МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КІБЕРБЕЗПЕКИ, КОМП'ЮТЕРНОЇ
ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ
КАФЕДРА КОМП'ЮТЕРНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри
_____ Аліна САВЧЕНКО
«___»_____2022 р.

КВАЛІФІКАЦІЙНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ МАГІСТР
ЗА ОСВІТНЬО-ПРОФЕСІЙНОЮ ПРОГРАМОЮ
«ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ПРОЕКТУВАННЯ»

Тема: «Симуляція циклу дня і ночі з використанням засобів post-processing на базі Unreal Engine 5»

Виконавець: Єгор ПАНЧЕНКО

Керівник: к.т.н., доцент Сергій ВОДОП'ЯНОВ

Нормоконтролер: к.т.н., доцент Олена ТОЛСТІКОВА

КИЇВ 2022

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет *кібербезпеки, комп'ютерної та програмної інженерії*
Кафедра *Комп'ютерних інформаційних технологій*
Спеціальність *122 «Комп'ютерні науки»*
Освітньо-професійна програма *«Інформаційні технології проектування»*

ЗАТВЕРДЖУЮ:
завідувач кафедри КІТ
_____Аліна САВЧЕНКО
(підпис)
«____»_____2022 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи
Панченко Єгора Олеговича
(ПІБ випускника)

1. Тема роботи: «Симуляція циклу дня і ночі з використанням засобів post-processing на базі Unreal Engine 5» затверджена наказом ректора № 1774/ст від 28.09.2022р.
2. Термін виконання роботи: з 26 вересня 2022 року по 27 листопада 2022 року.
3. Вихідні дані до роботи: Програма симуляції циклу дня та ночі на базі рушія Unreal Engine 5.
4. Зміст пояснювальної записки: 1. Огляд та аналіз рушія Unreal Engine. 2. Дослідження систем візуалізації освітлення 3. Реалізація проекту симуляції.
5. Перелік обов'язкового ілюстративного матеріалу: 1.Актуальність, 2.Проект симуляції циклу зміни дня та ночі, 3.Unreal Engine, 4.Дослідження візуалізації освітлення, 5.Демонстрація готового проекту.

6. Календарний план-графік

№ з/п	Завдання	Термін виконання	Підпис керівника
1.	Аналіз рушіїв та технологій освітлення та пост-процесінгу. Написання 1 розділу, представлення керівнику	26.09.2022-16.10.2022	
2.	Аналіз рушія Unreal Engine 5 та його технологій освітлення. Написання 2 розділу, представлення керівнику	17.10.2022-30.10.2022	
3.	Реалізація проекту симуляції циклу зміни дня та ночі. Написання 3 розділу, представлення керівнику	31.10.2022-14.11.2022	
4.	Загальне редагування та друк пояснювальної записки	15.11.2022-20.11.2022	
5.	Проходження нормоконтролю, перепліт пояснювальної записки.	16.11.2022-20.10.2022	
6.	Розробка тексту доповіді. Оформлення графічного матеріалу для презентації	20.11.2022-22.11.2022	

7. Дата видачі завдання 26.09.2022р.

Керівник кваліфікаційної роботи _____ Сергій ВОДОП'ЯНОВ
(підпис керівника)

Завдання прийняв до виконання _____ Єгор ПАНЧЕНКО
(підпис випускника)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи на тему: «Симуляція циклу дня і ночі з використанням засобів post-processing на базі Unreal Engine 5» містить: 86 сторінок, 104 рисунки, 24 інформаційних джерела, 1 додаток.

Об’єкт дослідження – Тривимірні проекти, створені на Unreal Engine.

Предмет дослідження – Системи симуляції циклу зміни дня та ночі, освітлення та пост процесінгу на базі рушія Unreal Engine 5.

Мета кваліфікаційної роботи – отримати готову програму – симуляцію циклу дня і ночі із застосуванням Unreal Engine 5.

Методи дослідження – рушій Unreal Engine 5, візуальна мова програмування Blueprint.

Результати кваліфікаційної роботи рекомендується використовувати як готовий плагін симуляції зміни циклу дня та ночі із налаштованим освітленням та системою погоди, який можна інтегрувати у будь-який проект на базі Unreal Engine 5.

Для розробки проекту симуляції циклу дня та ночі використано мову програмування Blueprint, документацію Unreal Engine.

СИМУЛЯЦІЯ, ТРИВИМІРНИЙ ПРОЕКТ, BLUEPRINT, UNREAL ENGINE, NIAGARA, NANITE.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

Blueprint	-	Візуальна мова програмування Unreal Engine
SDK *(Software development kit)*	-	Набір для розробки програмного забезпечення
IDE *(Integrated development environment)*	-	Інтегроване середовище розробки
API (Application programming interface)	-	Програмний інтерфейс
Participle System	-	Система часток для створення VFX
VFX *(Visual Effects)*	-	Візуальні ефекти
SFX *(Special Effects)*	-	Спеціальні або звукові ефекти
Emissive	-	Сяючий матеріал
URL *(Uniform Resource Locator)*	-	Визначник місцезнаходження сайту в мережі Інтернет

ВСТУП

Ігрові та загальні тривимірні додатки із акцентом на реалізм мають багато унікальних особливостей, які суттєво відрізняють їх від інших. Для досягнення реалістичної картинки розробники створюють моделі та текстури високого розширення, системи генерації рослин, реалістичні моделі поведінки рідин, процедурні лицеві анімації персонажів, та багато інших важливих елементів, потребу в спеціалістах щодо яких можна часто знайти на різноманітних сайтах із вакансіями.

Одним із найважливішим елементом серед них є освітлення. Гармонічно та вміло підібране освітлення може не тільки відобразити потрібну атмосферу у сцені, або задати настрій, але й приховати графічні недоліки завдяки вмілій стилізації. Однак, освітлення, як і інші елементи тривимірних проектів, являється надзвичайно складним на багатогранним конструктом, що потребує знань не тільки в суто комп'ютерних системах обробки світла, але й стандартних елементів постпроцесингу, який напряму корелює із основами фотографії.

У цій кваліфікаційній роботі була поставлена мета дослідити якомога більшу кількість можливих елементів корегування на покращення світлових, атмосферних та інших пов'язаних елементів візуалізації, та застосувати їх на практиці у створенні автономної системи циклу зміни дня та ночі на базі Unreal Engine 5.

Автономність системи була досягнута інкапсуляцією усіх необхідних класів, їх функцій, об'єктів освітлення та оточення у одному класі Blueprint, що створило можливість експортувати цей клас у будь-який потрібний проект, що одразу створює у цільовому проекті готову систему освітлення, із мінімумом потрібних налаштувань.

Окрім створення системи освітлення та симуляції циклу зміни дня та ночі, також була створена базова система погоди, а саме симуляція дощу, із використанням системи часток Niagara System для реалізації дощу, та системи Volumetric Clouds для створення та анімації хмар.

Система погоди була імплементована із міркувань збільшення спектра сценаріїв освітлення, що дозволило більш наглядно продемонструвати реалізацію освітлення у проекті.

Для візуаліції керування процесами швидкості зміни дня та ночі, та переключення погоди між ясною та дощем, було розроблено користувацький інтерфейс, який був створений вручну у додатку Photoshop. Такий підхід дозволив розробити унікально виглядяючий та візуально приємний інтерфейс, який так само інкапсюльований усередині класу, роблячи його універсальним для використання у інших проектах.

Unreal Engine 5 не тільки є одним із найбільш юзер-френдлі рушієм для будь-яких тривимірних проектів, але й має в арсеналі доволі прогресивні технології, такі як Lumen та Nanite, що створюють нові можливості та спрощують робочі процеси пов’язані із освітленням. Також багато допоміжних систем, такі як Lighting Channels, є надзвичайно корисними для створення унікально виглядаючих моделей освітлення та сцен.

Дуже важливим для майбутніх перспектив також є доступ до технологій Quixel, які забезпечують безкоштовний доступ до фотореалістичних моделей, текстур та утиліт, які працюють надзвичайно добре саме із Unreal Engine, що робить створення фотореалістичних проектів надзвичайно простою задачею, яка не потребує зовнішнього втручання та витрат на моделі, текстури та матеріали для створення сцен.

Навички у даній галузі не тільки є доволі затребувані, але й мають ідеальний вигляд для портфоліо, оскільки покривають не тільки галузь освітлення саме по собі, але й безліч інших елементів сцени, які потребують окремої уваги і навичок. А досвід із рушієм Unreal Engine 5 є однією із основних потреб для спеціалістів у галузі ігрової розробки та кінематографу, що включає у собі безліч професій та спеціалізацій, доступ до яких є набагато легшим за аналоги.

РОЗДІЛ 1
ОГЛЯД ТА АНАЛІЗ РУШІЯ UNREAL ENGINE

1.1. Unreal Engine

Впродовж доволі довгого часу розробка багатофункціонального тривимірного програмного забезпечення було складною задачею через власницькі рушії, які кожна компанія робила виключно для внутрішніх потреб. Через це, для незалежних осіб чи компаній – початківців, створення подібних проектів, в тому числі ігор, було набагато складнішим завданням, адже перш за все була необхідність у створенні свого рушія. На щастя, у середині нульових років почалась популяризація багатоплатформових рушіїв – інструментів для створення ігор, які були орієнтовані на більш широку публіку, яка не мала досвіду роботи із ігровими рушіями до цього. Серед них можна виділити Unity та Unreal Engine як найшвидші у розвитку та якості.[1]

Рис. 1.1 Рушії для розробки ігрових додатків

Однак у останні роки саме Unreal Engine став найкращим вибором не тільки для індивідуальних користувачів, але й для багатьох малих так великих компаній, не тільки через доступність, але й через багато технологій, власноруч створених Epic Games – компанією – розробником Unreal Engine, а також технологіями, придбаними разом із іншими компаніями.[2] Подібна доступність, а також підтримка та попит на людей із досвідом у Unreal, робить цей рушій універсальним вибором у сфері сучасних тривимірних проектів.

Деякі технології, зокрема Nanite, Lumen та технології Quixel є найбільш вражаючими, та потребують окремої уваги.[3]

1.2. Nanite

Nanite — технологія, що прийшла на зміну звичайного меш-рендеринга, є технологією віртуальної геометрії, модифікує процес рендеру моделі.

Nanite не завантажує усю геометрію об’єктів, а лише ту, яку бачить користувач.

Сіточний шейдинг – аналогічна технологія, яку до цього намагалася створити NVidia.

Стандартна система відмальовування полігонів моделей, яка наразі використовується – система LOD`ів. Через те, що їх було зазвичай від двох до чьотирьох інстансів, моделі повинні були відмальовувати по декілька разів в залежності від ситуації

LOD’и — це різнополігональні моделі, одного й того ж самого об’єкта. Вони рендеряться в потрібному розширенні та об’єму полігонів в залежності від відстані камери до них. Такий підхід значно полегшує обчислення для системи, але створює додаткові кроки при створенні тривимірної моделі. Адже зазвичай створюється від 2 до 4 таких ЛОДів. При роботі з Nanite, модель створюється лише один раз, і відразу у найвищій якості. І тоді гравець буде сам налаштовувати якість об’єкта з урахуванням відстані до нього.

Меш розбивається на логічні безшовні ділянки, так звані «кластери», які нагадують форму трикутника, базуючись на швах розгортки. До речі, кластери

генеруються на декількох рівнях деталізації, щоб замінювати один одного, або й зовсім не рендериться в залежності від відстані до камери і кута огляду.

Це відрізняється від простих LOD-ів — геометрія завантажується лише у необхідних для рендеру частинах меша. Під час рендеру вибираються найкращі LOD-кластери трикутників залежно від частини екрану, яку вони займають.

Nanite враховує роздільну здатність екрана та відстань до меша. Для збереження деталізації вона може створювати трикутники розміром із піксель. Використовуючи Nanite, не потрібно думати про кількість полігонів, але зі зміною процесу моделювання з'являються моменти, які потрібно враховувати при роботі з Nanite. Для прикладу, Nanite ще не вміє рендерити прозорі об'єкти — скло. Але Epic Games обіцяють скоро цю проблему вирішити.[4]

Рис. 1.2 Приклад роботи Nanite

1.3. Lumen

Lumen — це повністю динамічна система глобального освітлення та відображень Unreal Engine 5, розроблена для консолей наступного покоління, і система глобального освітлення та відображень за замовчуванням. Lumen створює дифузне взаємовідбиття з нескінченними відскоками та непрямими

дзеркальними відбиттями у великих деталізованих середовищах у масштабах від міліметрів до кілометрів.

Про освітлення в іграх ми вже трохи знаємо. Мабуть, всі чули про технологію Ray-tracing. Це вражаюча система динамічного освітлення, яка вираховує вектор траєкторії руху променів світла так, як би вони рухалися в реальному житті. Хорошим прикладом слугує Cyberpunk 2077. При ввімкнені цієї технології картинка виглядає максимально реалістично. Правильне відображення та світло роблять геймплей максимально аутентичним. Саме завдяки цьому гравець може повністю поринути в ігровий світ.

Але у цієї технології є явна проблема. Навіть декілька. Не всі комплектуючі можуть її підтримувати. Ray-tracing можна увімкнути лише на відеокартах від Nvidia з приставкою RTX, або на аналогах від AMD. А по-друге, це дуже важка задача. Адже для організації глобального освітлення потрібно вираховувати траєкторію променів в реальному часі. І не пропускати ті об’єкти, які знаходяться за кутом огляду камери користувача. Як ви розумієте, це забирає дуже багато ресурсів вашої відеокарти.

Рис. 1.3 Порівняння систем відображення

Lumen же спрощує підхід до обрахунку променів. Хоча технологія по своїй суті дуже схожа на вищеописану. Але Lumen дозволяє обходити великі витрати ресурсів, проводячи рендер по спрощеній системі.

Також козирем Lumen є м’які, розмиті тіні. Та відображення. Що несамовито додають реалістичності всій картині. На старих версіях Unreal Engine відображення та освітлення були різними функціями. Тут вони працюють як єдина система, і це дозволяє створювати більш реалістичне зображення, адже обмін даними відбувається напряму.[5]

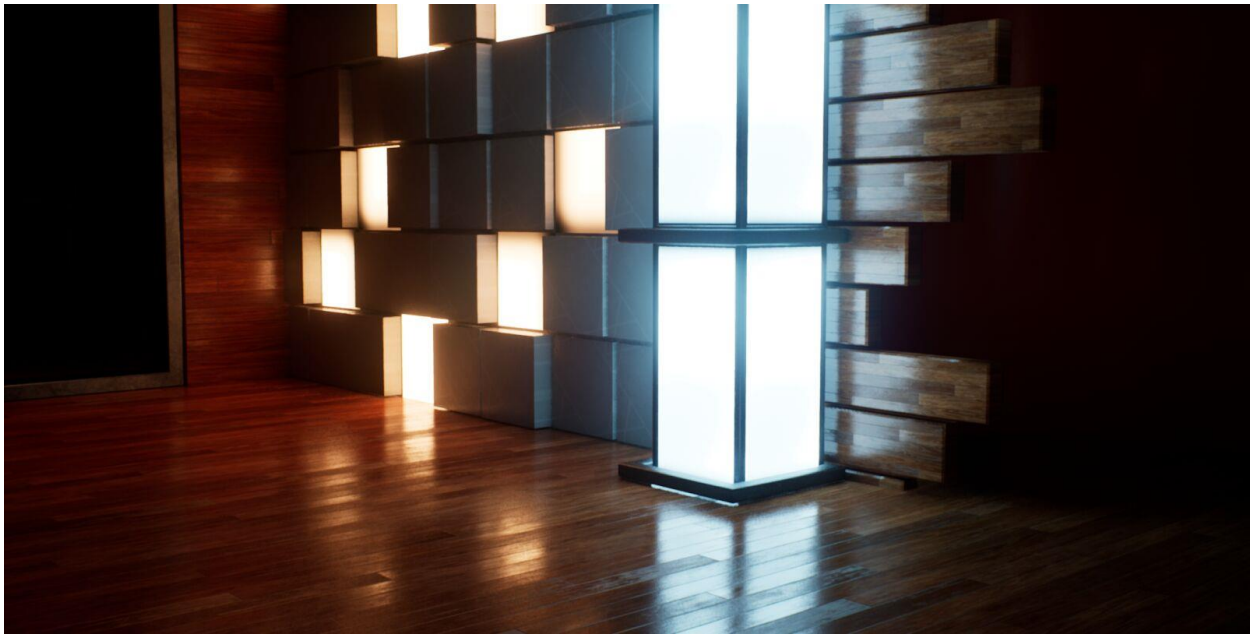


Рис. 1.4 Приклад роботи Lumen

1.4. Технології Quixel

Дві технології, які мають доволі вагомі переваги щодо вибору Unreal Engine, це Quixel Megascans+Quixel Bridge, та Quixel Mixer.

Quixel Megascans[6] дає можливість використовувати доволі широкий вибір фотореалістичних тривимірних моделей, текстур та матеріалів для створення реалістичного середовища, а плагін Quixel Bridge[7] для Unreal Engine дозволяє імпортувати усі ці ресурси до рушії усього за пару кліків. Так як увесь контент Quixel безкоштовний, такий функціонал дає можливість будь

кому створювати захоплюючі сцени та пейзажі без великих витрат на художників.

Рис. 1.5 Приклад 3д сканів Quixel Megascans

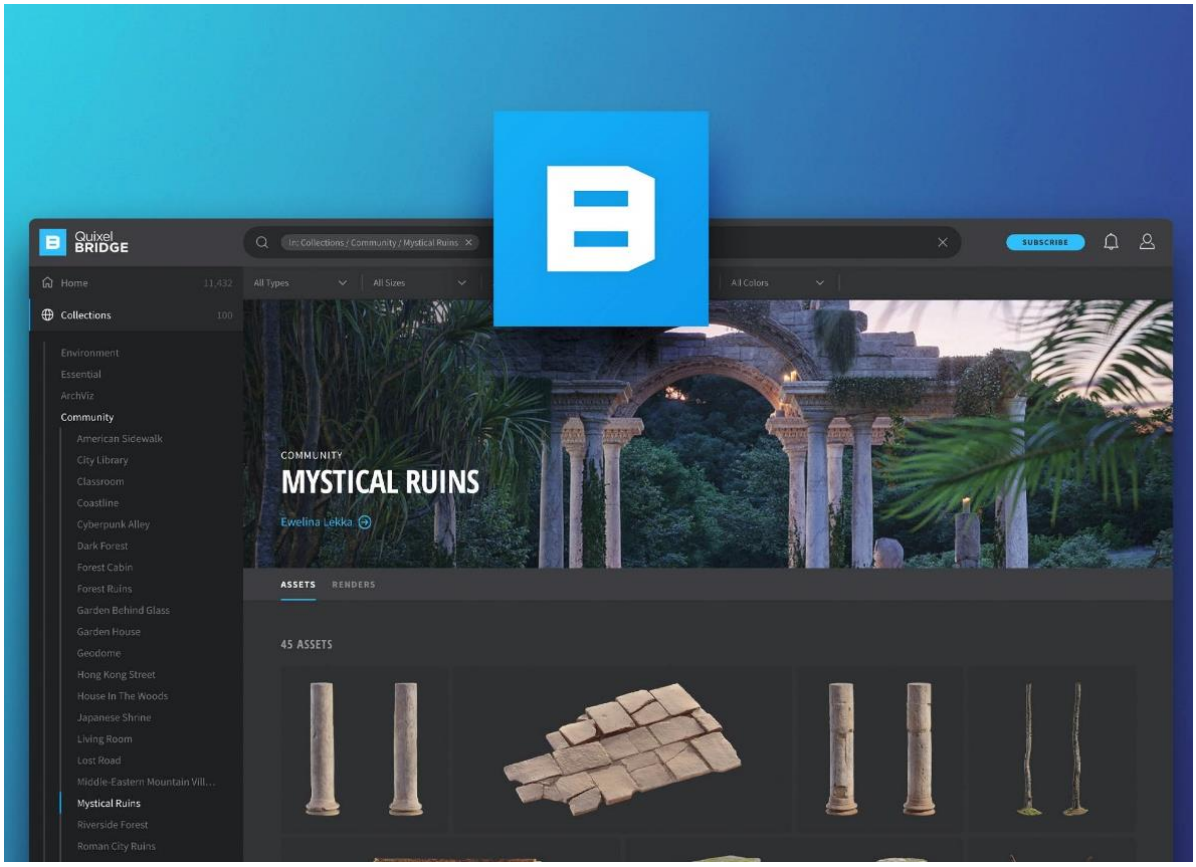


Рис. 1.6 Середовище плагіну Quixel Bridge для Unreal Engine 5

Щодо Quixel Mixer[8], ця утиліта є прямим безкоштовним конкурентом Adobe Substance Painter, що дозволяє створювати і корегувати текстури на тривимірних об’єктах, практично виконуючи роль «фотошопу для 3д». Так як цінник у Substance Painter доволі жорсткий, Mixer є дуже якісною і необхідною утилітою для таких задач.

Рис. 1.7 Середовщие додатку Quixel Mixer

ВИСНОВКИ ДО РОЗДІЛУ 1

В даному розділі було розглянуто рушій Unreal Engine, його переваги перед іншими рушіями, та перспективність.

Було детально розглянуто особливості та елементи п’ятої ітерації Unreal Engine, нові революційні технології, які дозволяють зменшити необхідні зусилля для створення проектів, а також покращити їх швидкодію без втручання користувача.

Також були розглянуті колаборації із іншими компаніями, які дозволили використовувати професійні програмні рішення без додаткових фінансувань.

