual elements to see the impact on performance.
5. Look at the memory usage of the Blueprint itself and see if there are any improvements needed.
6. Document the changes and see which ones improve the performance.

All findings together with potential changes will be showcased and discussed in detail in the testing section of this report at: Performance

9.1.4 Usability Test:

To test the usability of this system I created an example scene with an atmosphere set-up, volumetric clouds, and a water plane. I will ask my teammates to set up in this scene the dynamic weather blueprint to see how easily other developers could set up this system in other scenes.

I have written a short guide as they need to know how to link Fluid Ninja with the elements but other than that they need to figure out how to do it on their own.

Guide for the testers:

Link FluidNinja Live with the component you want to control.

1. Click on the FluidNinja Component (red N) and in the details panel in the components section go to the FluidNinjaComponent.

2. Then find LIveGeneric and at the Secondary Output Material add one material. (If you search for volume clouds/water you should find multiple examples.) Next at “Apply 2nd Out Mat to Actors with Tag” (both water and clouds) and ‘Apply 2nd Out Mat to Components with Tag” (here only for the clouds) type a name of your choice.

3. Then go to the element and in the detail panels at the tag section type the same tag you wrote in the fluidninja component.

Tasks for the testers:

1. Bring in the Dynamic Weather State Machine Blueprint. (NewContent>Blueprints)

2. Bring in all additional elements like rain and snow. (You can skip the river and fog as it will work without them as well.) (Rain and snow elements are particle systems called NS_Rainfall & NS_Snowfall -> NewContent>NiagaraParticles)

3. Make sure all references are set - Element Ref and Atmosphere Ref. The SkyLightNight and LightRefNight together with River and EasyFog need to be empty as we don't use them for this test.

4. Link the FluidNinja elements with the volumetric clouds and water. (Follow the guide provided)

5. Make sure the elements work.(Cycle through the weather states by pressing the numbers 1-5)

6. "Draw" on the water to see the interaction. "Draw" on the trace mesh (black plane) of the clouds to see the interaction.

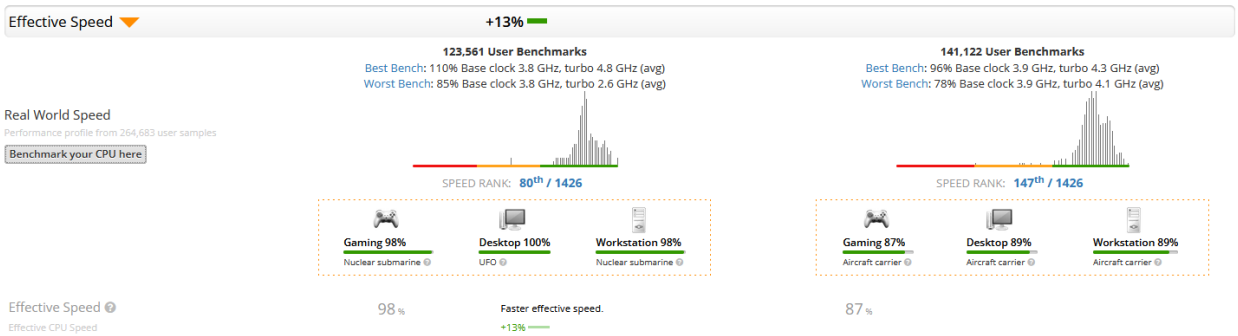
6. In the details panel of the Blueprint change the public variables from the sunny setting to change it to an overcast setting.

To measure the success of this I created a survey with a couple of questions for the testers. You can see the survey in detail here: https://forms.gle/9cuGHaRz8LXX3dgJA

The results of this survey will be discussed in the usability section of the testing part of this document: Usability

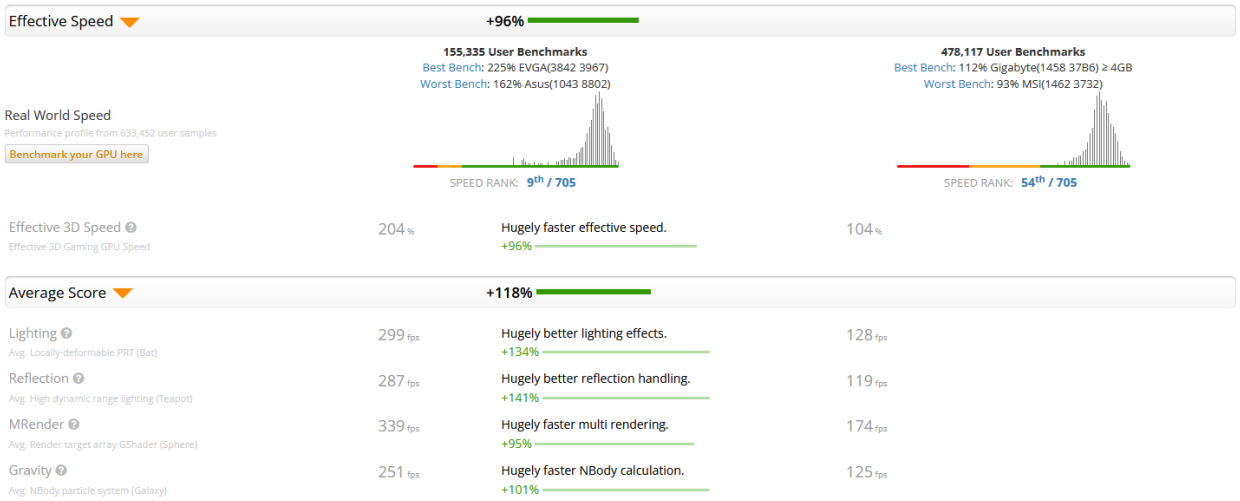
9.2 PC Hardware comparison:

CPU Comparison:



(Figure 37. Comparison between the two CPUs using UserBenchmark.)

GPU Comparison:



(Figure 38. Comparison between the two GPUs using Userbenchmark.)