РОЗДІЛ 2
ДОСЛІДЖЕННЯ СИСТЕМ ВІЗУАЛІЗАЦІЇ ОСВІТЛЕННЯ

2.1. Альбедо

Перш за все, для аутентичного вигляду необхідно, щоб усі текстури на об’єктах мали гармонічні рівні освітлення. Це досягається налаштуванням рівнів альбедо у матеріалах об’єктів. Для референсу використовується неосвітленний (“Unlit”) режим, де ми чітко бачимо, у яких об’єктів рівень освітленності відрізняється від середнього[9].

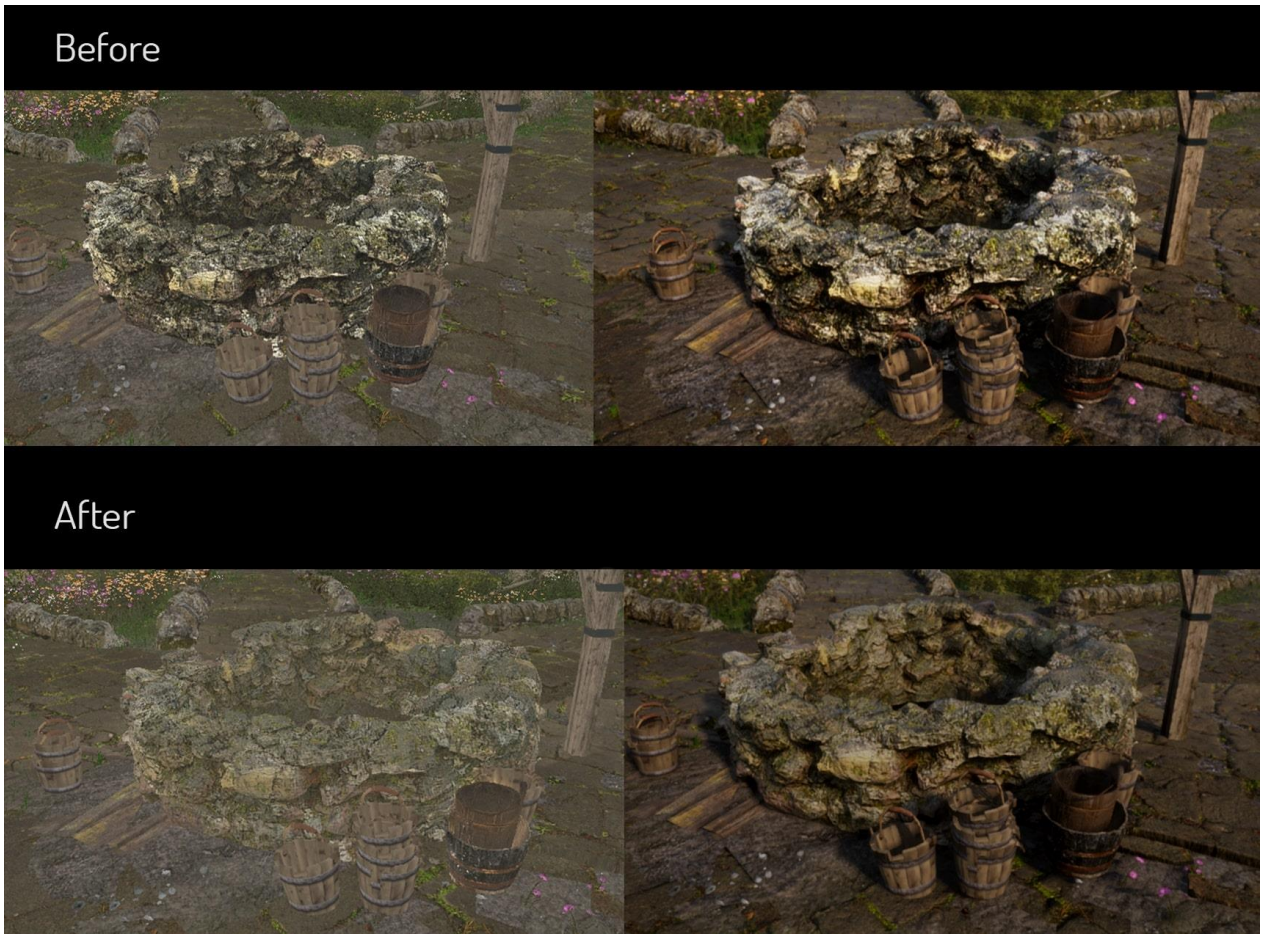


Рис. 2.1 Приклад застосування Альбедо

2.2. HDRI

High Dynamic Range Imaging, HDRI або просто HDR — загальна назва технологій роботи із зображеннями і відео, діапазон яскравості яких перевищує можливості стандартних технологій відображення зображень[10].

Найчастіше термін HDR вживається щодо отримання, зберігання і обробки растрових зображень. Широко використовуються на сьогоднішній день цифрові технології історично засновані на 8-бітових цілочисельних форматах представлення та обробки даних, що дає дуже вузький динамічний діапазон, який часто називають SDR (англ. Standard Dynamic Range) або LDR (англ. Low Dynamic Range). Для порівняння, поширені стандарти JPEG і MPEG дозволяють відобразити динамічний діапазон порядку 1 000:1, в той час як реальні сцени часто мають динамічний діапазон яскравості в 1 000 000:1 і вище. Застосування техніки HDR дозволяє працювати з повним діапазоном яскравості сцени, усуваючи історичні обмеження.

Технології HDR мають безліч практичних застосувань, такі як отримання зображень і відео натуральних висококонтрастних сцен, зберігання та обробку HDR контенту, створення LDR зображень на основі HDR зображень, а також досягнення різних художніх ефектів використовуючи HDR зображення.

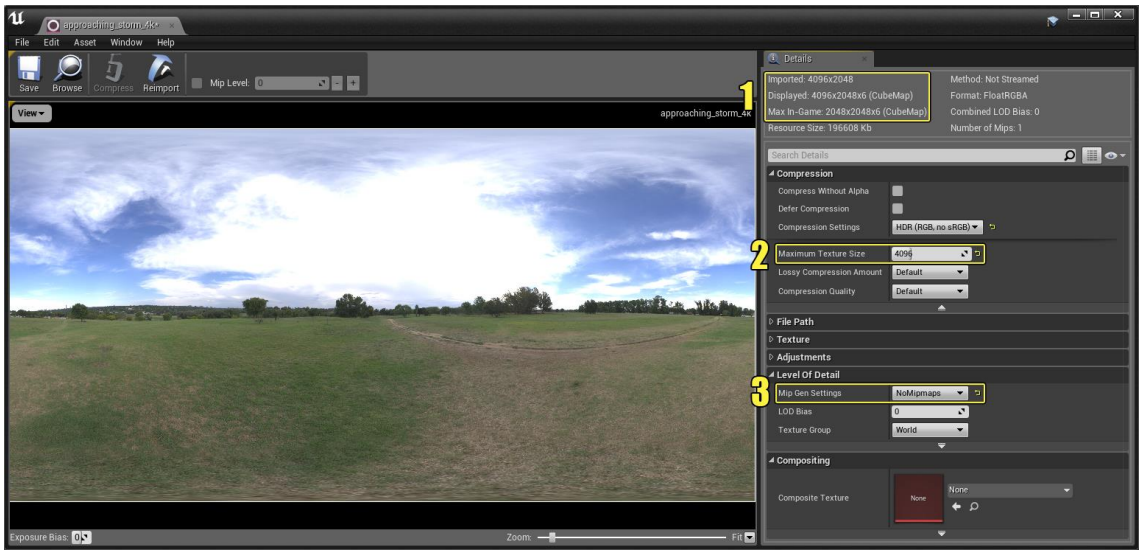


Рис. 2.2 HDRI меню у Unreal Engine

У нашому випадку, HDRI використовується як скайбокс із загальним освітленням. Для аутентичності освітлення використовують HDRI, які вже були б досить темними або яскравими. Звичайно, можна і трохи налаштувати яскравість і контраст тут і там, налаштувавши значення плану, але результат був би менш задовільним, ніж при налаштуванні самої текстури.

2.3. Directional light

Directional light (спрямоване світло) імітує світло, що випромінюється від нескінченно далекого джерела. Це означає, що всі тіні, які відкидає це світло, будуть паралельними, що робить його ідеальним вибором для імітації сонячного світла.[11]

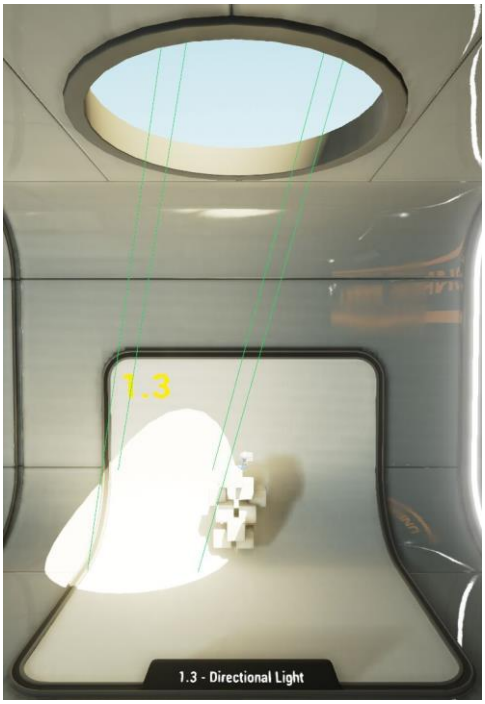


Рис. 2.3 Приклад роботи спрямованого світла

Для спрямованого світла можна встановити один із трьох параметрів мобільності:

- **Статичний** - означає, що світло не можна змінити в грі. Це найшвидший метод візуалізації, який дозволяє використовувати яскраве освітлення.

- **Стаціонарний** –означає, що світло матиме лише затінення та відбите освітлення від статичної геометрії, створеної Lightmass, усе інше освітлення буде динамічним. Це налаштування також дозволяє змінювати колір та інтенсивність світла під час гри, але воно не рухається й дозволяє частково запекти освітлення.
- **Пересувний** - означає, що світло повністю динамічне та дозволяє динамічно затіняти. Це найповільніше з точки зору візуалізації, але забезпечує найбільшу гнучкість під час гри.

Спрямоване світло має широкий вибір налаштувань, але найголовніші із них це інтенсивність, колір та температура. Колір використовується зазвичай для неординарних ситуацій, тоді як температура найбільш підходить для змінення дня та ночі.

Рис. 2.4 Налаштування спрямованого світла

2.4. Тіні

Тіні дають відчуття приземленності об’єктам у світі, та дають глядачеві відчуття глибини та простору. Тоді як статичні тіні майже непомітні в плані

навантаження для рендеру, динамічні тіні значно впливають на швидкодію рушію. Для прикладу впливу тіней на сприйняття освітлення та оточення у грі, таким чином змінюється вигляд трави на землі із зміною параметрів DistanceField Shadow Distance та збільшення параметру Contact Shadow Length(Рис 2.5). Такий підхід до візуалізації тіней дозволяє значно збільшити аутентичність та реалістичність оточення без значних навантажень на систему, які спричиняє збільшення кількості динамічних асетів.[12]

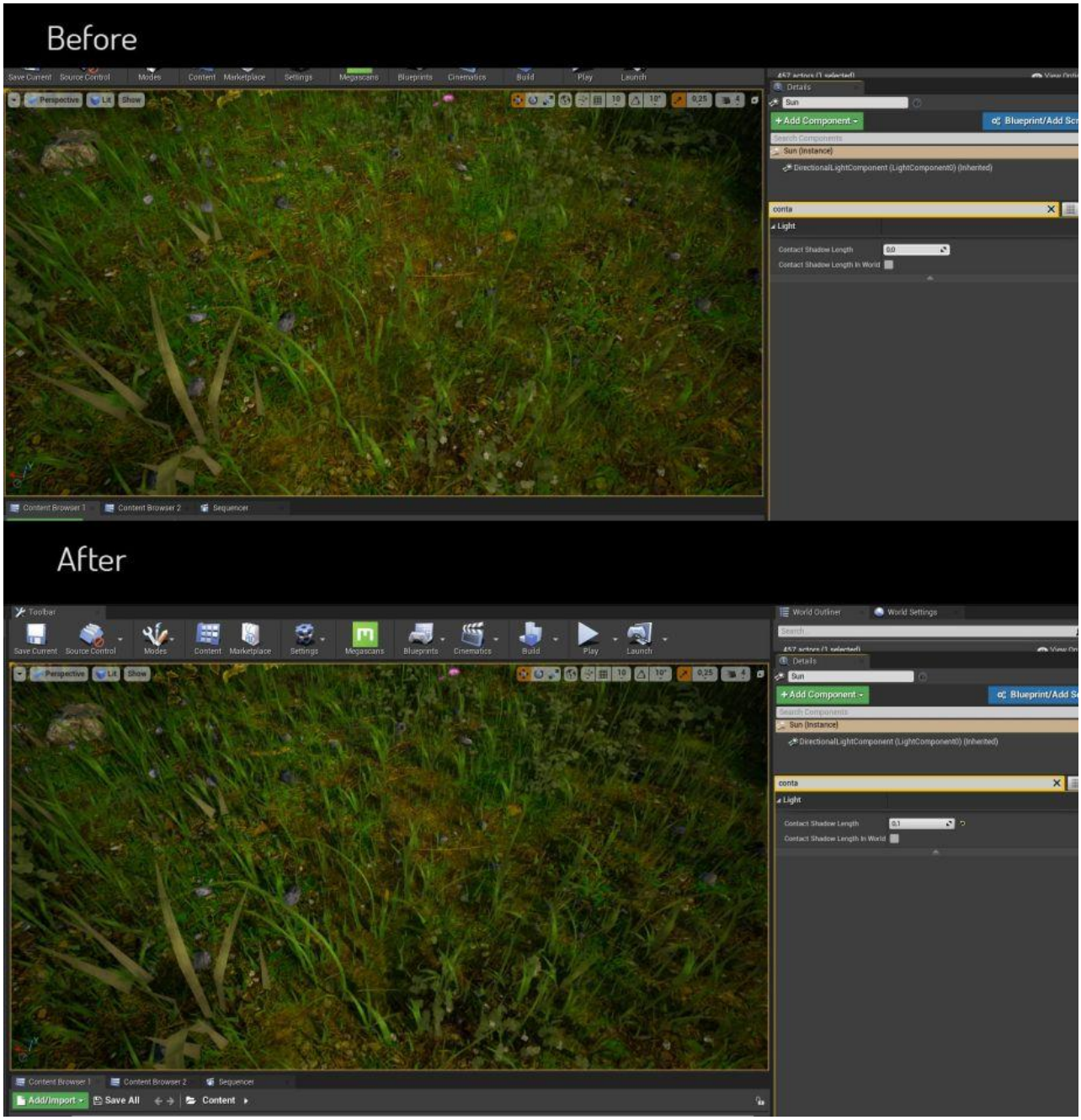


Рис. 2.5 Налаштування тіней

2.5. Туман

Для забезпечення реалізму та атмосферності навколишнього середовища використовується ефект туману, який не тільки робить сприйняття сцени більш реалістичною, але й забезпечує оптимізацію навантаження на комп’ютер, через зменшення розширення об’єктів, які існують поза туманом. Найчастіше використовується Exponential Height Fog – Туман експоненціальної висоти. Він створює більшу щільність у низьких місцях карти та меншу щільність у високих місцях. Перехід плавний, тому при збільшенні висоти ефект туману збільшується гармонічно. Туман експоненціальної висоти також забезпечує два кольори туману — один для півкулі, спрямованої до домінуючого спрямованого світла (або прямо вгору, якщо такого немає), і інший колір для протилежної півкулі.[13]

Рис. 2.6 Приклад роботи туману експоненціальної висоти

2.6. Канали освітлення

Канали освітлення дозволяють динамічному світлу впливати на об’єкти лише тоді, коли їхні канали освітлення перекриваються. Це насамперед призначено для кінематографічного використання, щоб надати користувачам

потужний контроль над освітленням акторів. На даний момент Unreal Engine підтримує до 3 каналів освітлення. За допомогою цього інструмента, об’єкти у одній сцені можуть бути освітленні різними джерелами світла, що дозволяє акцентувати увагу гравця на тому елементі, який хоче показати розробник.[14]

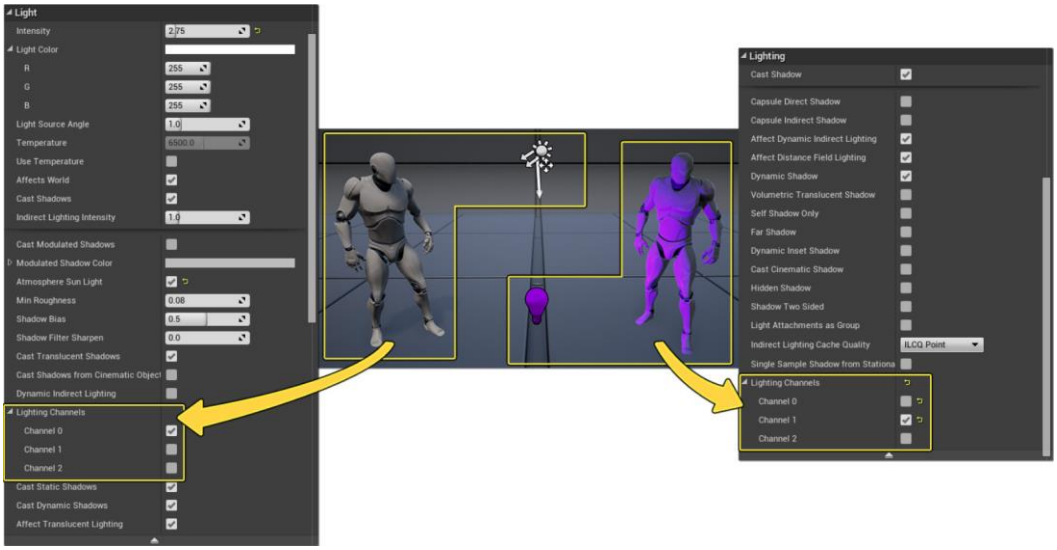


Рис. 2.7 Приклад роботи каналів освітлення

2.7. Lightmass Portal

Збираючи освітлення, Lightmass може повертатися до точкових, місцевих і спрямованих джерел світла за допомогою фотонів із техніки Photon Mapping. Це означає, що він може знаходити невеликі вікна, куди потрапляє світло від цих типів освітлення, і якісно розрізняти вхідне освітлення.

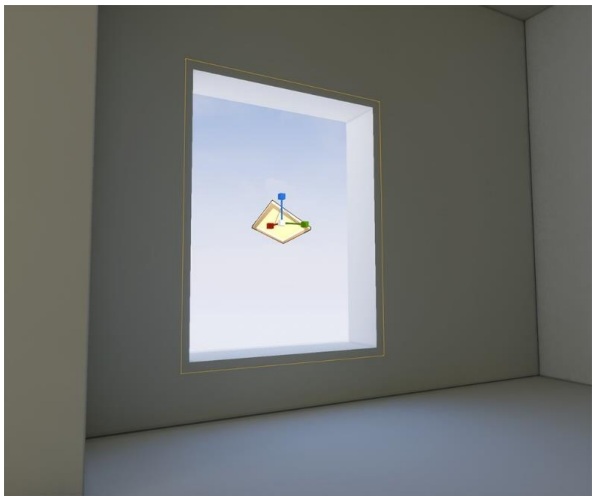


Рис. 2.8 Приклад розміщення Lightmass portal

Однак сітки SkyLights і Emissive не підтримують випромінювання фотонів ефективно, тому Lightmass може знайти лише невеликі важливі функції освітлення за допомогою грубої сили. Це проявляється у вигляді плямистих артефактів навколо внутрішніх кутів.

Щоб допомогти Lightmass краще зрозуміти, звідки має надходити світло, розміщують Lightmass портали навколо областей, які мають вирішальне значення для освітлення.[15]

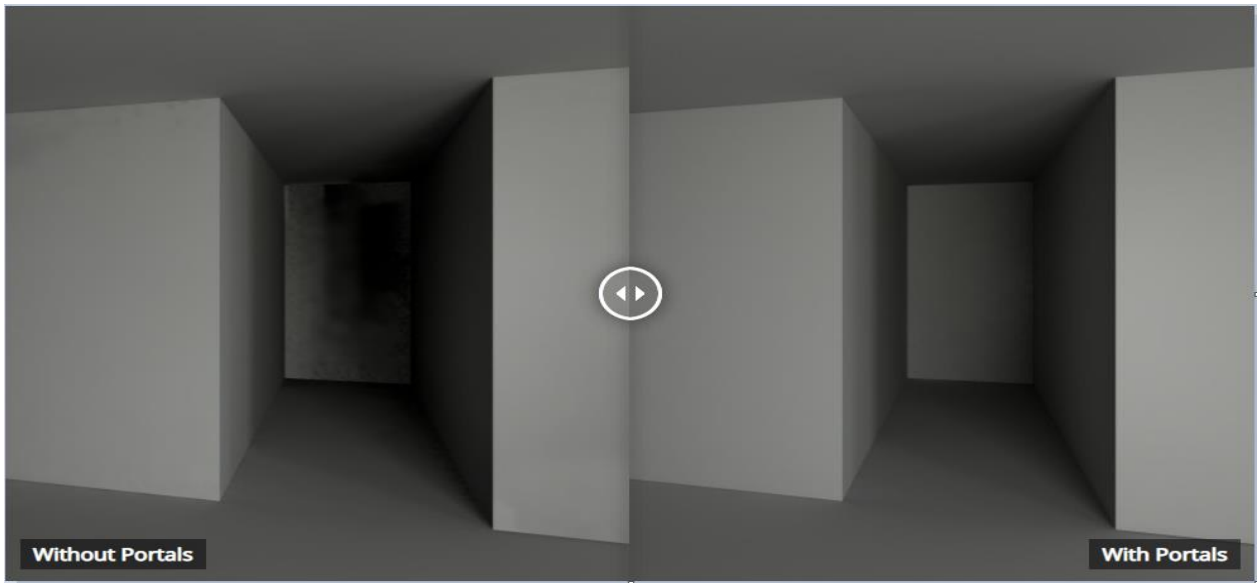


Рис. 2.9 Приклад роботи Lightmass portals

2.8. Volumetric Clouds

Компонент Volumetric Cloud — це фізична хмарна система візуалізації, яка використовує підхід, орієнтований на матеріалах, щоб надати художникам і дизайнерам свободу створювати будь-які типи хмар, які їм потрібні для своїх проектів. Хмарна система обробляє динамічні налаштування часу доби, які доповнюються Sky Atmosphere та Sky Light за допомогою режиму захоплення в реальному часі. Система надає масштабовані, визначені художником хмари, які можуть адаптуватися до проектів, використовуючи види землі, польоти та переходи з землі у відкритий космос.[16]

Volumetric Clouds дозволяють імплементувати атмосферні елементи, які були б неможливі при звичайних умовах. Наприклад, через те, що у цих хмар

присутній матеріал, вони можуть відбивати тіні на навколишнє середовище(Рис 2.9).

Рис. 2.10 Тіні при застосуванні Volumetric Clouds

ВИСНОВКИ ДО РОЗДІЛУ 2

В даному розділі було розглянуто методи освітлення та амбієнсу, які імплементовані до рушія Unreal Engine 5.

Було розглянуто вплив Альбедо на аутентичність та контрастність освітлення сцени, використання HDRI для створення реалістичного та швидкодійного заднього плану для сцени, застосування Directional Light для симуляції сонячного та місячного сяйва, та його вплив на сцену, значна роль тіней на сприйняття сцени, та їх вплив на швидкодію, туман як спосіб збільшити швидкодію проекту, та створити реалістичний амбієнс, кастомізація світла через канали освітлення.

Також були розглянуті Lightmass Portals, які дозволяють вручну налаштовувати об’єкти на сцені, які виглядають кепсько при налаштуваннях за замовчуванням, та сучасну систему хмар – Volumetric Clouds, які дають змогу налаштовувати реалістичну погоду завдяки використанню матеріалу із Voronoi Texture.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ПРОЕКТУ СИМУЛЯЦІЇ

3.1. Створення циклю дня та ночі

3.1.1. Створення проекту

Спочатку потрібно створити проект у середовищі Unreal Engine 5, який буде відповідати вимогам програми. Для заданої програми підходять налаштування ігри із перспективою від першої особи – це дозволить споглядати зміни у проекті безпосередньо із очей «гравця», що буде краще відображати навколишні зміни.

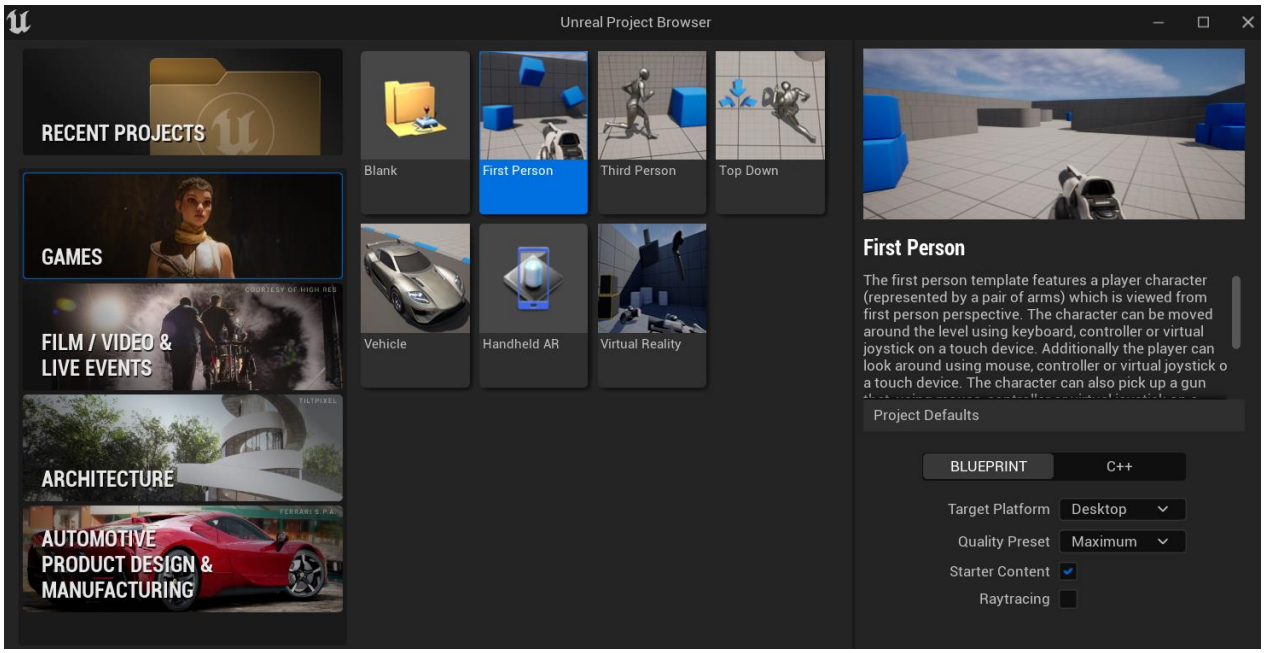


Рис. 3.1 Створення проекту у середовищі Unreal Engine 5

Після цього, необхідно завантажити сцену в якій будуть налаштовуватися усі основні системи та освітлення.

Для даних цілей було обрано готову сцену із замком і лісами під назвою Medieval Castle Modular Vol 1 [17]. У даній сцені є усі важливі елементи, такі як споруди, відкриті простори та різні рівні для максимальної візуалізації налаштувань освітлення та інших систем. Після того, як файли сцени були додані до проекту, залишається завантажити сцену та видалити усі активні системи освітлення, атмосфери та пост-процессінгу, щоб вони не заважали роботі тих, які будуть створені вручну.

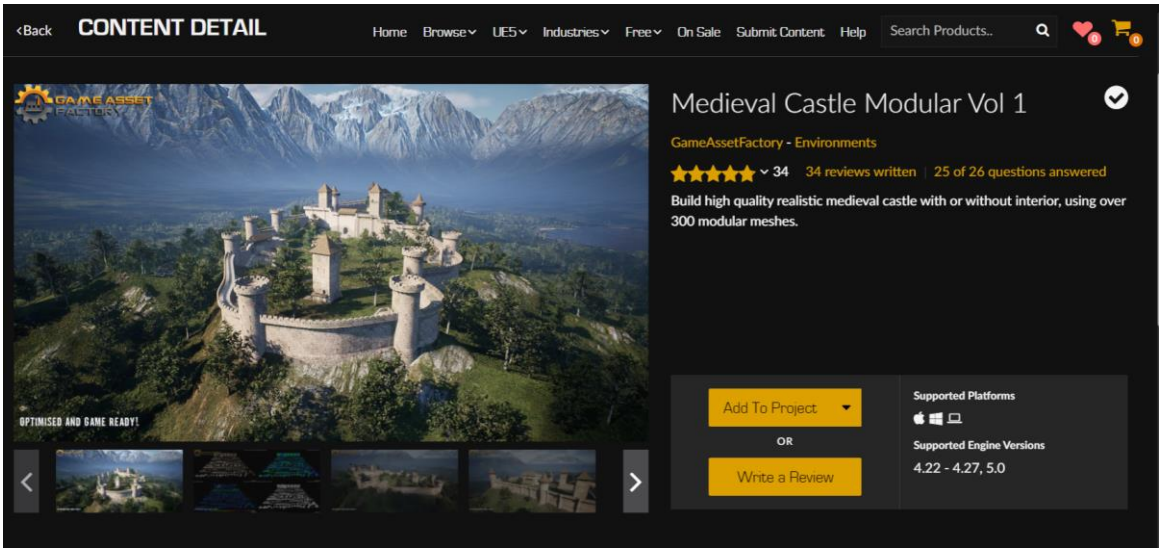


Рис. 3.2 Сцена Medieval Castle Modular Vol 1

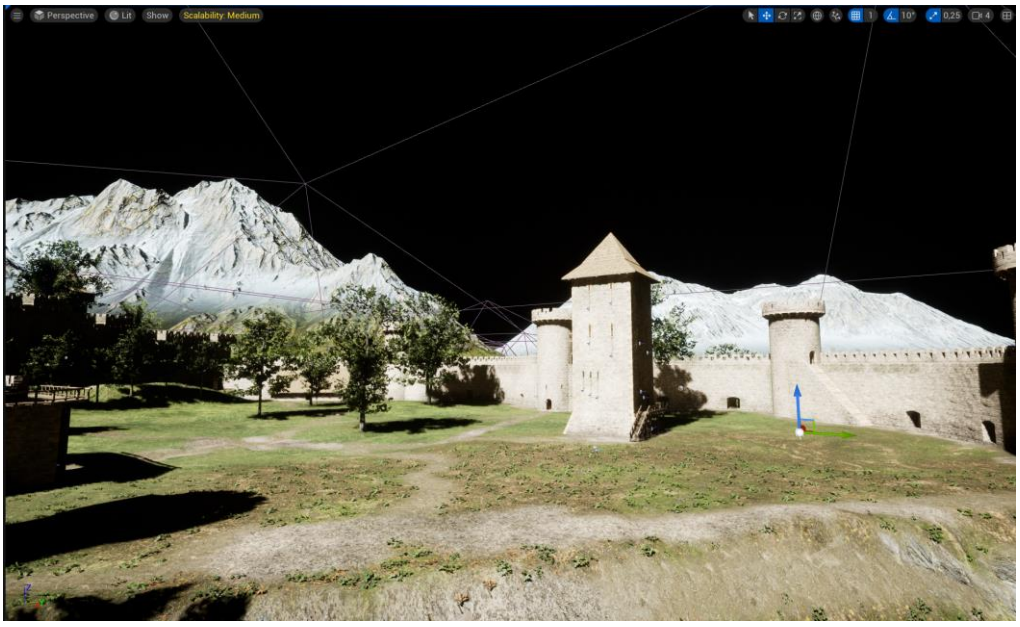


Рис. 3.3 Сцена Medieval Castle із видаленими джерелами освітлення та пост-процессінгу

Останнім кроком, у налаштуваннях проекту виставляємо сцену CF_01_Demo_Scene як основну при запуску програми. Без цього налаштування буде запускатися сцена за замовчуванням, якою наразі є тестова сцена First_Person_Map, яка нам не потрібна. Із цим, сцена готова до експериментів, можна перейти до створення класу безпосередньо системи зміни циклу дня та ночі.

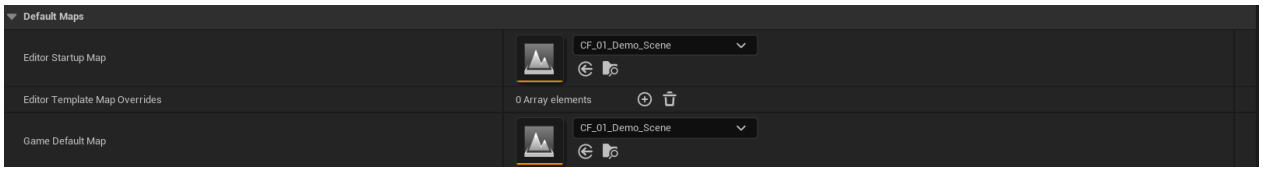


Рис. 3.4 Налаштування сцени за замовчуванням

3.1.2. Створення системи зміни циклу дня та ночі

Для створення системи зміну циклу дня та ночі буде зручно використати інтегрований у Unreal Engine 5 плагін Sun Position Calculator [18]. Це доволі зручний плагін, який дозволяє задавати позицію сонця в залежності від координат потрібного місця, часу та дати. Хоч він і доволі лімітований у певних аспектах, на його базі доволі зручно зробити точну і ефективну систему зміни циклу доби, а також згодом на його базі можна буде створити симуляцію погодних умов.

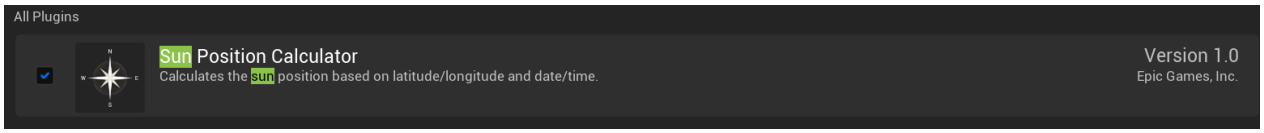


Рис. 3.5 Плагін Sun Position Calculator

Для початку, треба знайти Blueprint – файл, який містить реалізацію плагіну Sun Position Calculator. Після цього, створюється копія цього файлу, для того щоб налаштувати його відносно до потреб програми. Готовий файл треба перейменувати на BP_Custom_Sun_SkyNew, для більш зручного доступу до нього. Після того, як копія створена, її треба перетягнути на сцену.

Із перетягнутим класом плагіну, сцена тепер має більш ор менш адекватне освітлення та активне джерело світла – сонце.

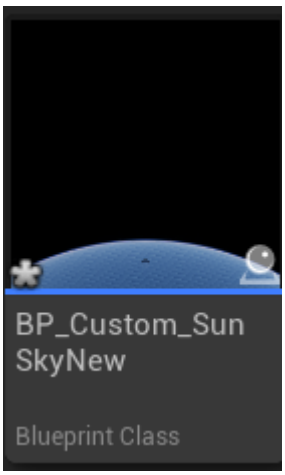


Рис. 3.6 Файл класу плагіну Sun Position Calculator

Рис. 3.7 Сцена із активним класом BP_Custom_Sun_SkyNew

Наступний крок – налаштування часу. Час за замовчуванням плагіну вимірюється змінною Solar Time, із значеннями від 0 до 22. Такий метод виміру часу не є оптимальний для цілей програми, тому треба змінити систему виміру на більш зручну – а саме на стандартні години, хвилини та секунди.

Для цього, у графі подій треба замінити float змінну Solar time на три integer змінні Hours, Minutes, Seconds відповідно.

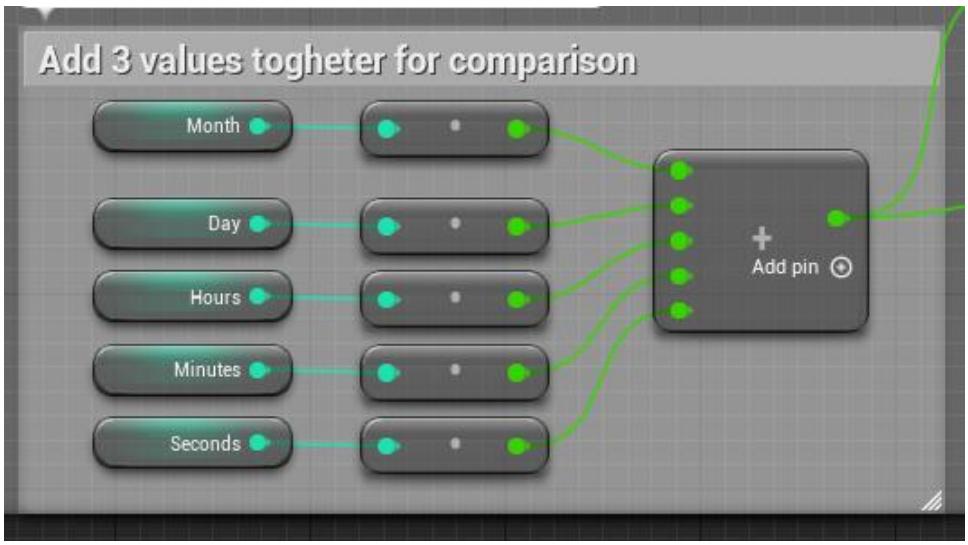


Рис. 3.8 Заміна Solar Time на Hours, Minutes, Seconds

Після цього, треба також замінити Solar time у функції Update Sun, яка відповідає за зміну положення сонця відповідно до введених даних.

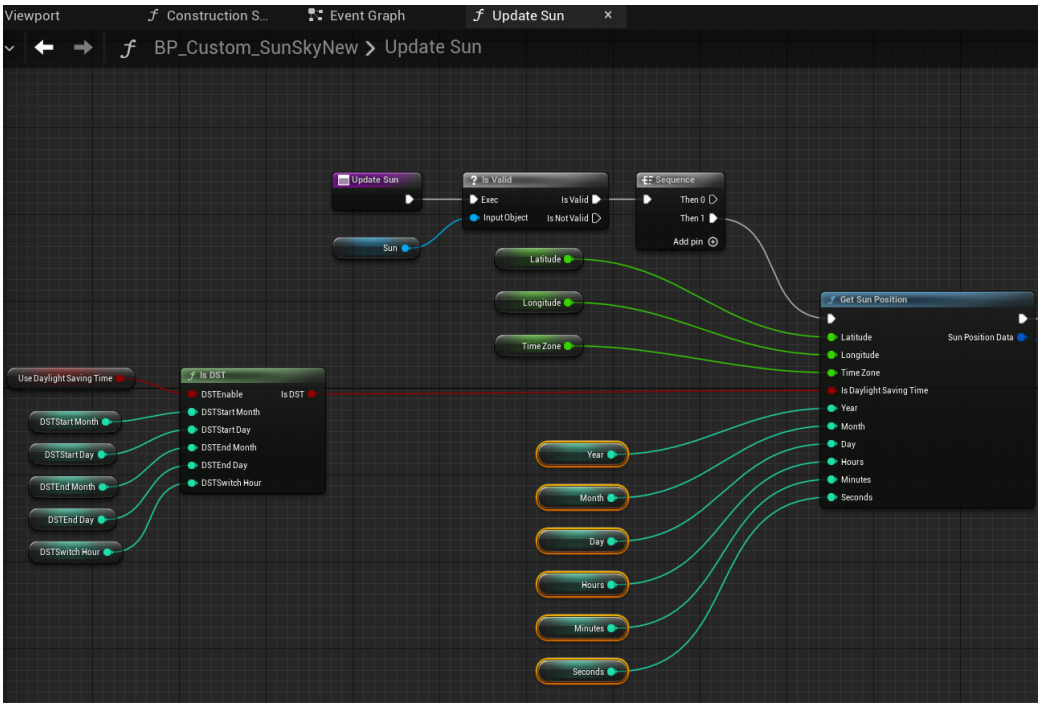


Рис. 3.9 Заміна Solar Time у функції Update_Sun

Остання функція, у якій потрібно замінити Solar Time, це функція Is DST (Is Daylight Saving Time), яка відповідає за Літній час.

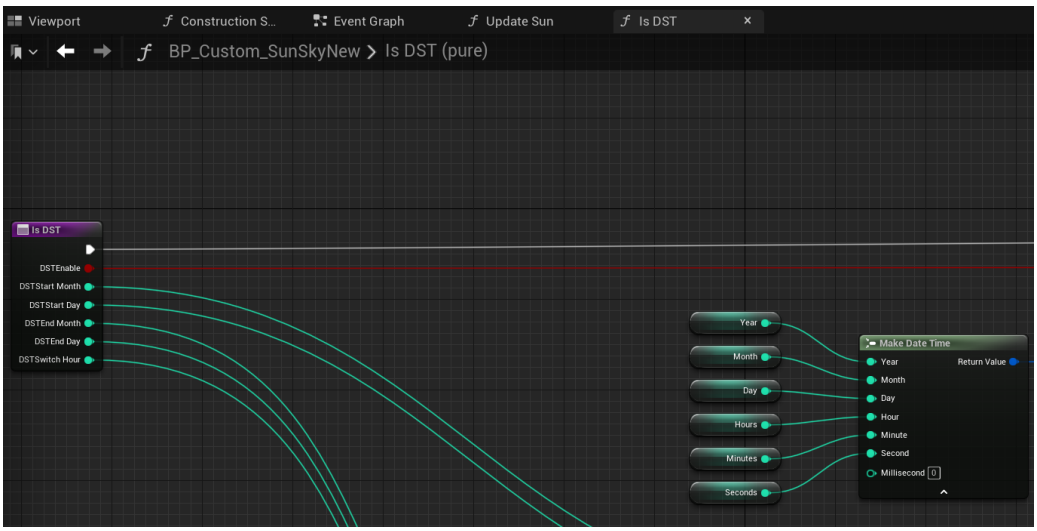


Рис. 3.10 Заміна Solar Time у функції Is DST

Наступне, що необхідно створити, це місяць. За замовчуванням, у класі Sun Position Calculator існує лише сонце, яке повністю зникає при нічному часі. Це треба виправити, тому треба створити ще один Directional Light, який буде прив`язаний до оригінального. Для уникнення плутанини, оригінальний Directional Light перейменовується на Sun, і підв'язаним до нього створюється Directional light, який треба перейменувати у Moon. Для Moon виставляється значення інтенсивності сяйва 0.6 lux, щоб він був темніший за сонце (у сонця 3.16 lux), а поворот по осі Y виставляється у 180 градусів, щоб місяць показувався тільки тоді, коли заходить сонце.