9.3 Interview questions:

Questions I asked Kristiyan Petrov, to get a better understanding of the dynamic system requirements and their expectations.

Q1: Why does Indietopia need such a system?

Q2: Will it be developed to be used only in this project or should it be integrable in future projects as well?

Q3: What are my responsibilities? What part of the development will I be in charge?

Q4: Do you think there is a better method than blueprints to create the functionality of this system?

Q5: What is the desired performance for this system, especially in this project?

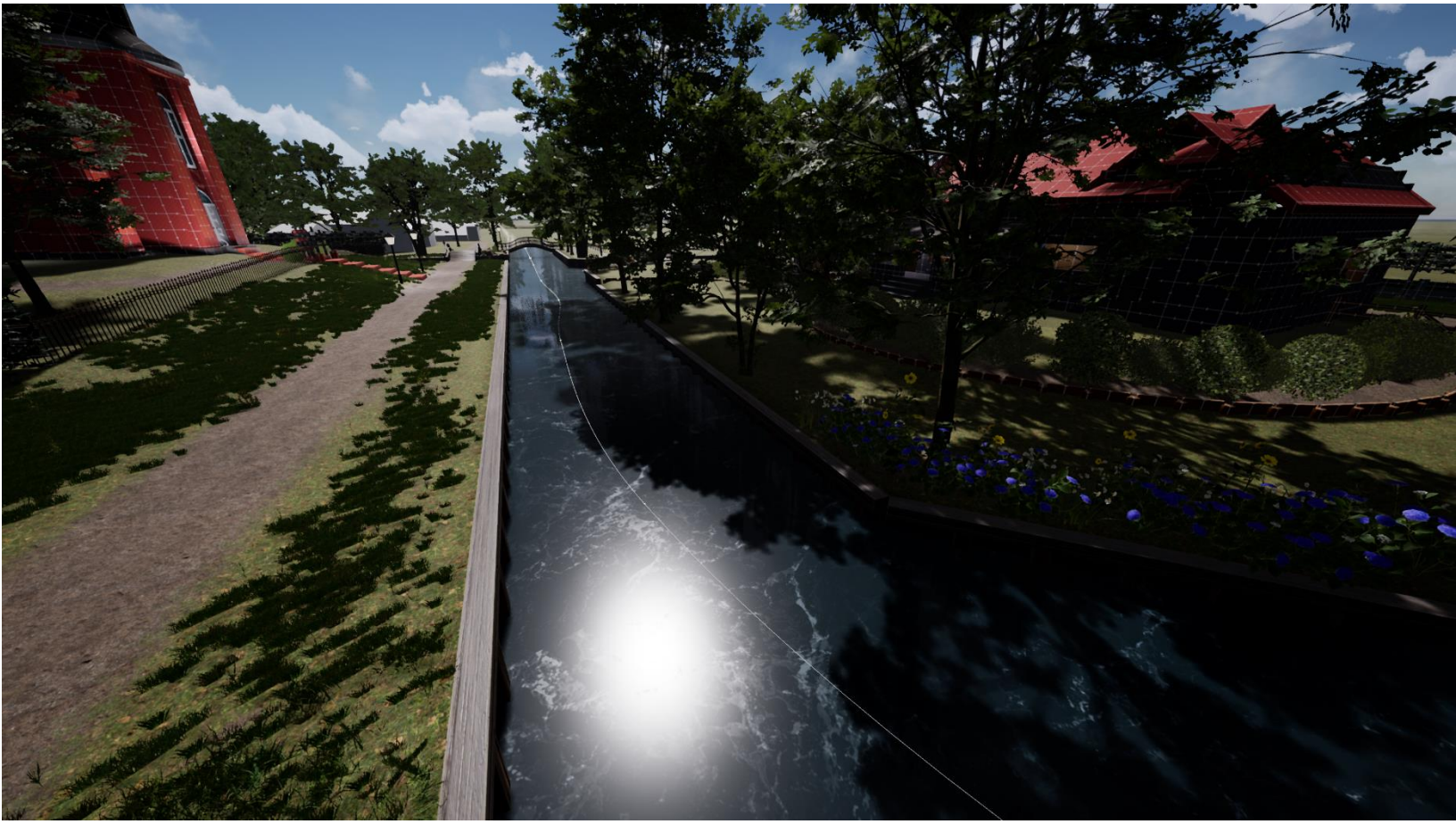
Q6: What do you expect this system is capable of?

Q7: In which levels of this project will this system be implemented?