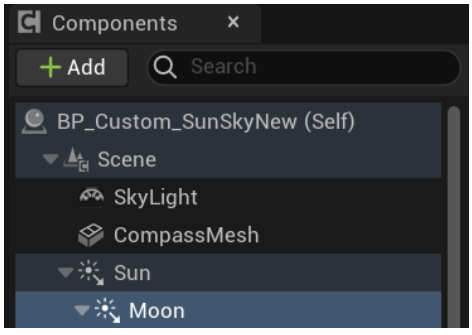


Рис. 3.11 Створення додаткового Directional Light, та перейменування обох у Sun та Moon

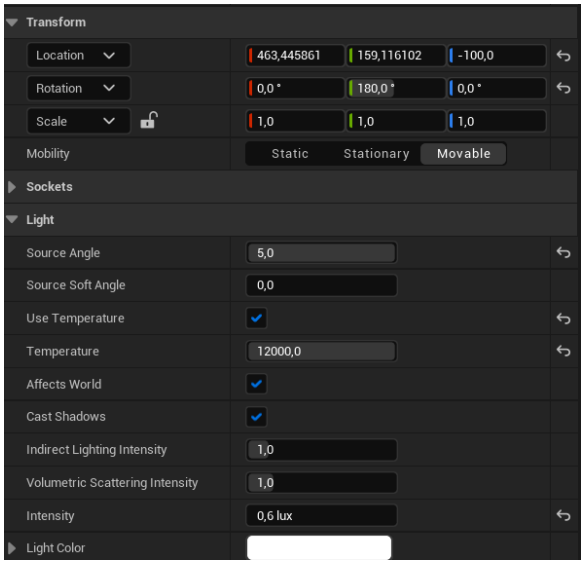


Рис. 3.12 Налаштування Moon

Наступний крок – створення Custom Game State – «стану гри», який допоможе впливати на рух сонця та місяця. Стан гри являється Blueprint класом. В заданому стані гри створюється система, яка регулює положення сонця (і, відповідно, місяця) в залежності від заданих значень Hours, Minutes, Seconds, та задається залежність секунд від тіків системи, що дає можливість регулювати швидкість плину часу, та, відповідно, зміну циклу дня та ночі. У подальшому, можна буде використовувати змінну Amount of Seconds to Add Per Tick для контроля плину часу.

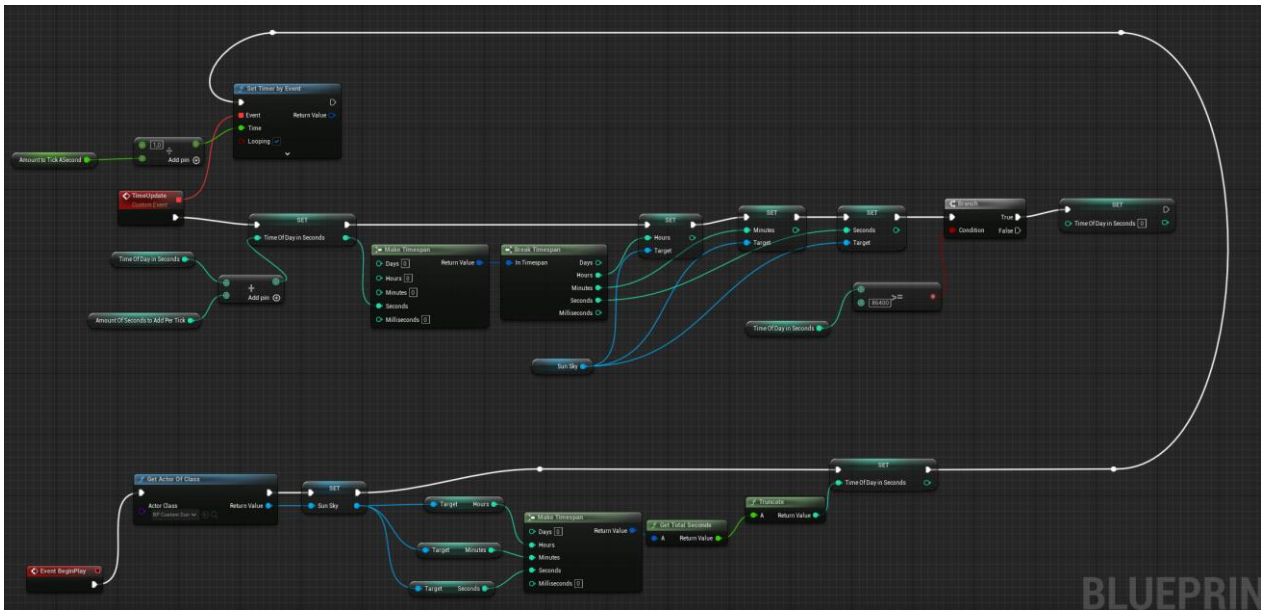


Рис. 3.13 Клас Custom Game State

Після того як клас Bp_Custom_Game_State створений, треба активувати його, змінивши у поточному Game Mode. Для цього, в налаштуваннях World Settings, потрібно змінити стан гри за замовчуванням на Bp_Custom_Game_State.

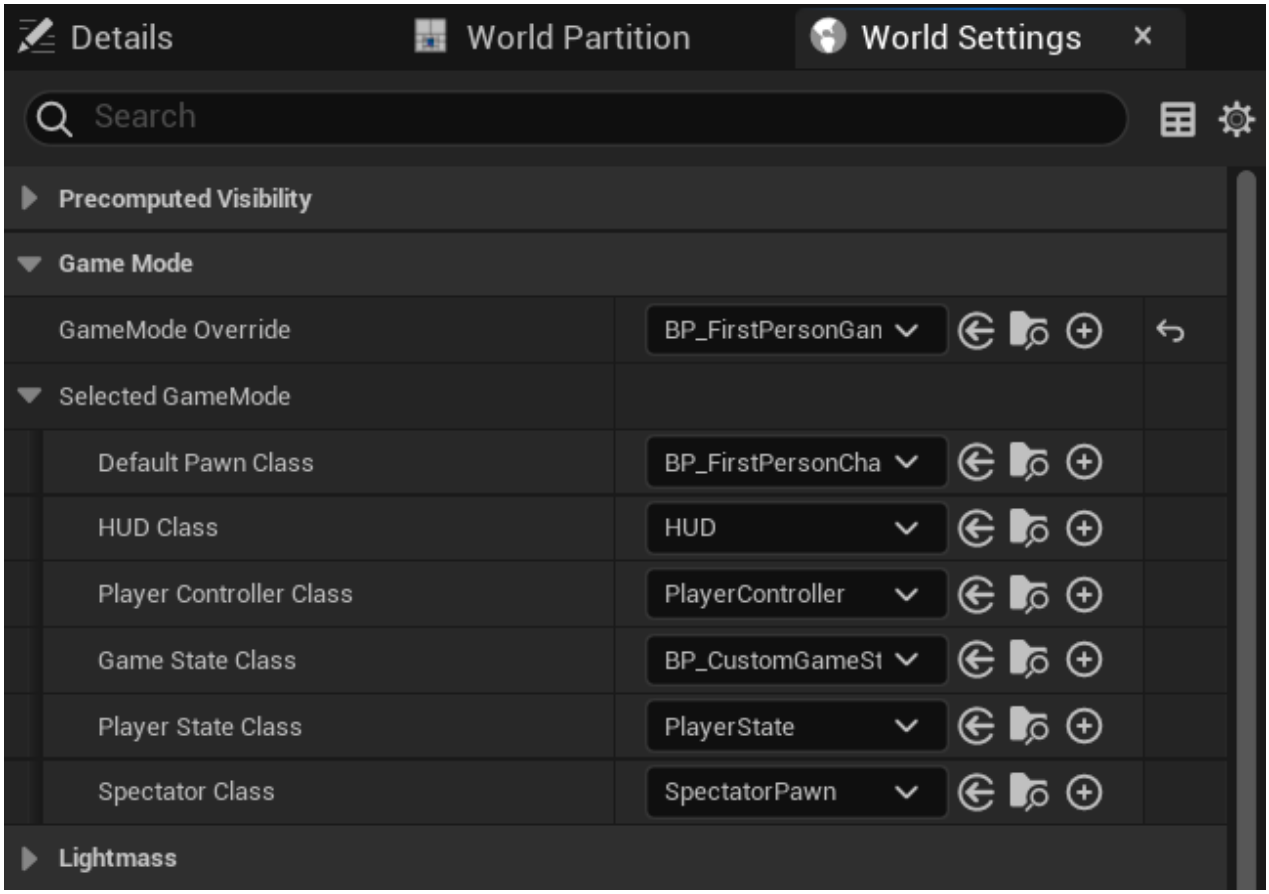


Рис. 3.14 Налаштування World Settings

Останнім кроком є створення зірок для нічного неба. Для цього треба використати прозорий матеріал зірок та інвертовану сферу навколо неба. Сфера прив’язується до місяця, щоб зірки рухались одночасно із місяцем, і нічним небом відповідно, а матеріал налаштовується таким чином, щоб його прозорість корелювала напряму із освітленістю навколишнього середовища – таким чином, кожен раз, коли освітленість збільшується (тобто настає день), зірки стають більш прозорими, доки не стають невидимими цілком.

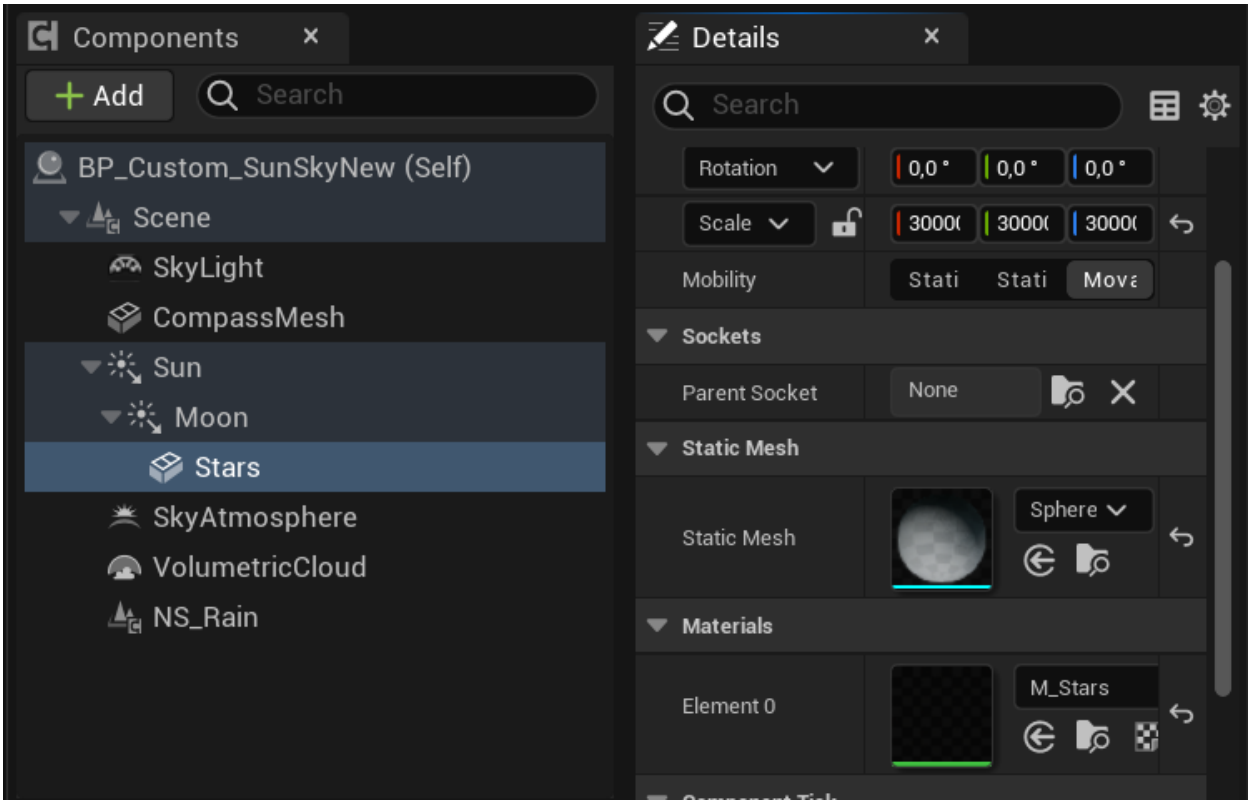


Рис. 3.15 Сфера Stars, підв’язана до місяця

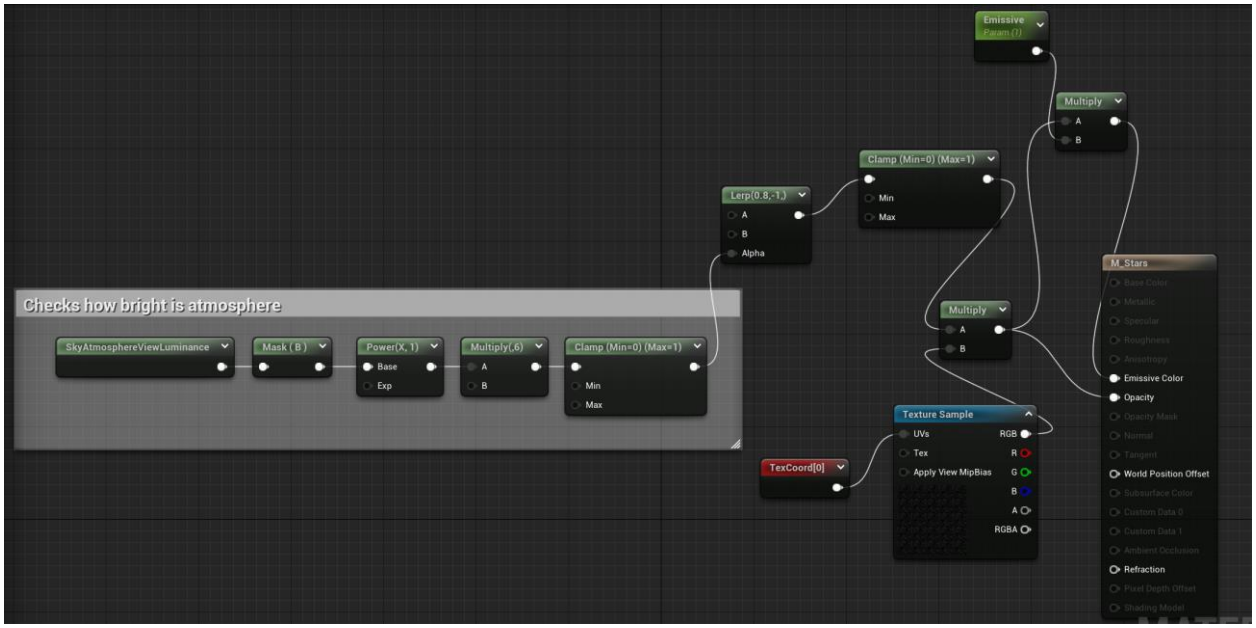


Рис. 3.16 Матеріал Stars, який регулює прозорість зірок

Таким чином, система зміни циклю дня та ночі створена, автоматизована, та регулюється змінними часу, дати та локації, а також мають регулювання швидкості плину часу змінною, якою можна буде скористатися у майбутньому.

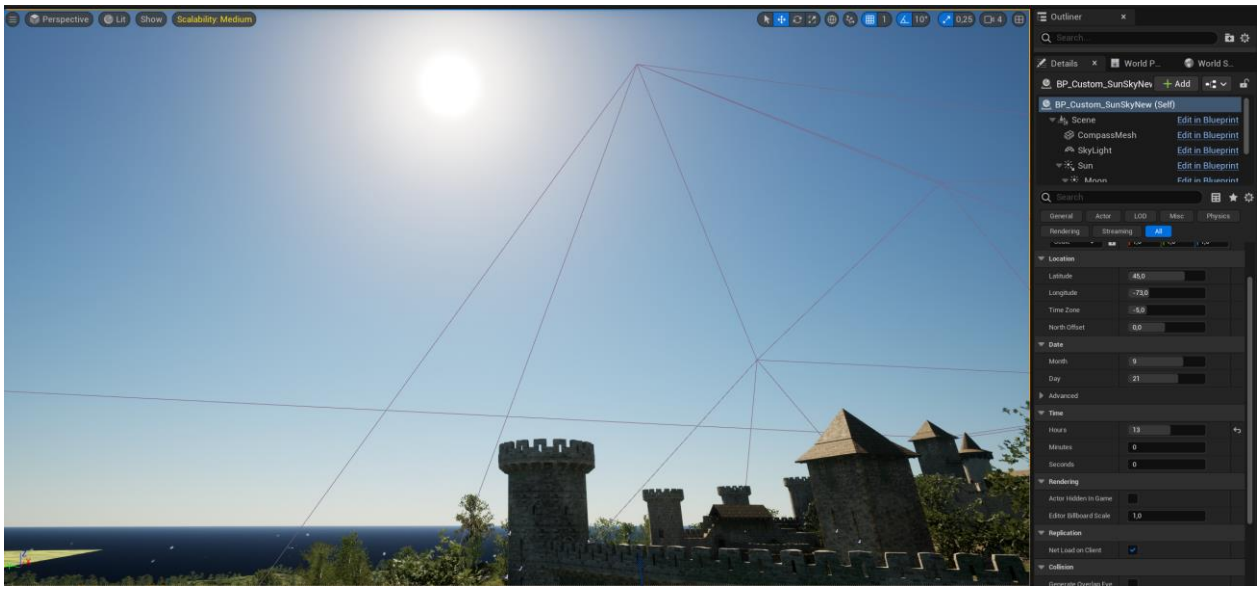


Рис. 3.17 Вигляд сцени, 13 година дня

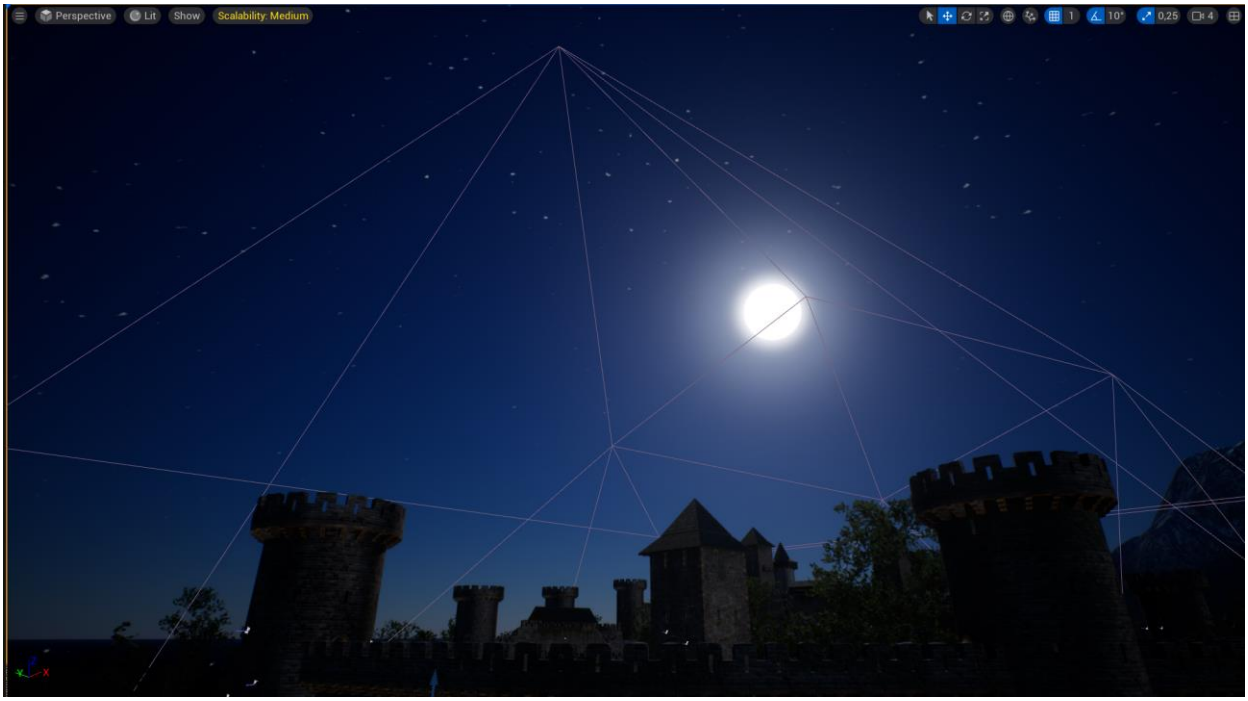


Рис. 3.18 Вигляд сцени, 4 година ночі

3.1.3. Volumetric clouds

Для забезпечення більшої реалістичності неба треба створити хмари. Найкращим і сучасним варіантом є Volumetric clouds – об’ємні хмари. Вони є більш сучасним варіантом генерації хмар, який дозволяє не тільки надати небу більшої реалістичності, а й детально налаштовувати вигляд, освітлення та

навіть фізику хмар. Такий ефект досягається завдяки спеціальному матеріалу, який використовують ці хмари, текстуру нормалей, яка детально дає зрозуміти рушію, як він повинен будувати ці хмари, та спеціальному об’ємному типу текстур – Voronoi Noise[19]. Ця текстура шуму ідеально підходить до детальних налаштувань хвилястих хмар, хмар їдких речовин та інших подібних патернів, які можна знайти у природі. Інші назви, які використовуються для опису подібного шуму, це Worley або Клітинний шум, але усі вони є одним і тим самим.

Для початку, треба додати компонент Volumetric Cloud до класу BP_Custom_SunSkyNew.