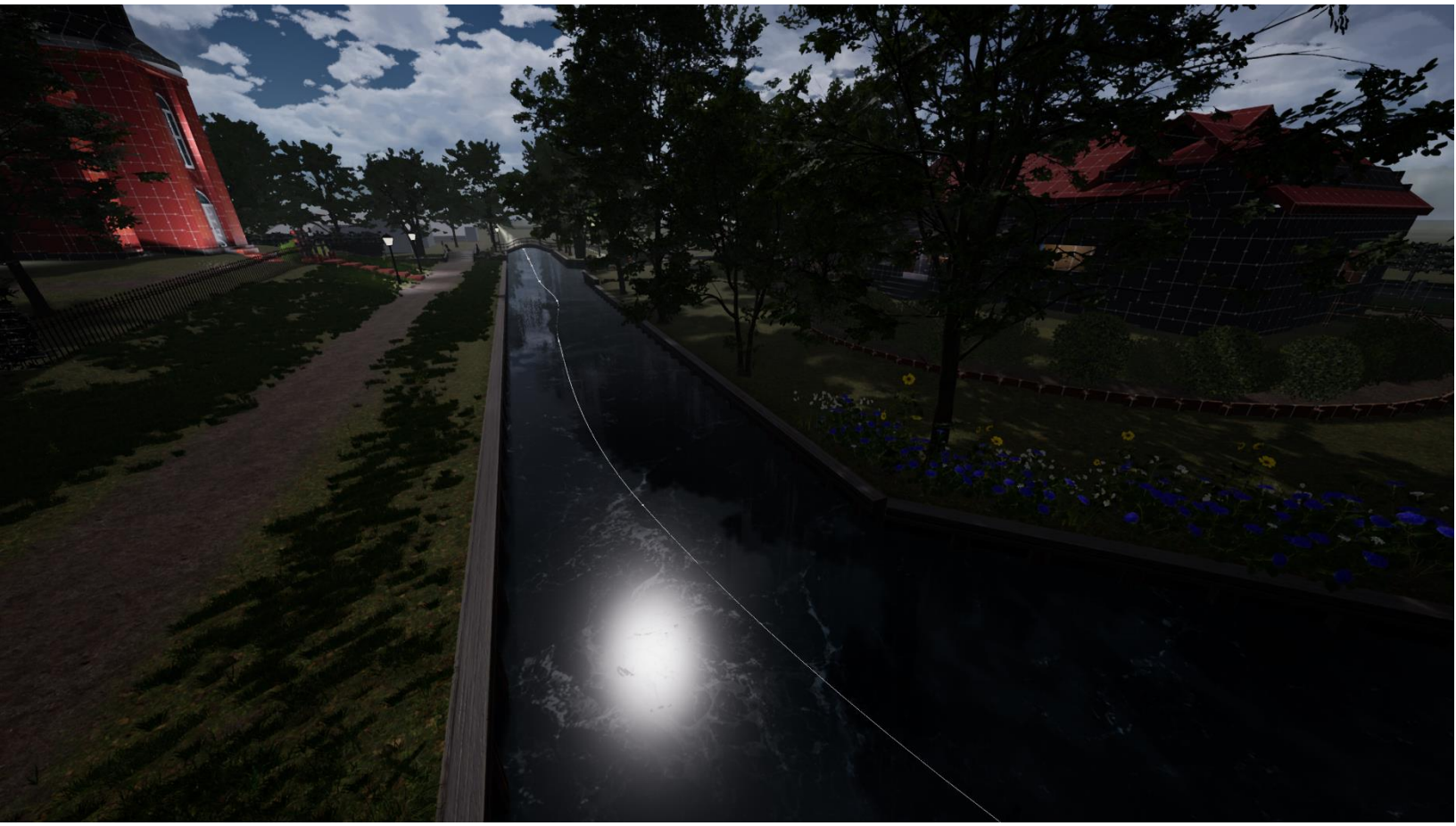
Q8: What does the client expect from this system?

The key takeaways from this interview are: Indietopia needs this dynamic system to improve the experience of the end user while using this application by increasing the immersion of the player. This system will be mainly used for this VR project but it needs to be easily implemented in other projects as well. So futureproof is an important aspect. I will be in charge of creating the system from scratch and implement it in the levels. I will also research what other tools like 3rd party software are required for this system. Another developer will reference it in the other system to integrate seamlessly with the whole experience. Blueprints will be used to create the functionality because all other systems are based on blueprints as well, so they need to communicate together. This system needs to not impact the performance so the levels become unplayable. The system needs to be able to control the atmosphere in the level by allowing the developers to create different weather types. This system will be implemented in all levels of this project priority having levels 4,5 and 6. The client has no technical expertise so what they expect is the levels to be immersive and to illustrate the story of each level through the atmosphere. So the developers job is to come up with the best solution to achieve this.

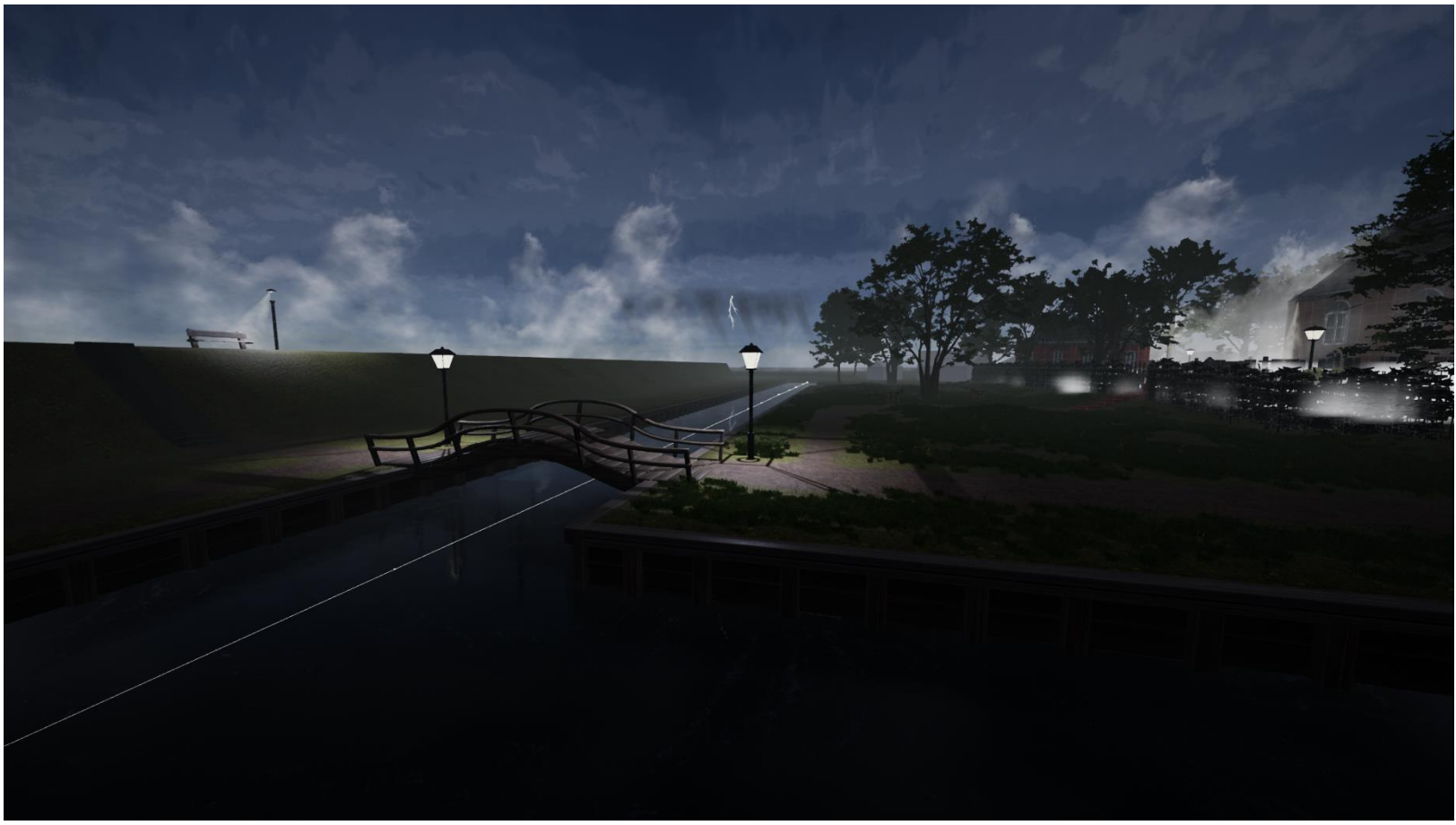
9.4 Final Product Showcase:



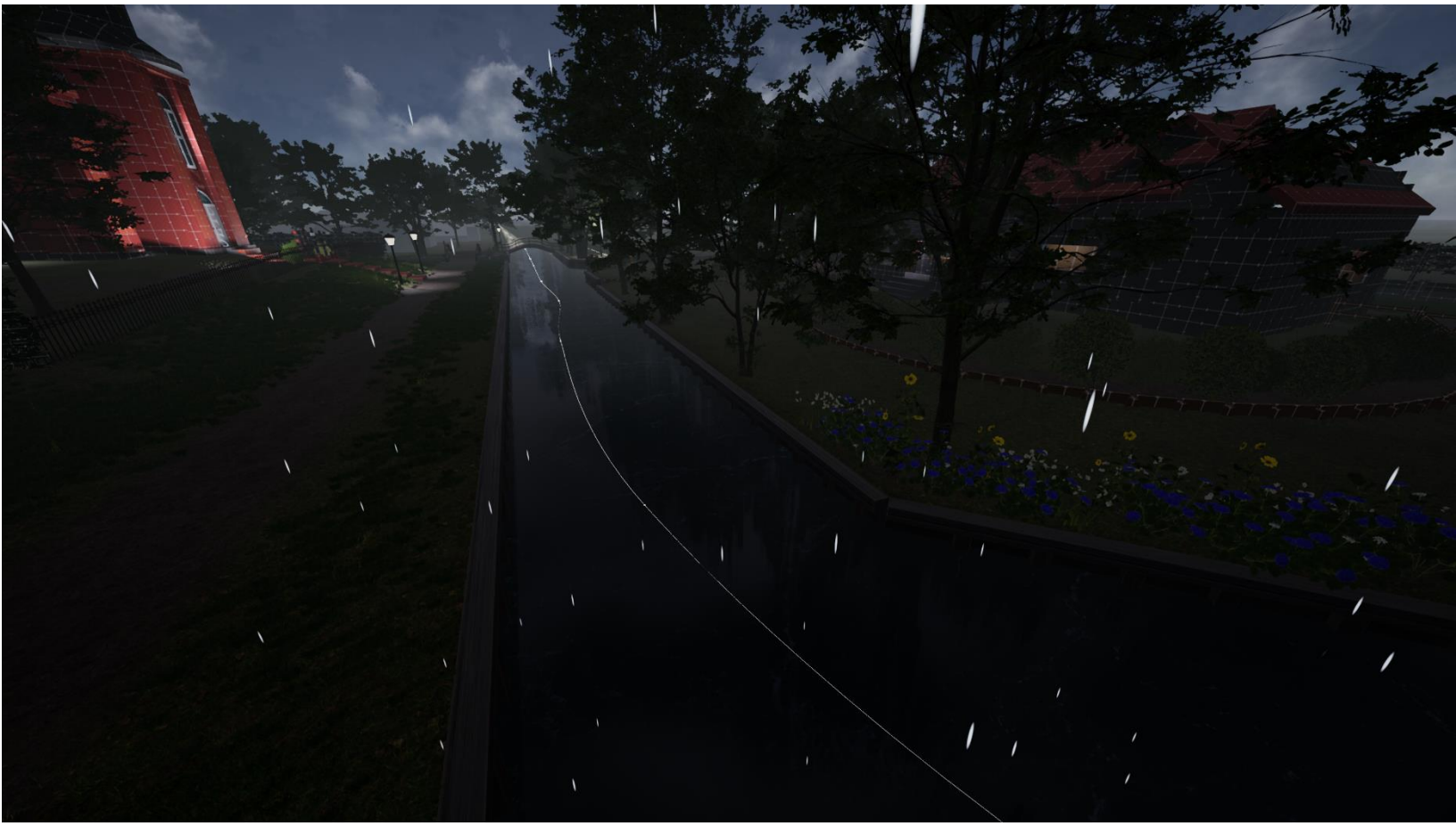
(Figure 39. Sunny weather state + canal water)



(Figure 40. Cloudy weather state + canal water)