Рис. 3.19 Пустеля створена за допомогою Voronoi Texture

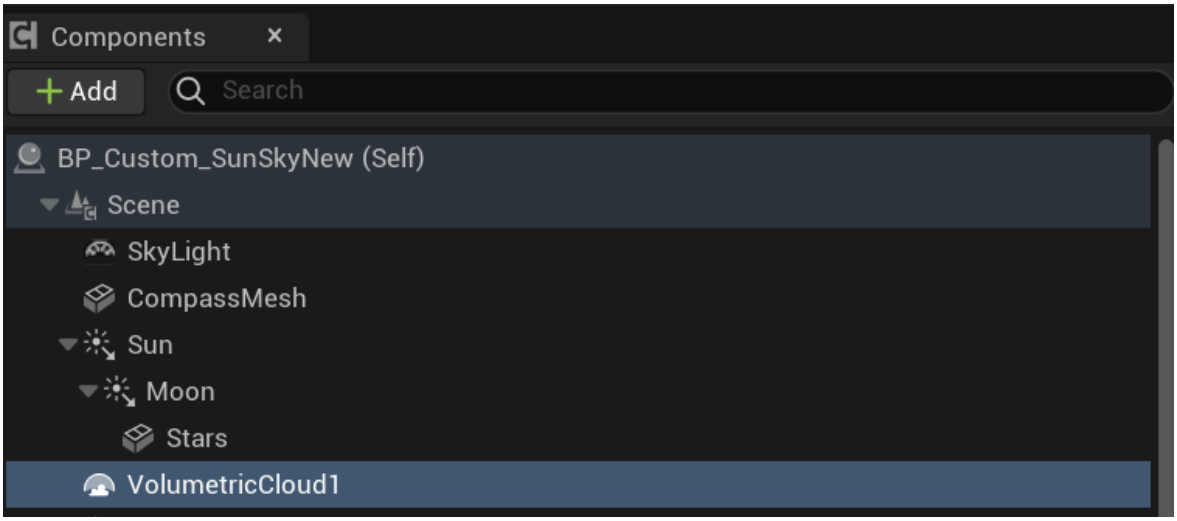


Рис. 3.20 Компоненти класу BP_Custom_SunSkyNew

Тепер на сцені з’явилися хмари, які тепер можна детально налаштовувати. Перш за все, треба налаштувати базове відображення хмар для звичайного ясного дня. Для цього треба зменшити густоту та кількість хмар у небі. Усі налаштування, які стосуються об’ємних хмар, змінюються у базових налаштуваннях компоненту хмар на рівні, та в матеріалі хмар. Налаштування компоненту об’ємних хмар достатньо лімітовані і по суті відповідають тільки за висоту, рівень та рівень деталізації хмар. Усі найбільш детальні налаштування знаходяться у змінних матеріалу хмар. У подальшому ці змінні також можна буде використати задля створення хмарного неба для дощу.

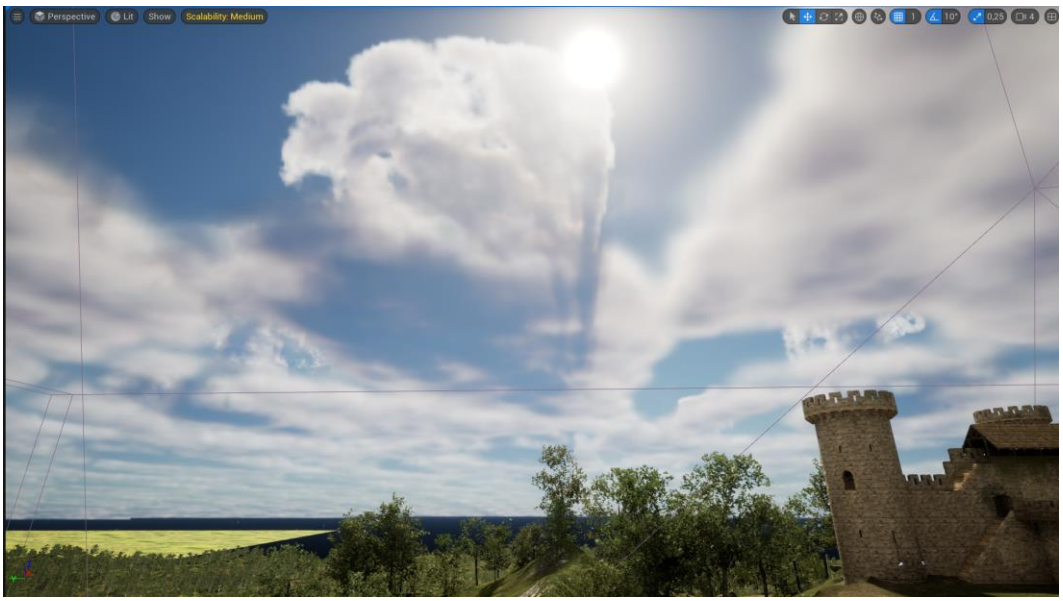


Рис. 3.21 Вигляд об’ємних хмар за замовчуванням

Для початку треба розглянути матеріал хмар, який використовується. Варіант за замовчуванням виглядає доволі лякаюче, але насправді містить усього три основні функції, які відповідають за модифікацію вигляду об’ємних хмар. Для більш адекватного сприйняття коду, треба чітко позначити вищезгадані функції.

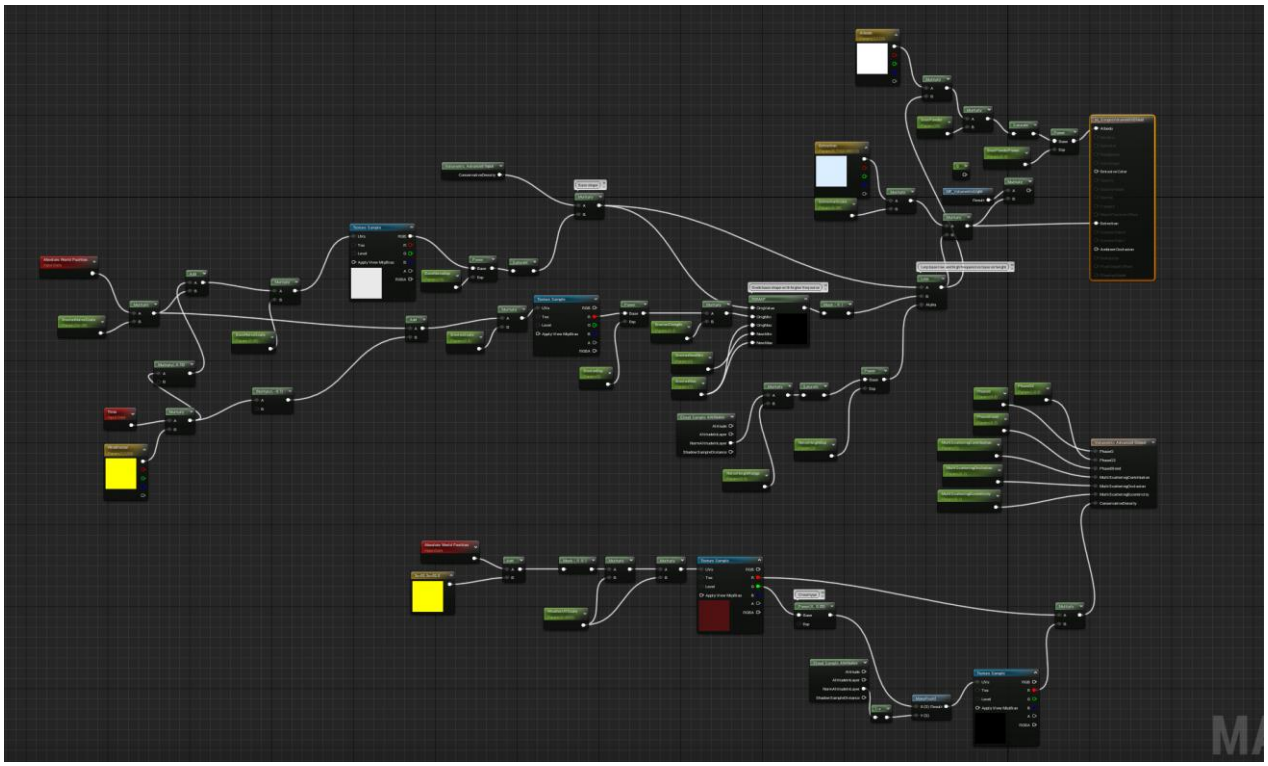


Рис. 3.22 Матеріал хмар за замовчуванням

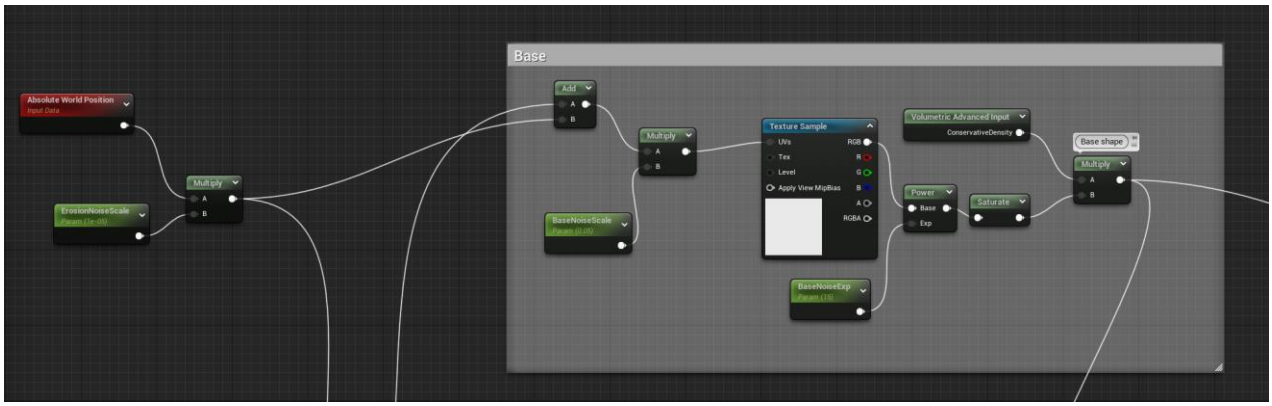


Рис. 3.23 Матеріал хмар за замовчуванням

Перша із функцій матеріалу – базова форма хмар. Ця функція відповідає за надання текстури та базової форми об’єктам хмар, це самі примітиви, які

здебільшого просто є формою, яку вже згодом деталізують інші функції. Текстура базової форми містить нормалі, які показують, як будуть будуватися ці хмари. Ця текстура має два основні канали, червоний відповідає за основний рівень хмар, тоді як зелений канал відповідає за місцеву висоту хмар.

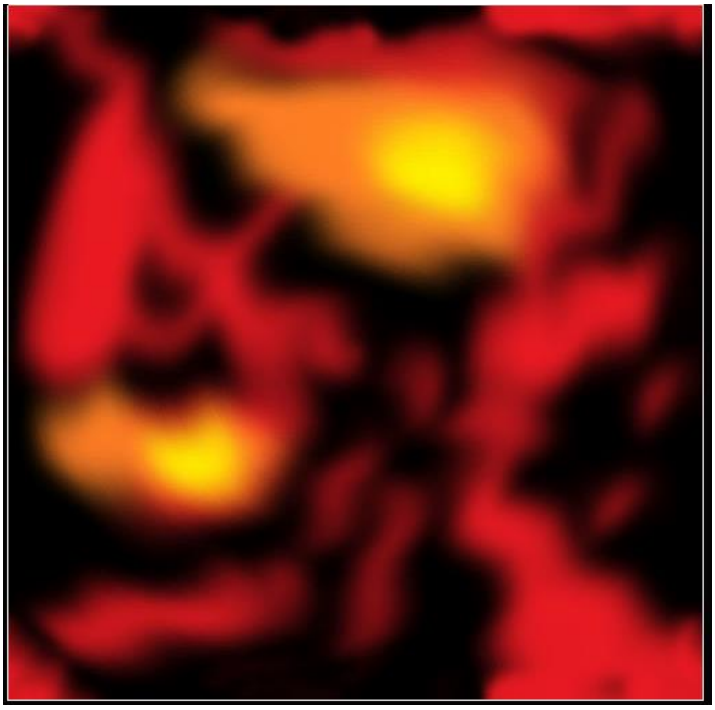


Рис. 3.24 Текстура, що відповідає за форму об’ємних хмар

Наступна функція – Noise - відповідає за деталізацію хмар. Без цієї функції хмари будуть виглядати як шматки вати або морозива, без присутніх реальним хмарам візерунків.

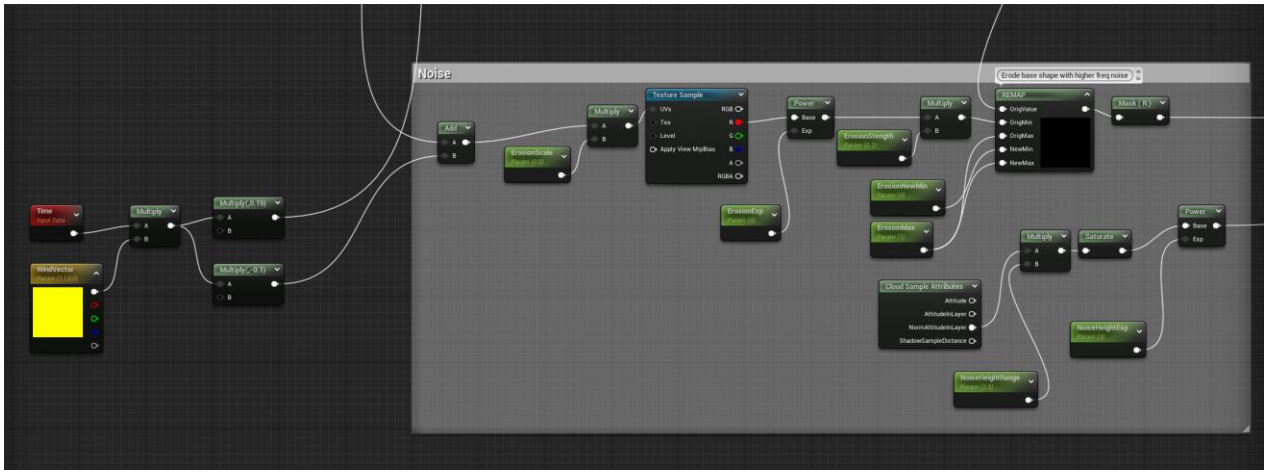


Рис. 3.25 Функція Noise

Тоді як правильні налаштування роблять хмари реалістичнішими, надмірне використання цієї функції може призвести до неприємних результатів. Ця функція є вирішальною перевагою цих хмар перед будь-якими іншими, адже можливість детально налаштовувати хмари у реальному часі із таким об’ємом опцій робить створення погодних умов куди більш аутентичним, дружнім до користувача.

Рис. 3.26 Вигляд хмар відповідно до впливу функції Noise: 1 - без функції, 2 – із якісно налаштованою фукнцією, 3 – із неякісно налаштованою функцією

Нарешті, фінальна функція, яка є безпосереднім контриб’ютором до створення анімацій і більш детального налаштування формації хмар уцілом, є Advanced Output. Advanced Output дозволяє розширити діапазон опцій, які надають широкий функціонал для редагування навіть готових хмар. За допомогою цієї функції, такі параметри як Extinction дозволяють змінювати форму хмар не дивлячись на базову форму, а WeatherUVScale надає можливість редагувати відображення хмар згідно до створенних погодніх умов. Саме ця функція безпосередньо буде використана для забезпечення симуляції погодних умов, а саме дощу у цьому проекті.

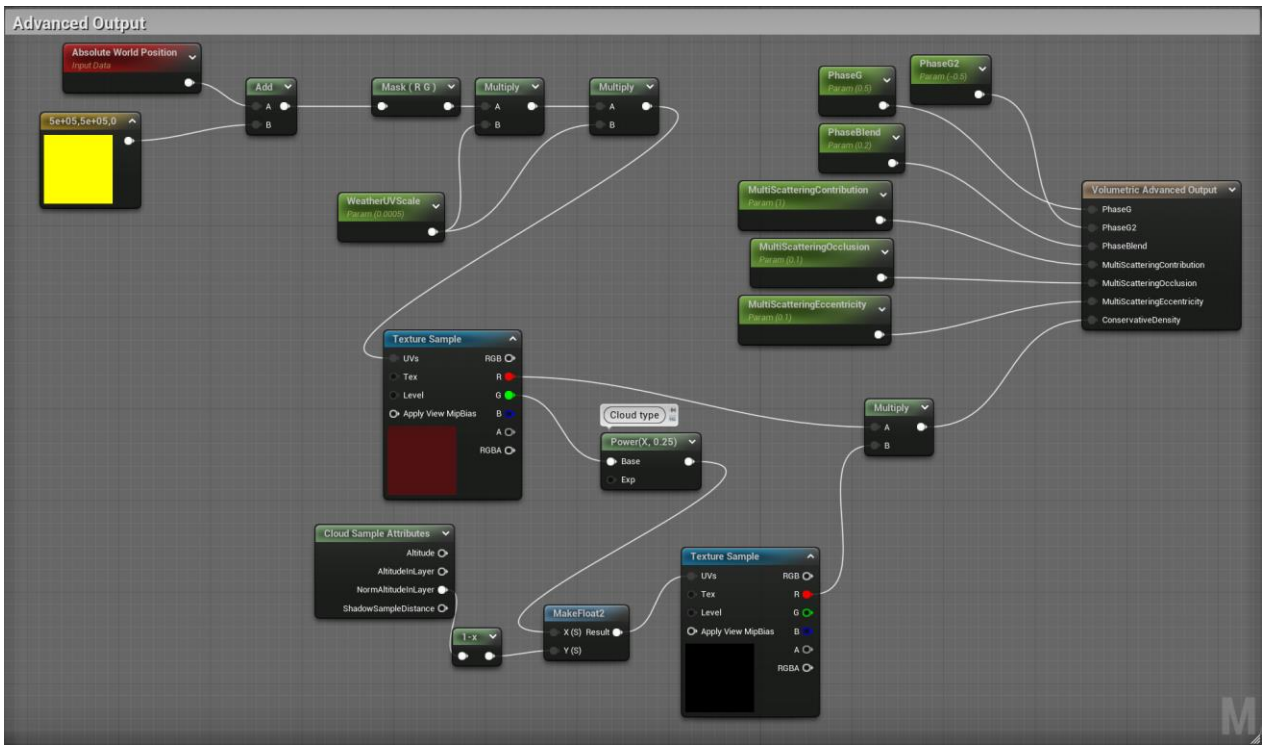


Рис. 3.27 Функція Advanced Output

Наступним кроком, треба створити інстанс матеріалу, який ми будемо використовувати у подальшому. Інстанс матеріалу – це «запакований» клас матеріалу у об’єкту, в якому можна редагувати public значення класу в режимі реального часу, не змінюючи при цьому коду оригінального класу. Такий підхід забезпечує більш ефективний та безпечний спосіб маніпулювання виглядом, а відповідно, матеріалом хмари, аніж постійне втручання до коду джерела матеріалу. Після того, як інстанс створений, у його опціях можна гнучко налашатувати вигляд хмар, які у результаті будуть використані у проекті. Для початку, збільшується вплив фунції Noise на хмари, задля більш монументального та реалістичного сприйняття. Наступним кроком, зменшується Extinction, щоб вивільнити більший об’єм неба, адже погода за замовчуванням повинна бути ясна, із малою кількістю хмар. Останнім кроком налаштовується WeatherUVScale, який робить хмари більш об’ємними, і буде корегувати їх у майбутньому при зміні погоди.

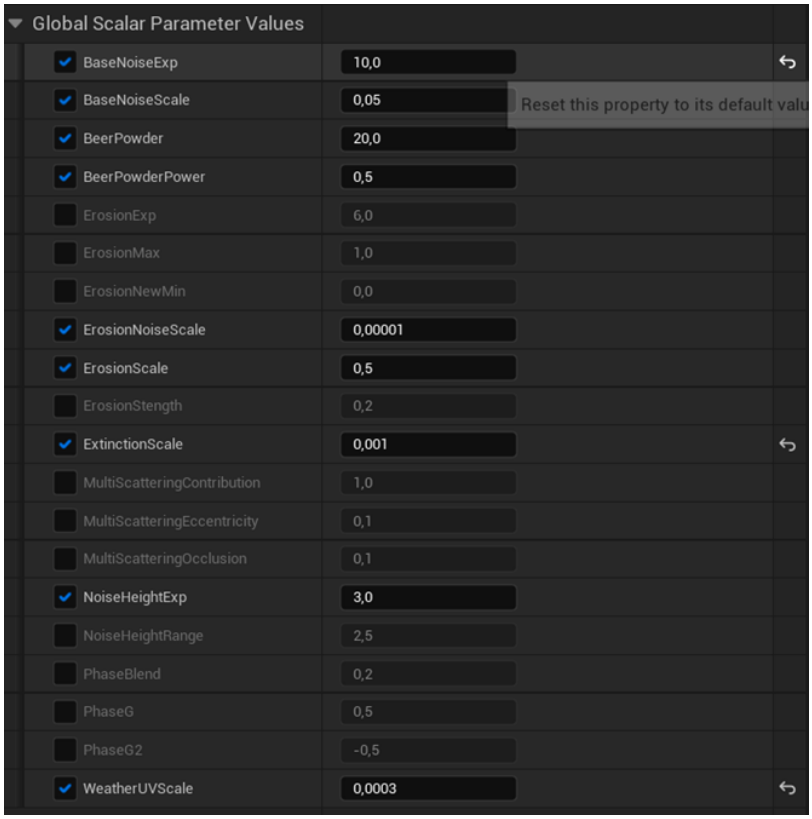


Рис. 3.28 Налаштування об’ємної хмари

Останнім кроком, налаштовується WindVector хмари. Треба змінити значення векторів R та G, щоб «хвилясті» рухи хмар відображалися у напрямку одночасно осей X та Y. Ця опція дозволяє надати хмарам анімацію руху, що створює ілюзію «живих» хмар, які постійно рухаються по небу.