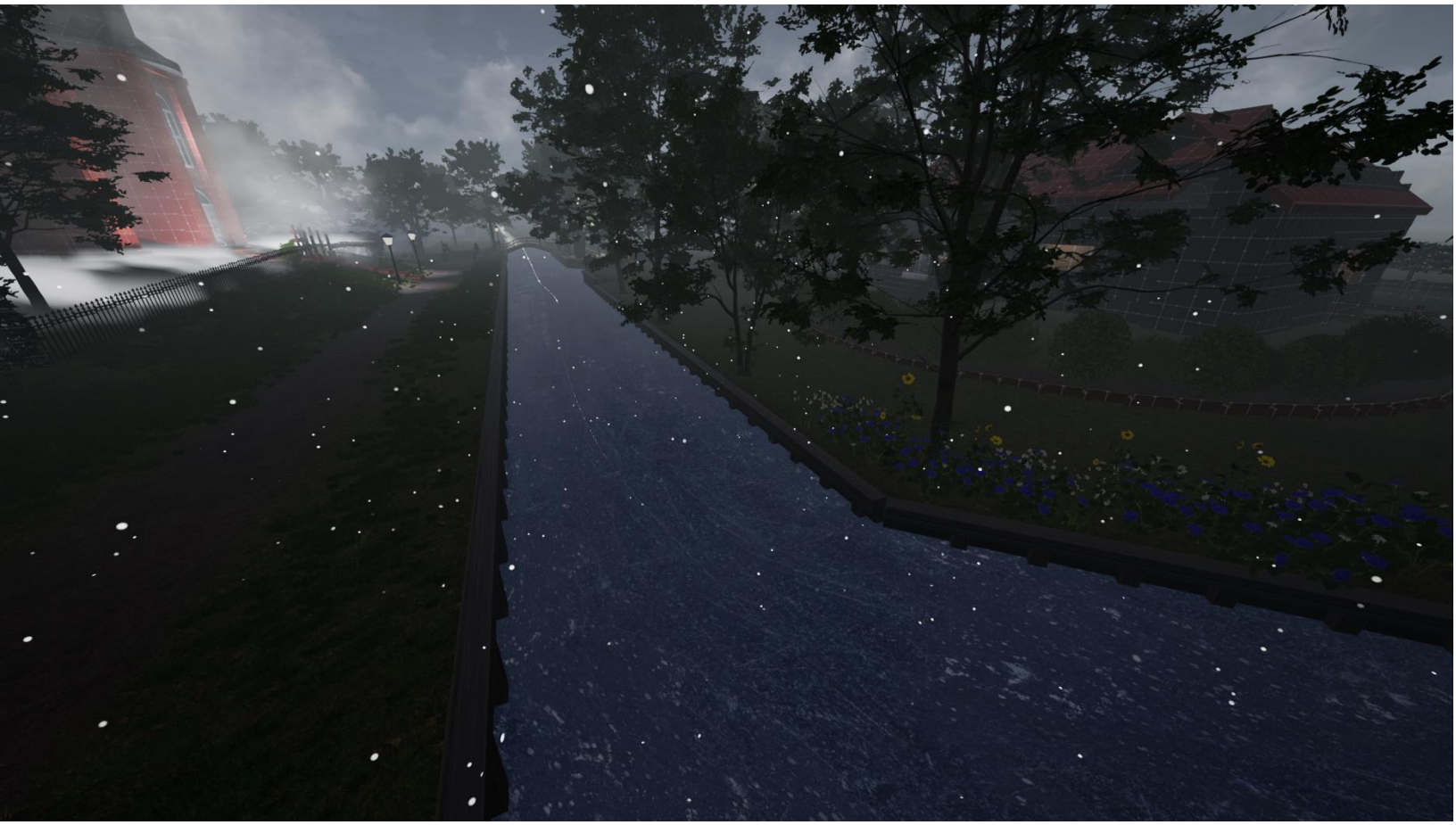


(Figure 41. Stormy weather state + rainfall element + lightning element + canal water)



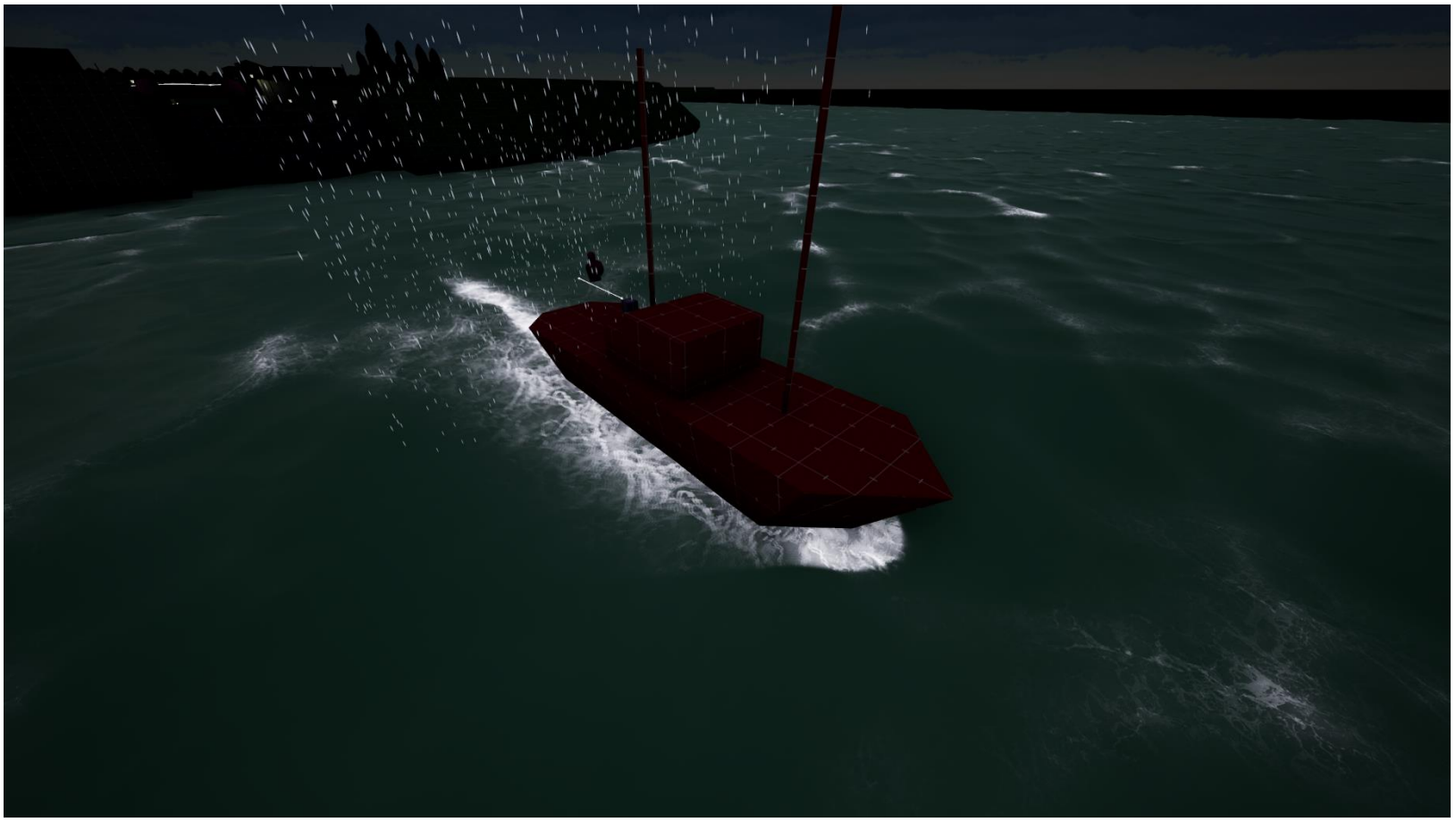
(Figure 42. Rainy weather state + rainfall particles + canal water)

(Figure 43. Rainy weather state + rainfall particles increased intensity.)



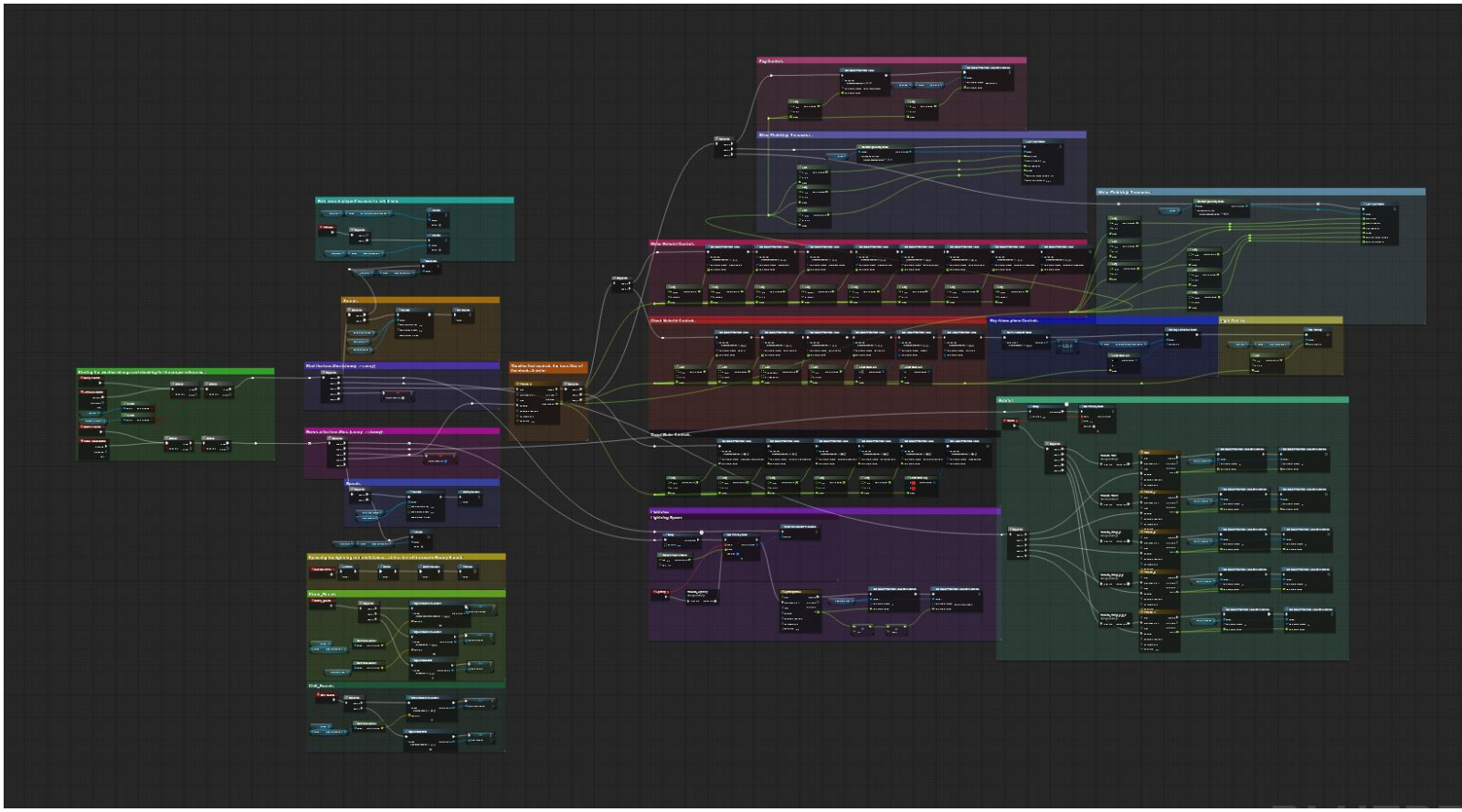
(Figure 44. Snowy weather state + snowfall particles + canal water.)

(Figure 44. Snowy weather state + snowfall particles increased intensity.)