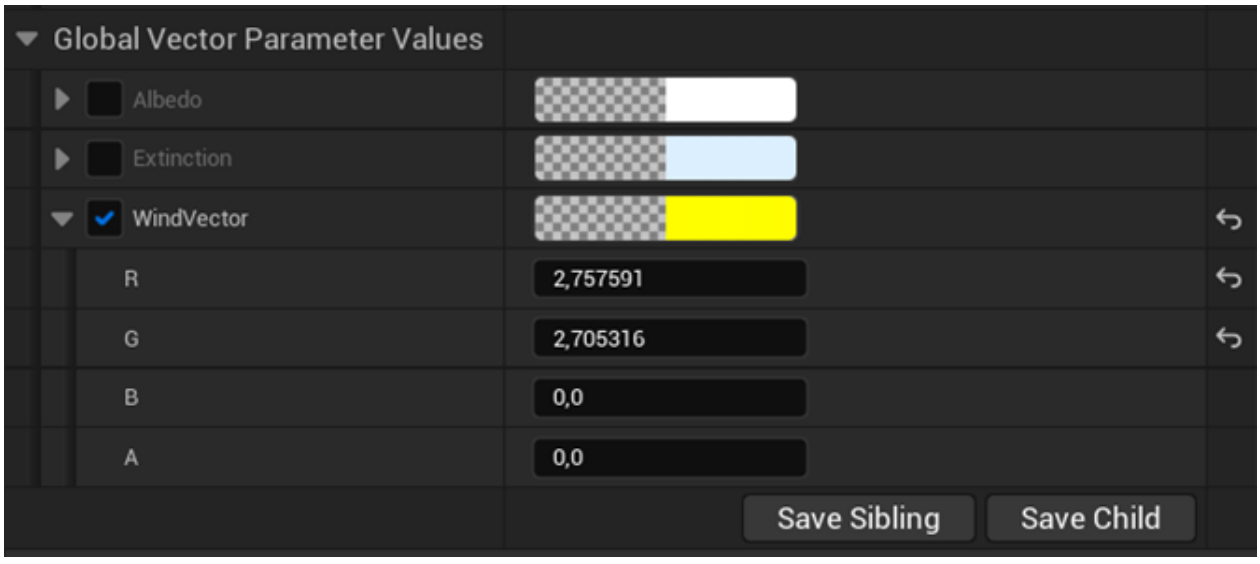


Рис. 3.29 Налаштування векторів вітру, які створюють ілюзію переміщення хмар

Після усіх налаштувань, створено хмари, які виглядають доволі реалістично, і відповідають поставленим цілям.

Рис. 3.30 Готовий вигляд налаштованих хмар

3.1.4. Створення дощу

Для створення дощу перш за все створюється система часток, яка буде генерувати каплі дощу, із змінними, які напряму впливають на параметри дощу, такі як швидкість падіння капель, кількість створюваних капель, час циклу їх існування, колізія та ін. У попередніх версіях Unreal Engine використовувалась система Participle system, буквально «Система часток», але наразі вона замінена на більш сучасну систему Niagara. Тоді як Participle system мала лімітовані можливості, а також деякі доволі незручні особливості, уникнення яких могло зайняти доволі великий час. Одним із таких недоліків було зникнення відображення ефектів під певним кутом, що призводило до спеціальних налаштувань суто задля уникнення дефектів. Niagara виправила ці недоліки, а також покращила швидкодію, створила більш зручний та інтуїтивний, дружній до користувача інтерфейс, і застримлайнила використання створених користувачем змінних для маніпуляції системою часток.

Перш за все, створюється клас Niagara System, який називається NS_Rain. Об’єкт цього класу додається до класу BP_Custom_SunSkyNew, таким чином буде зручно їм управляти у гармонії із іншими об’єктами класу.

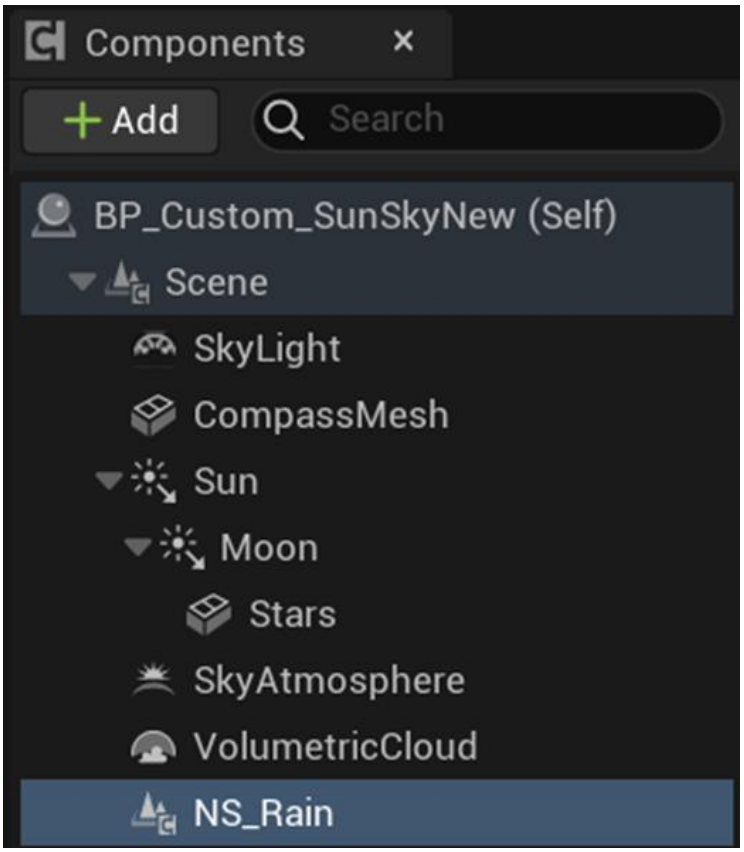


Рис. 3.31 Об’єкт класу NS_Rain, створений у класі BP_Custom_SunSkyNew

Після цього, у графі класу створюється два «емітери», функції що відповідають за створення ефектів. Зазвичай, емітери мають роль «спавнерів», або генераторів спецефектів, із заданим матеріалом та математичними формулами, що описують алгоритм роботи цих ефектів. У даному випадку, матеріал залишається за замовчуванням, адже для крапель дощу цілком підходить стандартний білий матеріал, а алгоритми будуть налаштовані таким чином, щоб відображення падаючих крапель було максимально реалістичним.

Перший емітер, Rain_Streak, відповідає за задану область, у межах якої генеруються краплі краплі дощу, що поводять себе так, як налаштовано заданими параметрами. Другий емітер, Hanging_Particulates, генерує сплески крапель у місці колізії крапель, що створені першим емітером. Колізія

відбувається із будь яким іншим предметом із колізію, будь то дорога, будинок, гравець, тощо.

Рис. 3.32 Граф класу Niagara System NS_Rain

Наступним кроком створюється користувацька змінна User_SpawnRate. Ця змінна буде використовуватися для зміни кількості згенерованих капель у певний період часу, що дозволить керувати силою дощу, використовуючи цю змінну.

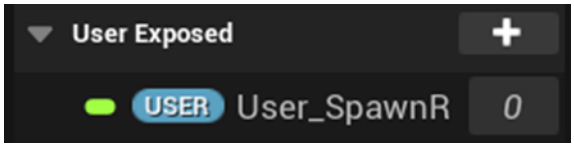


Рис. 3.33 Користувацька змінна User_SpawnRate у NS_Rain

Тепер можна перейти безпосередньо до налаштування емітерів. Перш за все, треба налаштувати емітер створення самого дощу, тобто падаючих крапель із неба. Для цього треба налаштувати емітер Rain_Streak. Першим параметром буде Initialize Particle. Так, як краплі дощу випаровуються із різною швидкістю, Lifetime Mode виставляється Random, щоб кожна крапля мала індивідуальний «період життя». Однак, щоб уникнути цілком неконтрольованих крапель, ліміт, у якому генеруються ці випадкові значення, виставляється між 1.4 та 1.75 секундами. Такі ж самі налаштування створюються для маси часток, у ліміті між 0.75 та 1.25 секундами. Колір залишається білим, він підходить до дощу. Розмір крапель-часток виставляється як Non-Uniform, із значеннями 1.0 по осі X та 50.0 по осі Y. Поворот часток також виставляється як Random, для створення ілюзії різного напрямку дощу.

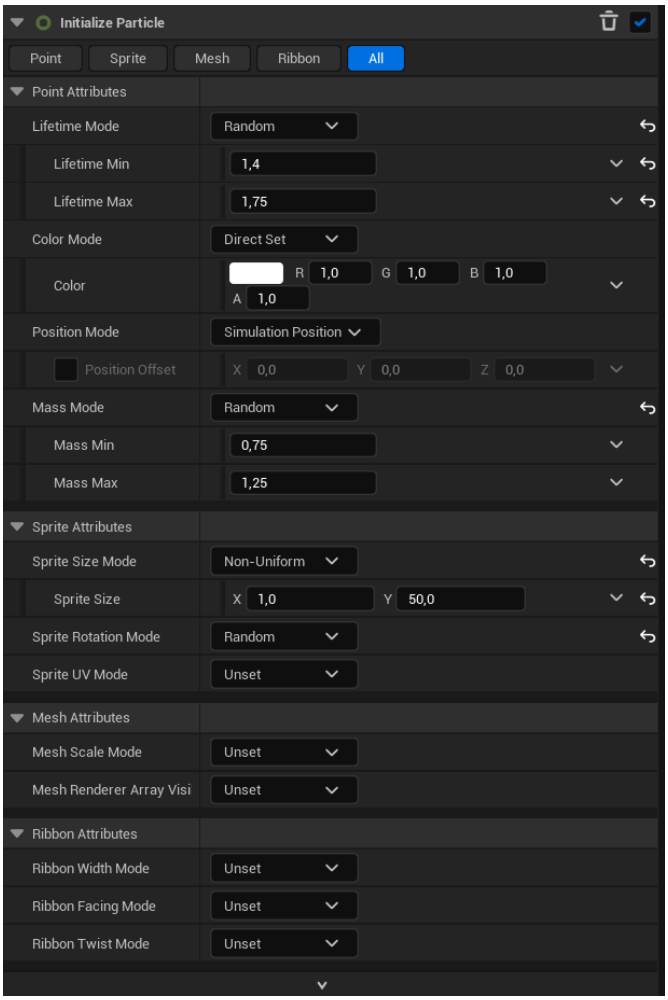


Рис. 3.34 Налаштування Initialize Particle у емітері Rain_Streak

Наступний крок – створення області, у якій буде йти дощ. Для найкращої демонстрації можливостей проекту, достатньо території замку, створення більшої території дощу буде негативно впливати на швидкодію проекту. Для цих налаштувань використовується опція Shape Location. Для того щоб легше покрити периметр замку, обирається примітив Box, щоб область дощу мала прямокутну форму, розмір задається 10000x10000x1000, саме таку площу приблизно має замок, Scale залишається за замовчуванням, адже його збільшення змаштабує краплі дощу, що буде виглядати доволі кепсько.

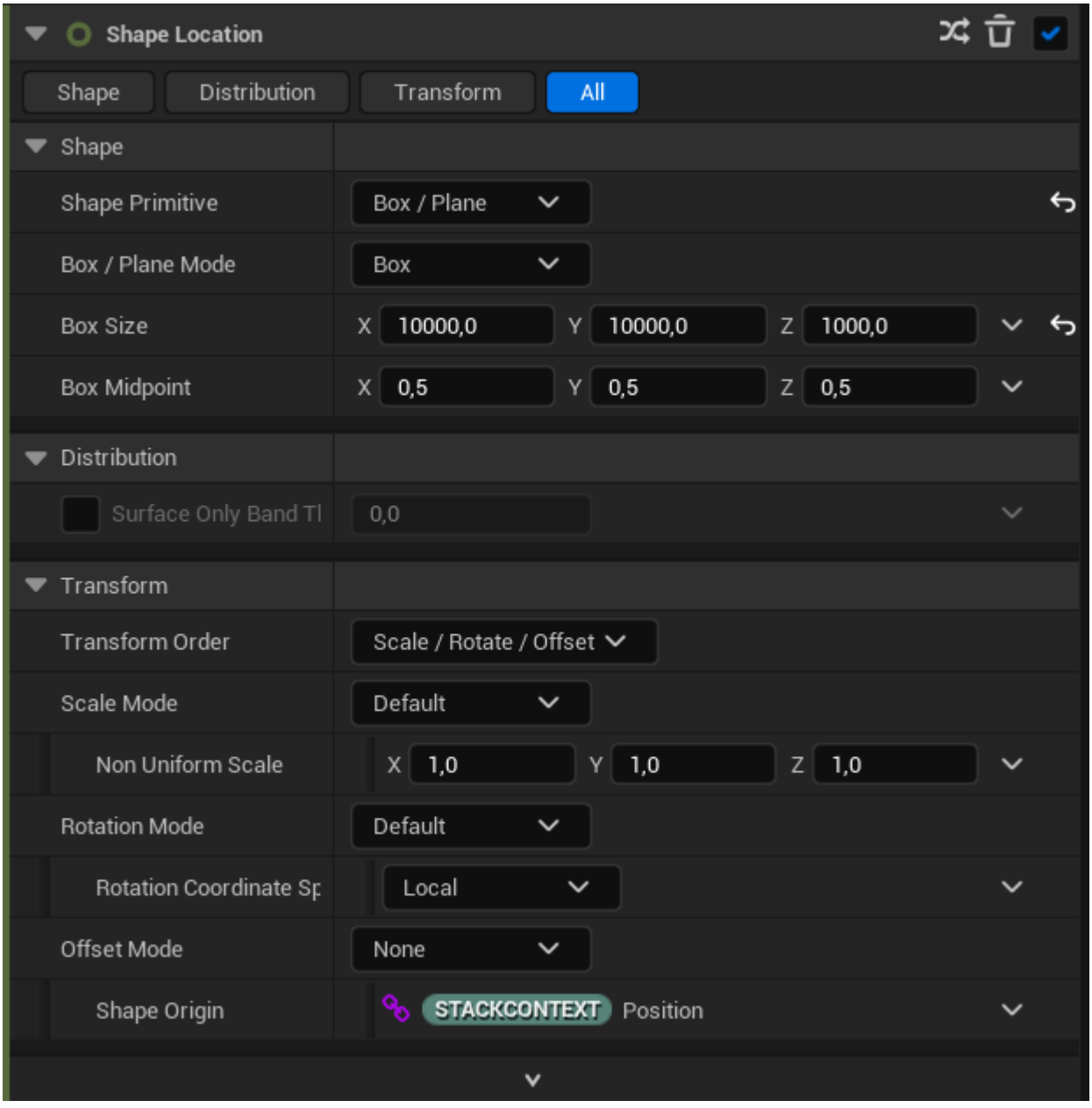


Рис. 3.35 Налаштування Shape Location у емітері Rain_Streak

Після цього, налаштовується параметр Add Velocity, який відповідає за швидкість падіння крапель дощу. Параметр по осі Y виставляється значенням 300, щоб створити ілюзію впливу вітру на напрям падіння крапель. Параметр по осі Z виставляється як -500. Значення 500 відображає швидкість вертикальної зміни координат крапель, а від'ємне значення відображає напрям руху зверху вниз, адже за замовчуванням система часток направляє рух часток із низу уверх.

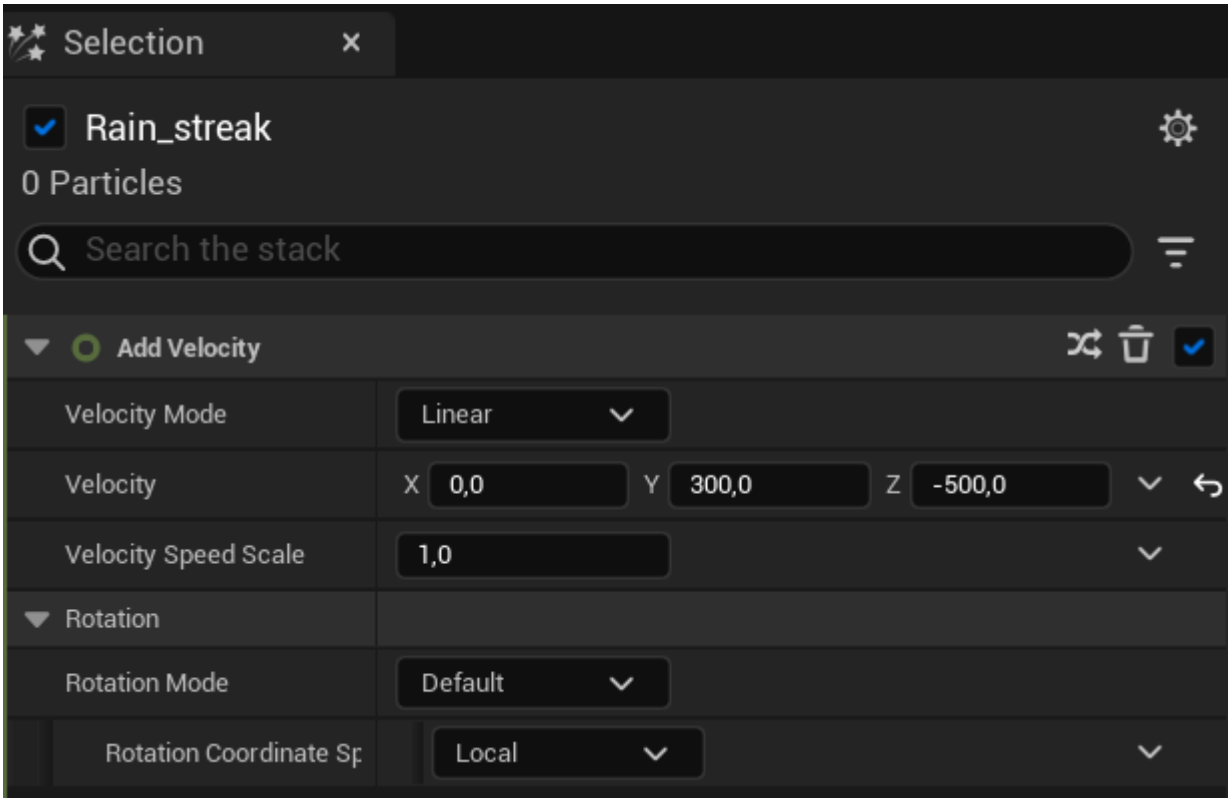


Рис. 3.36 Налаштування Add Velocity у емітері Rain_Streak

Нарешті, остання опція, яку необхідно налаштувати, це Колізія (Colission). Так, як краплі, пролітаючі крізь землю, споруди, гравців виглядають нереалістично та дешево, необхідно налаштувати колізію крапель таким чином, щоб вони зникали при контакті із іншими об'єктами із колізією. У подальшому у момент зникнення крапель при колізії також будуть зроблені сплески крапель, щоб візуалізувати дію колізії.

Для початку, включається Ray Traced тип колізії процесора, для більш точного обрахунку. Після цього, виставляється радіус колізії частки 0.1, радіус

обрахунку колізії виставляється як Sprite, а тип обрахунку колізії виставляється по межам спрайту, щоб колізія траплялася саме при контакті краплі, а не області навколо неї. Далі, налаштовується Bounce, що відповідає за невеликий ефект підскакування краплі після контакту із поверхнею. Restitution виставляється із значенням 0,6.