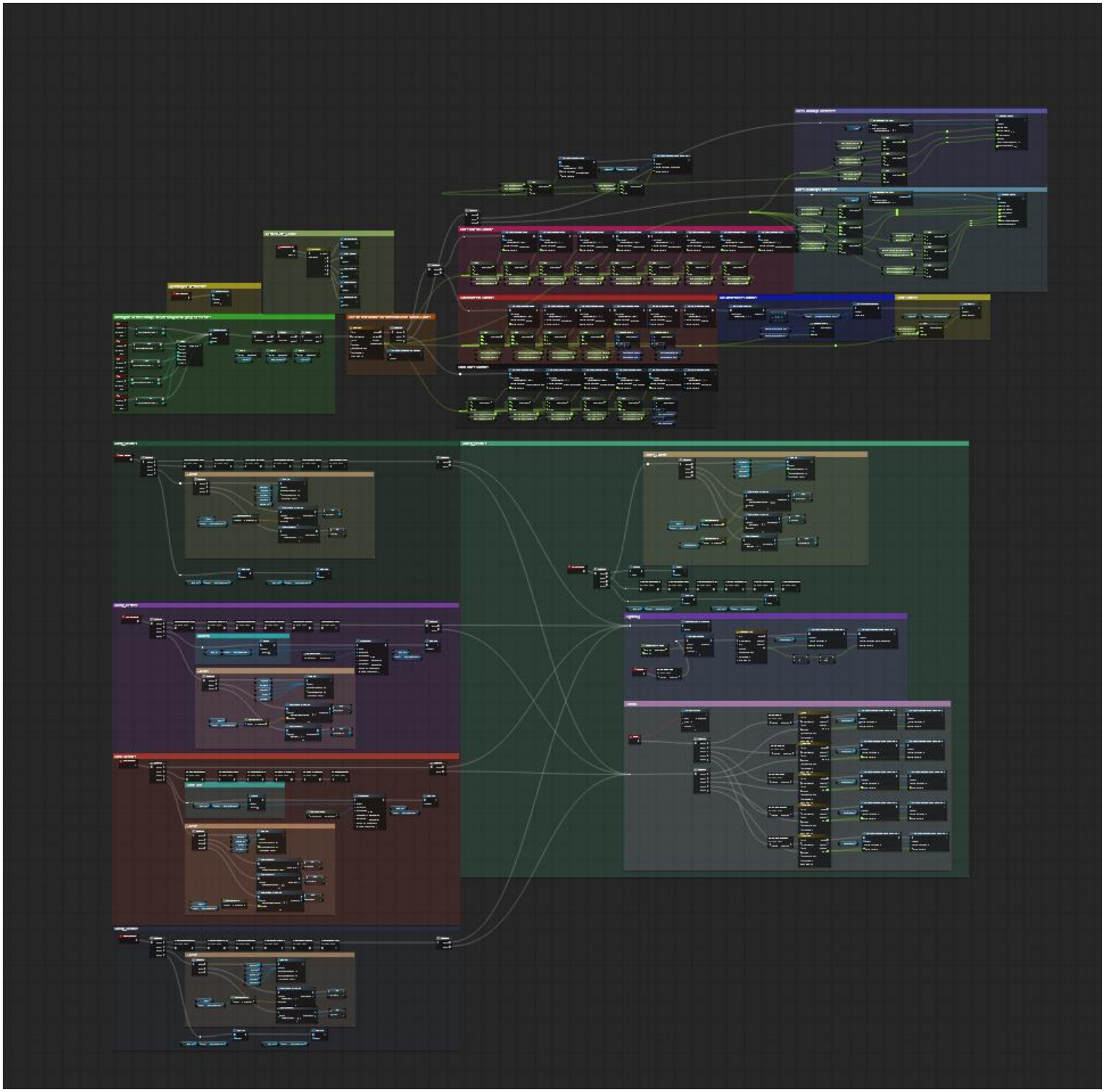


(Figure 45. Boat interaction with the water.)

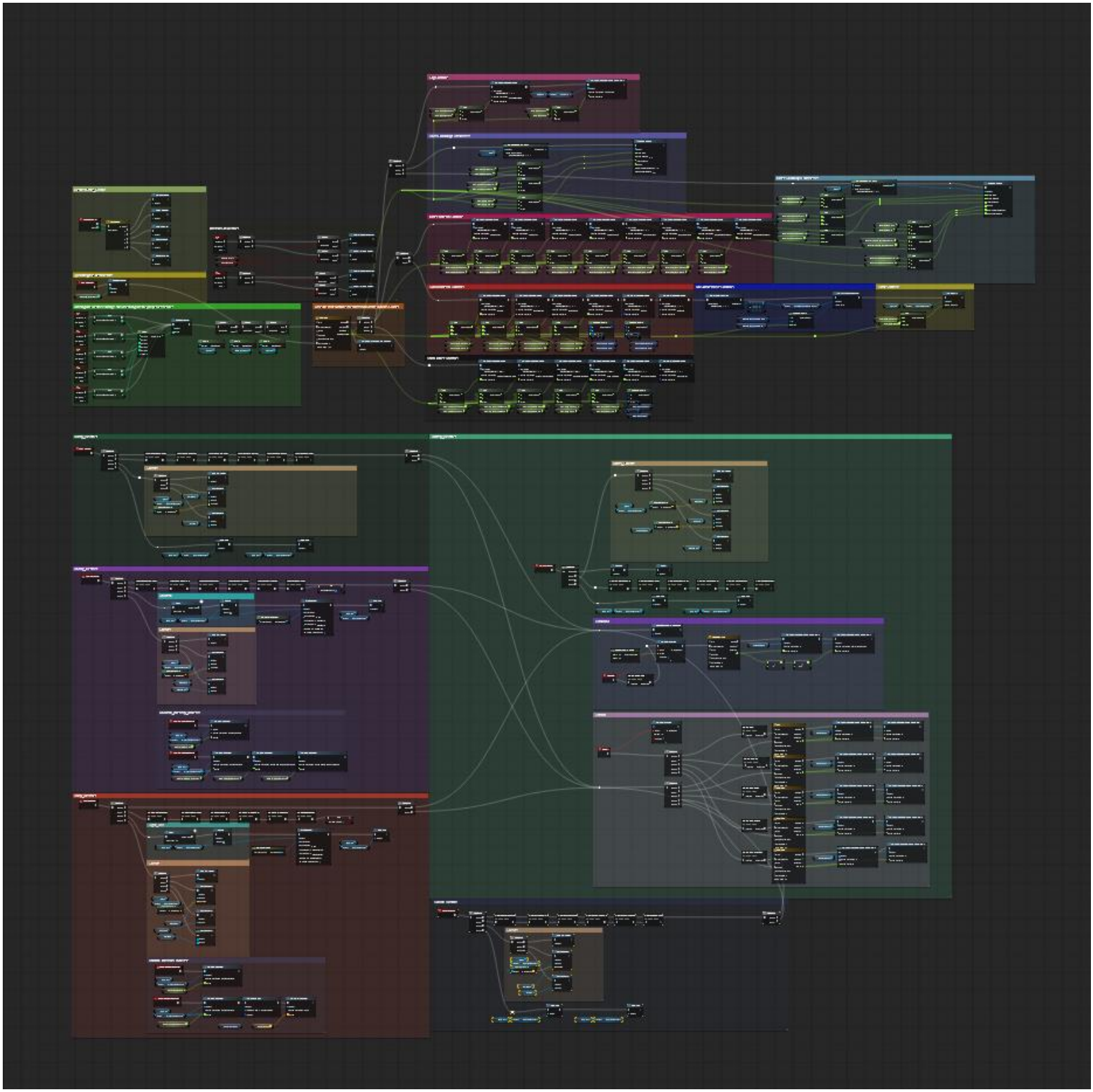
9.4.1. Blueprint versions showcase:



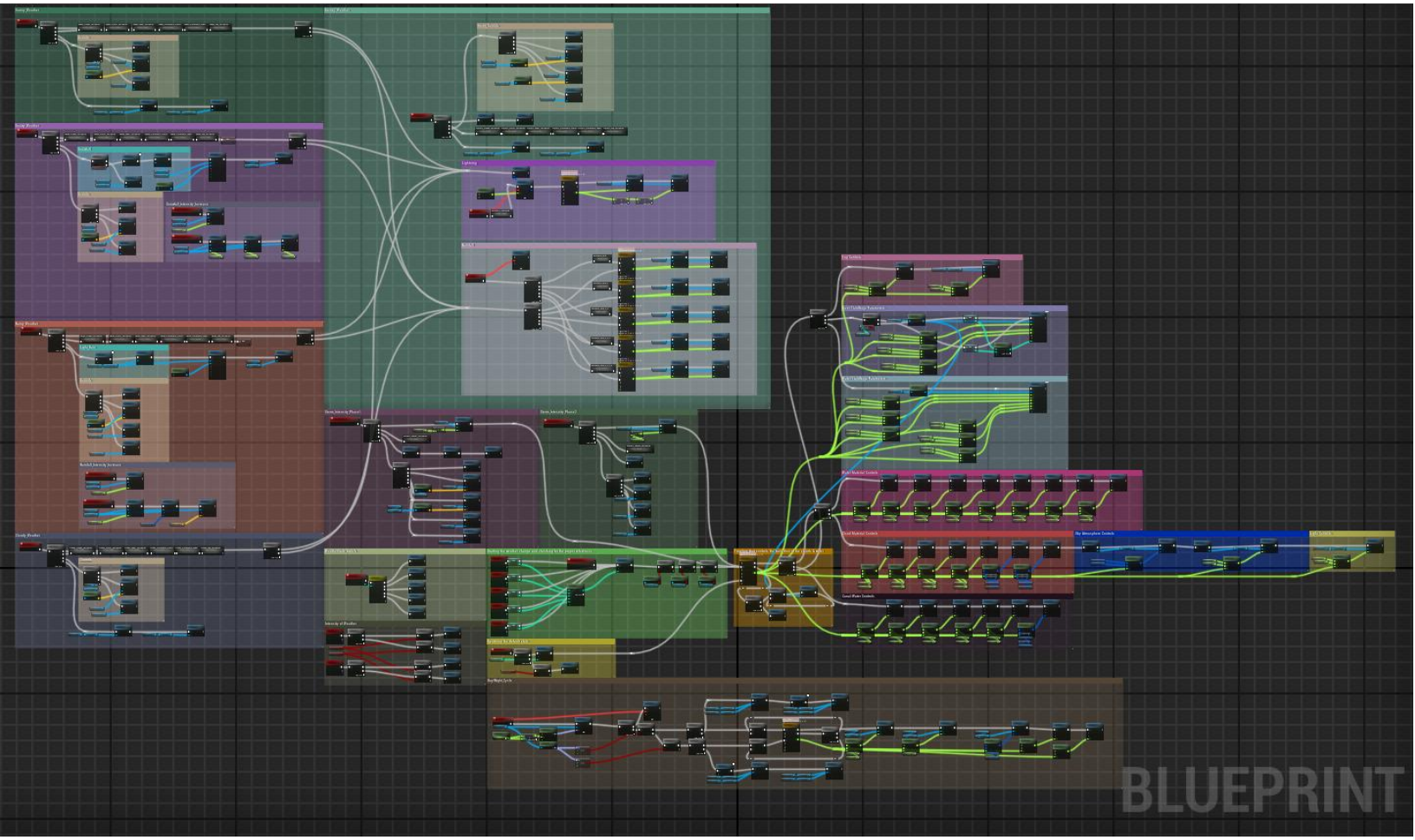
(Figure 46. First version of the dynamic weather blueprint.) Only two weather states and no sounds. Limited control via public variables and no particles.



(Figure 47. Second version of the dynamic weather state blueprint.) 5 weather states but sound not working 100%. Limited particles. Extensive control via public variables.



(Figure 48. Version 2.5 of the dynamic weather system.) 100% working sounds and ability to increase the intensity of the particle effects.



(Figure 49. Final version of the dynamic weather state.) Day and Night cycle added. Ability to change location of the secondary elements.

If you want to see these Blueprints in detail use the links to blueprintue.com where you can zoom in and see each block of code in detail:

Version 1: https://blueprintue.com/blueprint/_dpm-bsy/

Version 2: https://blueprintue.com/blueprint/38dtzbfo/

Version 2.5: https://blueprintue.com/blueprint/4o808ph9/

Version 3: https://blueprintue.com/blueprint/448d11ci/