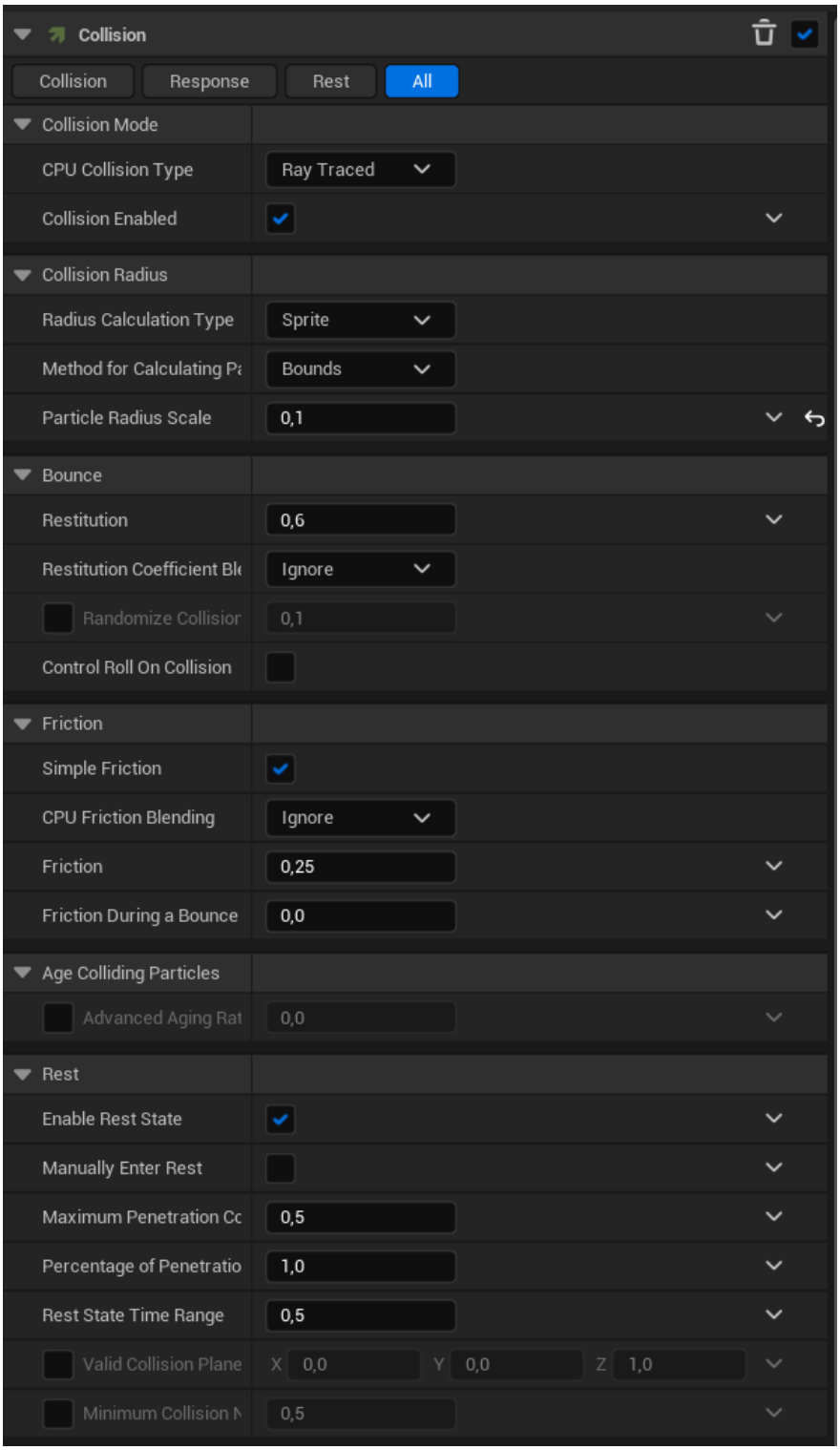


Рис. 3.37 Налаштування Collision у емітері Rain_Streak

Після того, як емітер Rain Streak повністю налаштований, треба налаштувати наступний емітер, який буду генерувати сплески крапель у місці контакту крапель із поверхнею. Для початку, налаштовується параметр Life Cycle у властивості Emitter State. Life Cycle Mode виставляється як Self, поведінка циклу виставляється як Infinite, а тривалість циклу буде 2 секунди. Режим масштабування налаштовується згідно системи.

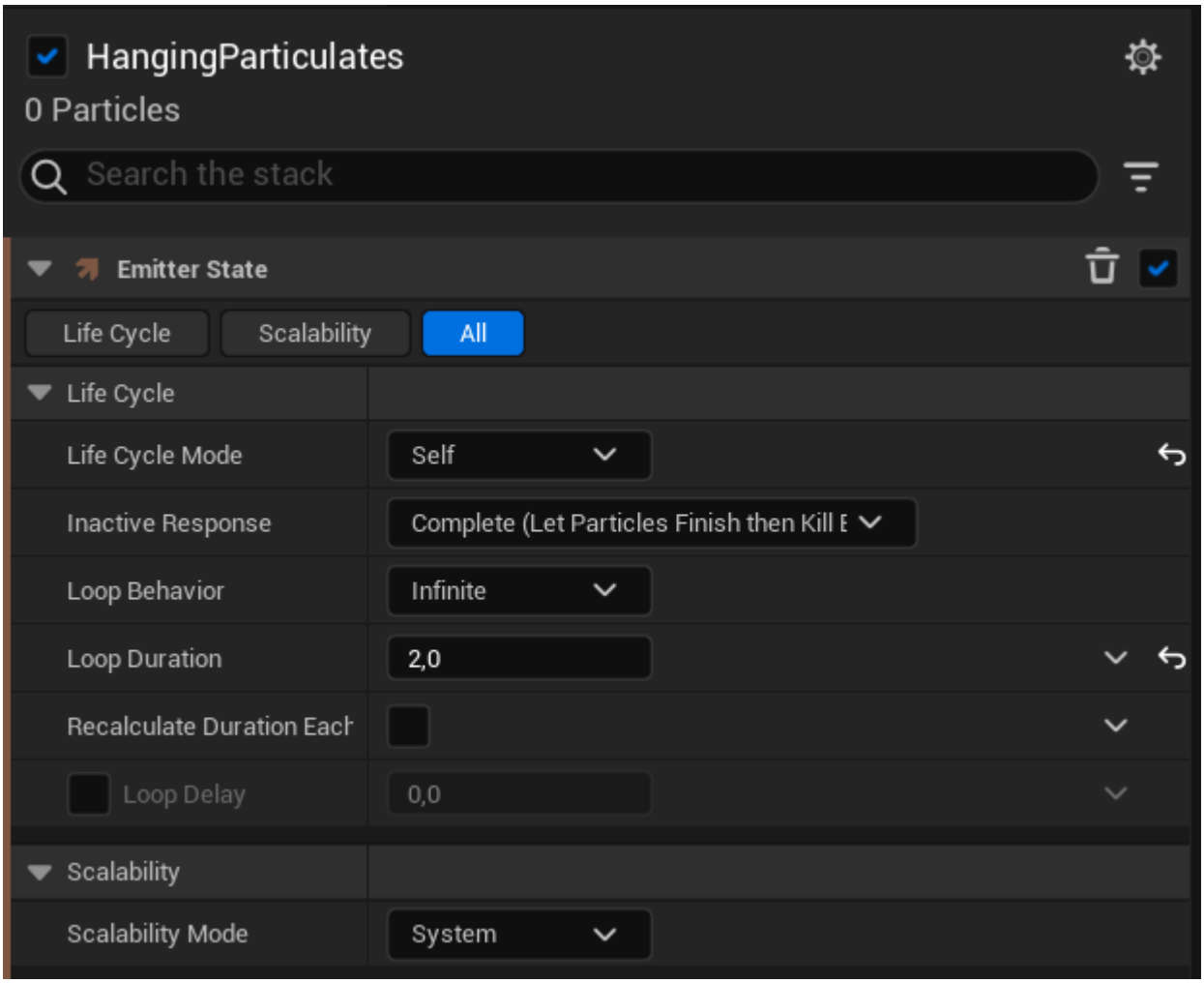


Рис. 3.38 Налаштування Emitter State у емітері Hanging_Particulates

Наступним кроком налаштовується функція Initialize Particle, яка ініціалізує сплески крапель. У розділі Point Attributes, режим життєвого циклу виставляється аналогічно до самих крапель дощу, Random, задля створення більш натурального вигляду випадкових сплесків. Як і для крапель, тривалість життєвого циклу хоч і є випадковою, але виставляється у рамках певного

періоду часу, а саме від 0.5 до 0.7 секунди у даному випадку. Колір залишається білим за замовчуванням, це також цілком задовольняє дизайн цього проекту.

Після цього, налаштовуються атрибути спрайту, тобто самої моделі сплеску. Режим розміру сплесків виставляється як Random Uniform, у проміжку між 2.0 та 3.5 міліметрами. Так само випадковими виставляються повороти та випадковими по осям X/Y – UV режим спрайту. Останній відповідає за координати текстур, які накладаються на спрайти.

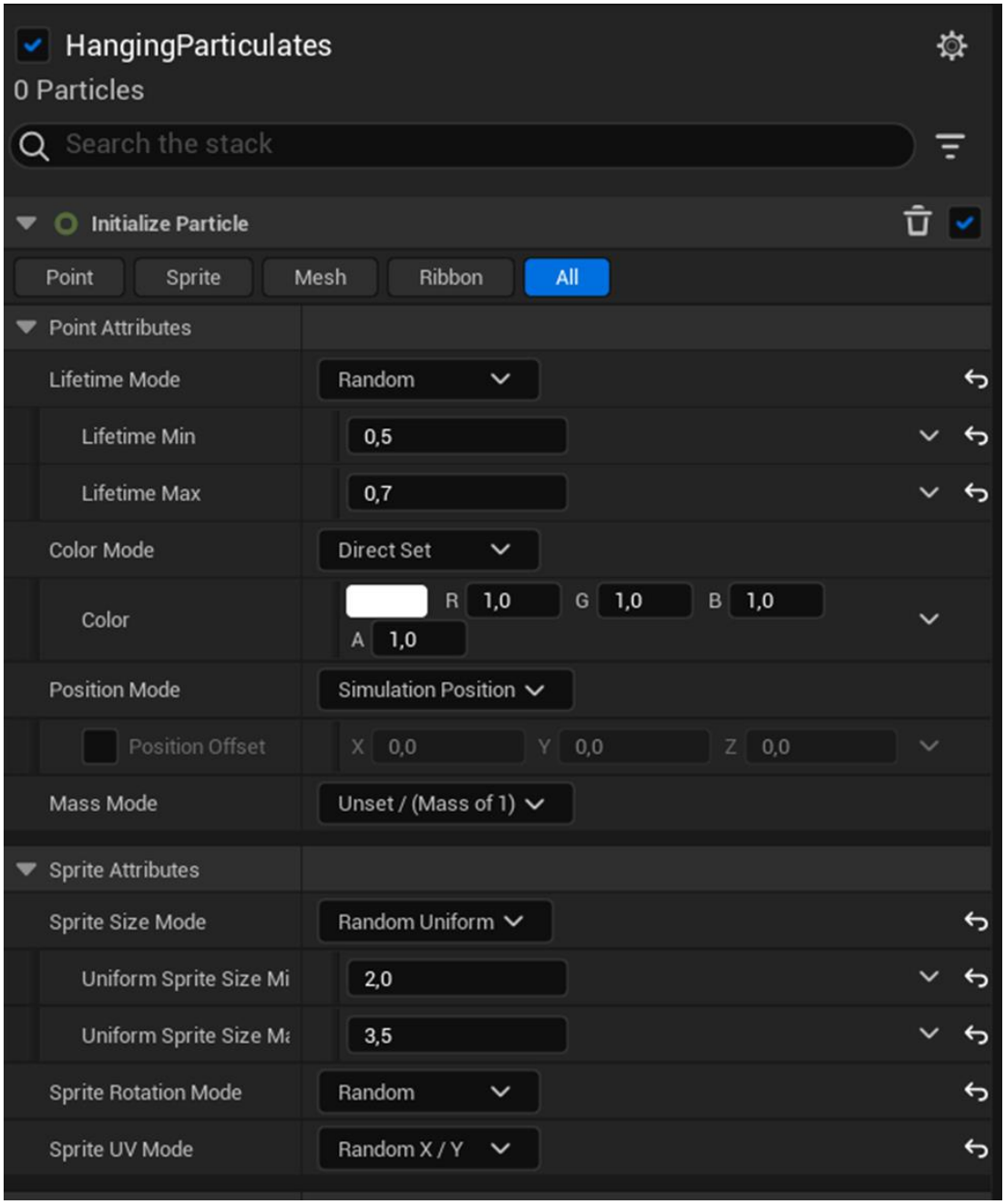


Рис. 3.39 Налаштування Initialize Particle у емітері Hanging_Particulates

Після усіх налаштувань, при активації NS_Rain, на сцені активується доволі реалістичний дощ. Наразі він постійно ллє із максимальною силою, що у подальшому буде відрегульовано за допомогою користувацької змінної User_SpawnRate таким чином, як потрібно для проекту.

Рис. 3.40 Вигляд реалізації дощу через систему часток Niagara System

3.1.5. Створення туману

Одним із найважливіших факторів правильної роботи тривимірного проекту є оптимізація. Так як цей проект відбувається на доволі великій сцені, відображення усіх деталей одночасно може спричинити велике навантаження на систему. Для того щоб знизити навантаження на систему, при цьому не нехтуючи естетикою візуальної складової проекту, найчастіше використовується ефект туману. Він особливо добре підійде для симуляції дощу, через те що вода випаровується при дощу, створюючи туманну димку.

Окрім того, туман також виконує часткову роль пост процесінгу, тобто надає необробленій, часто неприємно виглядаючій сцені більш натурального вигляду, маскуючи недоліки. Для прикладу, на рис. 3.41 відображено сцену проекту, яка наразі створюється, без будь-якого туману. Після повноцінного

налаштування туману, на рис. 3.43 можна побачити, як усього лиш додавання до сцени туману кардинально змінило її візуальне сприйняття.

Рис. 3.41 Вигляд сцени без туману

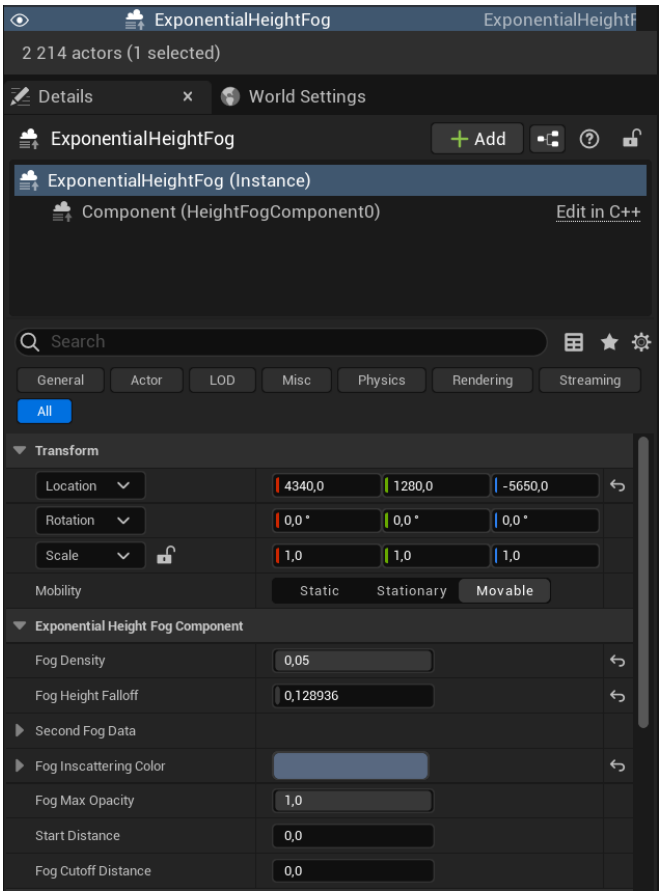


Рис. 3.42 Функція туману Exponential Height Fog

Для цілей даного проекту буде використаний клас ExponentialHeightFog, який окрім щільності туману також має функцію налаштування висоти та градієнту щільності в залежності від висоти, що дозволяє створити куди більш реалістичну симуляцію туману.

Тепер треба налаштувати базові значення туману, щільність туману виставляється на 0,05, коефіцієнт зменшення щільності із збільшенням висоти ставиться на 0,128936, колір відтінку туману виставляється на синювато-сірий, максимальний об’єм туману ставиться на 1,0.

Рис. 3.43 Вигляд сцени із туманом

3.1.6. Створення основного коду програми

Так, як усі основні елементи проекту готові, наступним кроком створюється код програми, який буде відповідати за функціонування таких функцій, як зміна погоди, роботу UI, та регулювання швидкості зміни дня та ночі. Так, як однією із найважливіших факторів створення довгограючого коду та функціоналу є «standalone» класи, більшість усього коду буде створено усередині класу Bp_Custom_SunSkyNew, який відповідає за функціонування зміни дня та ночі. Єдиним виключенням являється активація функцій при натисканні клавіш, бо за замовчуванням цей функціонал притаманний лише

класу персонажу або рівня. У цьому проекті функціонал натискання клавіш реалізований у класі рівня Level blueprint.

Першою функцією, яка буде створена у класі – це функція швидкості зміни дня та ночі, за функціонал якої відповідає змінна із класу Update Sun під назвою Amount_Of_Seconds_To_Add_Per_Tick. При активації цієї функції відбувається посилання на клас BP_CustomGameState, що являється створеним користувацьким станом гри, а після цього активується корегування значення змінної в залежності від натиснутої клавіші, відповідно клавіші 1-4 та змінні x1-x4.

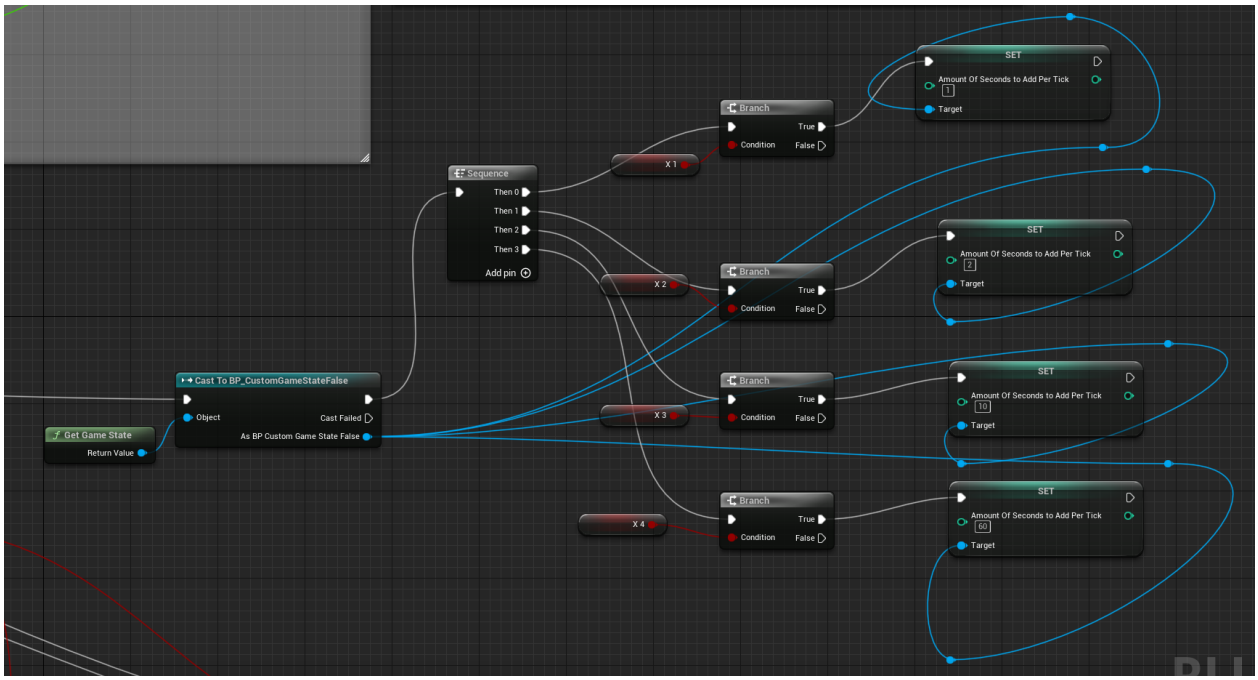


Рис. 3.44 Функція зміни швидкості зміни дня та ночі

Після цього, у класі рівня, налаштовуємо самі клавіши, тобто функції, які спрацьовують при натисканні потрібних клавіш, у даному випадку клавіші 1-4, які відповідають за швидкість зміни дня та ночі.

Для початку, основною функцією, яка буде підтримувати роботу цих натискань, є Event Tick. Event Tick надсилає сигнал до перевірку статусу активації функції кожен тік. Після цього, активується функція кожної із клавіш відповідно, яка посилає сигнал до функції Gate, яка в свою чергу активує наступні функції тільки за умови натискання клавіші. Після цього, робиться

посилання на змінні класу BP_Custom_SunSkyNew, які активуються при натисканні потрібної клавіші, та деактивуються через 0.2 секунди після деактивації, щоб запобігти неправильній роботі програмі, при якій можуть бути одночасно активовані декілька функцій, що змінюють значення змінної, що відповідає за зміну дня та ночі, Amount_Of_Seconds_To_Add_Per_Tick, що в результаті призведе до того, що функція зміни дня та ночі перестане працювати.

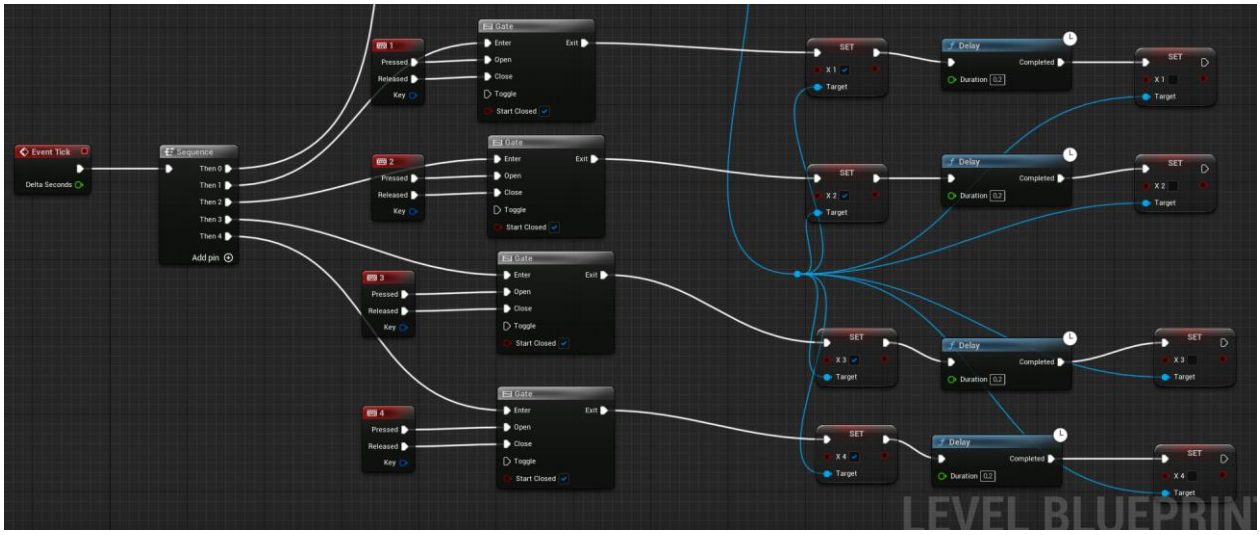


Рис. 3.45 Активація функцій дня та ночі при натисканні клавіш

Наступним кроком, створюється функція, яка відповідає за активацію дощу. Активація відбувається при натисканні кнопки F, що відображено у класі рівня.

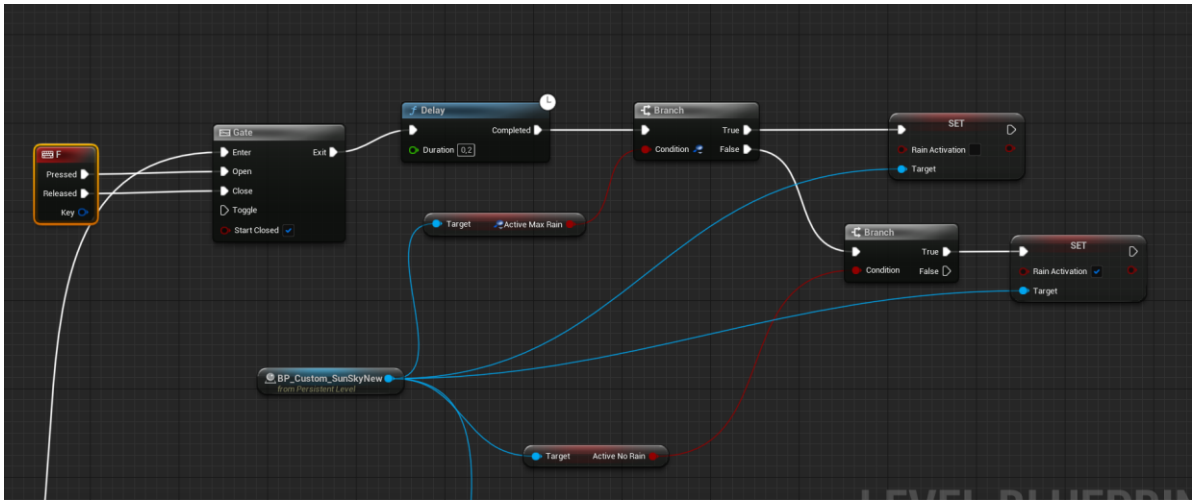


Рис. 3.46 Активація функції дощу при натисканні клавіши F

Після натискання клавіші, активується функція Gate, яка із затримкою у 0.2 секунди деактивує дощ при позитивному значенні змінної Active_Max_Rain, яка обраховується порівнянням поточного часу анімації дощу із 30 секундами, та змінної Rain_Activation, що відповідає за те, чи є активним дощ. Така умова існує, щоб зворотній процес дощу не активувася при натисненні кнопки до того моменту, доки не пройде 30 секунд (або попередньо задане значення).

Сама активація дощу також перевіряє, щоб дощ не йшов/повністю закінчився, щоб не зламати процес.

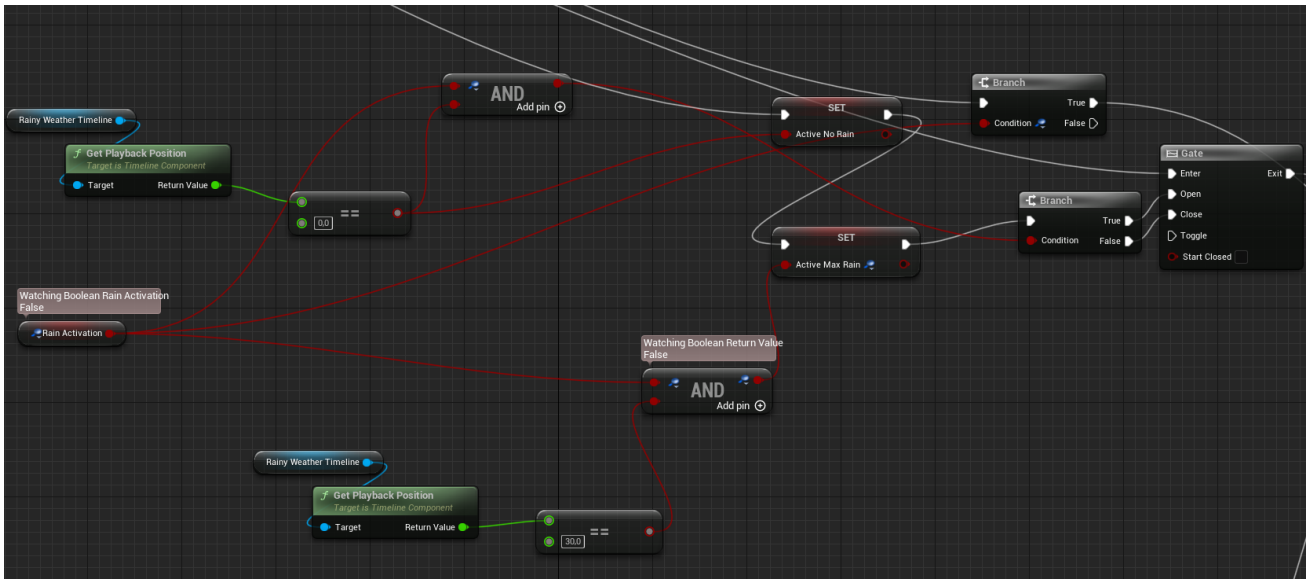


Рис. 3.47 Активація функції дощу при натисканні клавіши F

Після того, як кнопка натиснута, ще раз перевіряються умови, вже у класі Bp_Custom_SunSkyNew, а саме умови, які забезпечують факт того, що дощ вже скінчився, або не активний, та що дощ не йде на максимальній силі. Після того, як вищезгадані умови, та сама змінна активації дощу є позитивною, активується наступний код, який регулює дощ та інші погодні умови, створені у попередніх розділах.

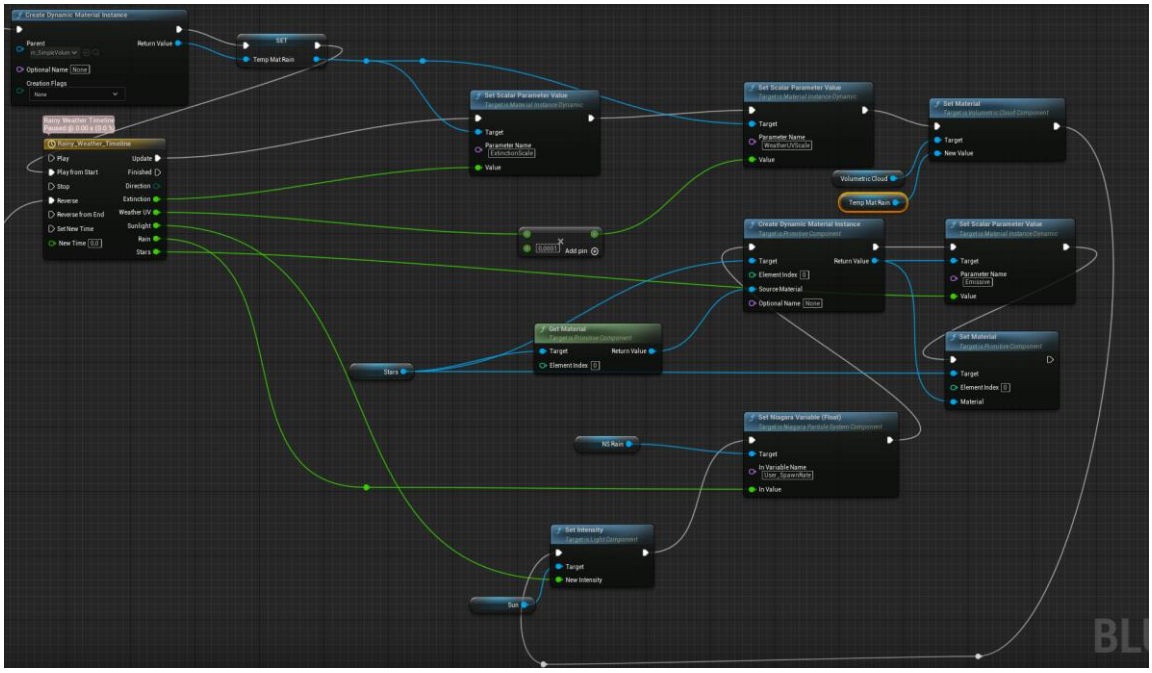


Рис. 3.48 Основна функція активації дощу

Для початку, після активації дощу створюється функція типу Timeline, яка буде регулювати змінення змінних, що відповідають за зміну погоди, відносно до часу. Ця функція наразі має два входи активації, Play from Start та Reverse. Play from start активує анімацію для старту дощу, тоді як Reverse запускає цю ж анімацію, тільки із кінця в початок, що створює ефект припинення дощу та покращення погоди.

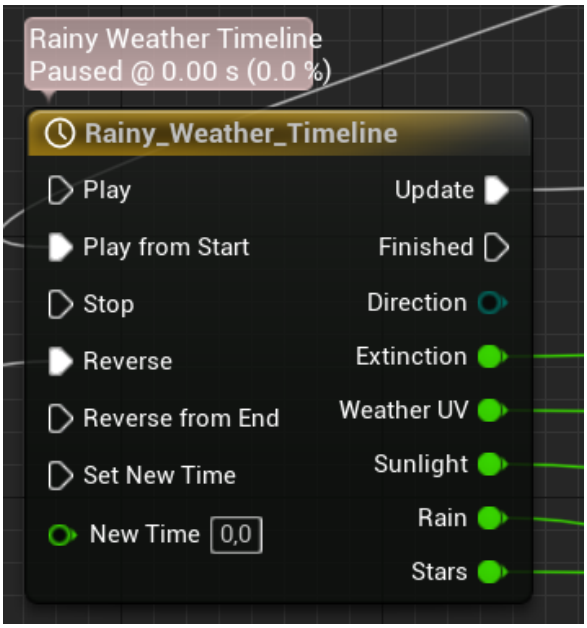


Рис. 3.49 Функція Timeline

Як видно на Рис.3.49, таймлайн Rainy_Weather_Timeline змінює значення п’яти float змінних. На рівні рушію, подібні зміни забезпечуються змінами у інстансах матералів цих хмар.

Але звичайні інстанси зазвичай не змінюють у реальному часі, тому що це пермаментно змінює їх значення, що відображається доволі негативно на кінцевому резальтутаті, адже при будь-якій помилці у програми доводиться налаштовувати усі інстанси матеріалів спочатку.

Для того, щоб уникнути таких проблем, використовується функція створення динамічних інстансів матеріалів, які створюються тільки під час роботи програми, і видаляються із пам’яті після завершення її роботи. Ця функція називається Create_Dynamic_Material_Instance.

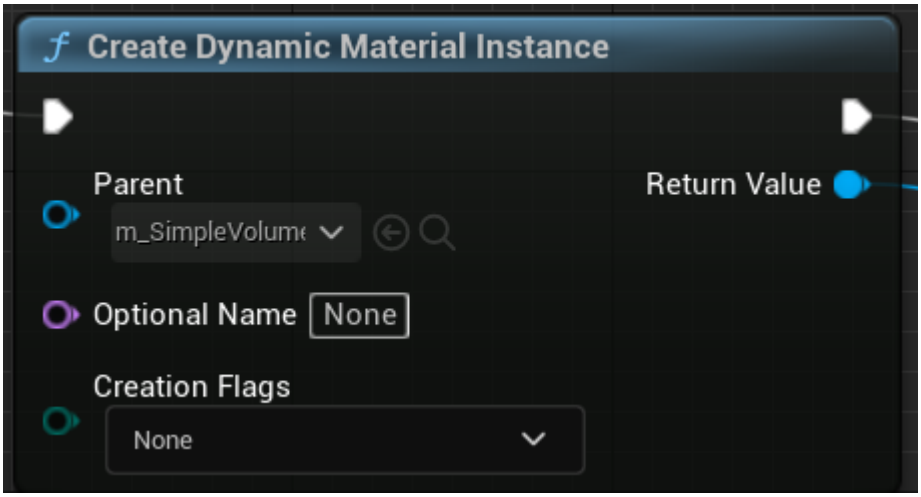


Рис. 3.50 Функція Create Dynamic Material Instance

Після того, як динамічний інстанс матеріалу створений, функція Set_Scalar_Parameter_Value дозволяє змінювати індивідуально обраний параметер матеріалу.

Є також аналоги цієї функції, так як Set_Vector_Parameter, що дозволяє оперувати не тільки звичайними змінними, а й іншими типами, такі як вектори. Функція вектору, для прикладу, особливо корисна для зміни кольору у реальному часі, адже колір налаштовується як вектор.

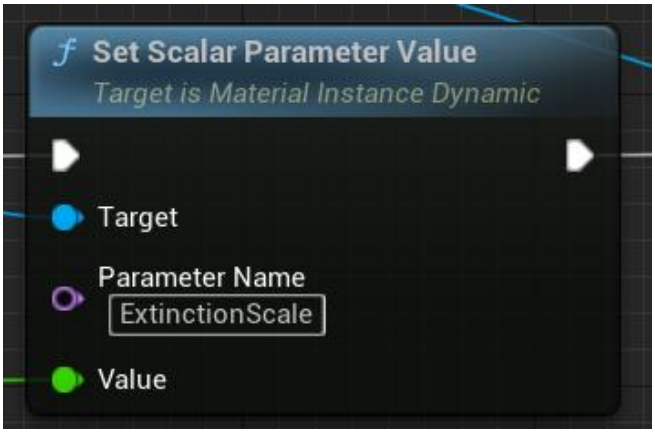


Рис. 3.51 Функція Set Scalar Parameter Value

Повертаючись до таймлайну Rainy_Weather_Timeline, перші дві змінні, Extinction та WeatherUV, відповідають за зміну форми та області покриття об’ємних хмар. Так, як дощ завжди супроводжується доволі хмарним небом, кінцеві значення цих змінних відображають саме таке небо.

Протягом 30 секунд, значення Extinction хмари збільшується від 0,001 до 0,5, а значення WeatherUV – зменшується від 3,0 до 1,5.

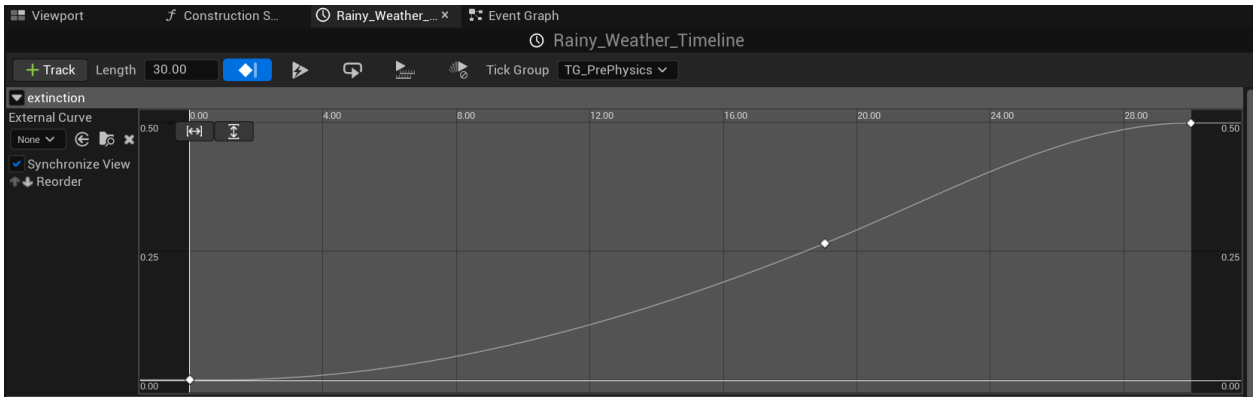


Рис. 3.52 Таймлайн змінної Extinction

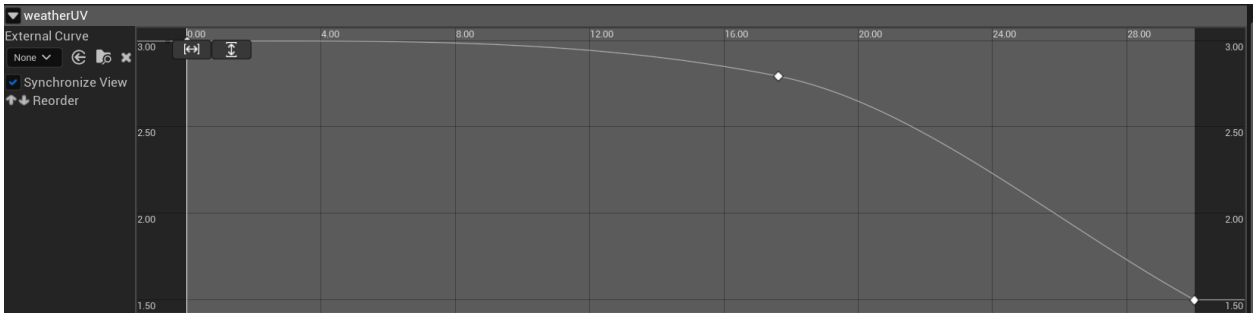


Рис. 3.53 Таймлайн змінної WeatherUV

Наступна змінна, Sunlight, змінює значення інтенсивності сяйва сонця у lux, що забезпечує меншу освітленість, яка корелює із щільністю хмар на небі. Зміна інтенсивності, на відміну від хмар, робиться напряму, посилаючись на об’єкт сонця у сцені, та його змінну, що відповідає за інтенсивність, так як сонце не потребує матеріалу взагалі. Таймлайн Sunlight зменшує значення Intensity сонця від 3,16 lux до 0,5 lux протягом 30 секунд.

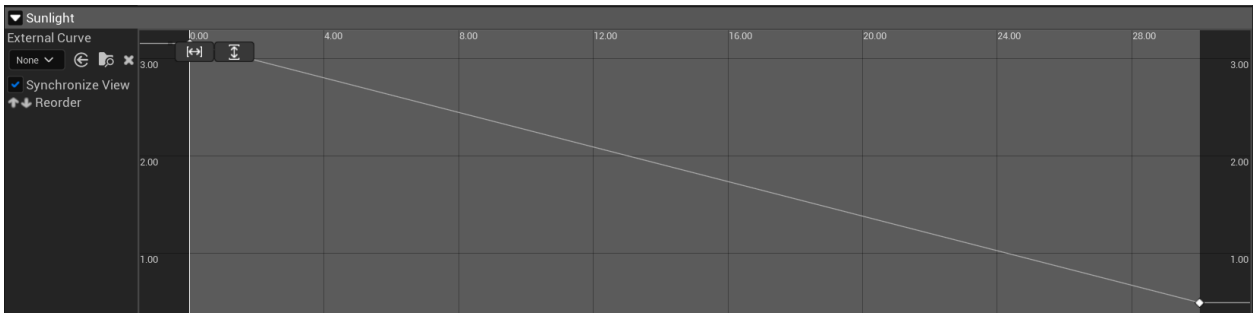


Рис. 3.54 Таймлайн змінної Sunlight

Після цього, змінна Rain регулює інтенсивність дощу. Ця змінна напряму посилається на користувацьку змінну User_SpawnRate у класі NS_Rain, що є класом системи часток Niagara. Ця змінна напряму корегує, наскільки часто створюються частки крапель, які падають із неба. Із збільшенням цього ефекта, створюється ілюзія збільшення інтенсивності дощу. Починаючи із восьмої секунди, змінна User_SpawnRate збільшується від 0,0 до 12000,0 на тридцятій секунді. Початок із восьмої секунди зроблений із розрахунку на те, що хмари повинні достатньо згуститися, щоб почав крапати дощ.

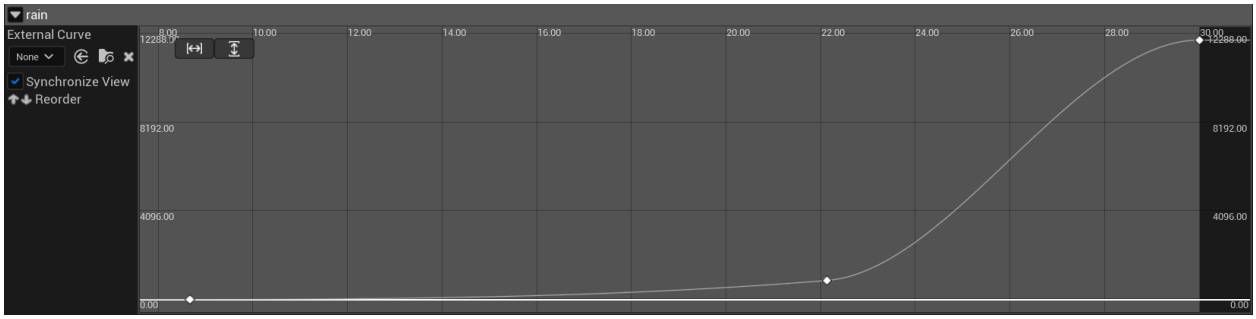


Рис. 3.54 Таймлайн змінної Rain

Остання змінна, яку потрібно змінити, це змінна сфери зірок - Stars. Проблем в тому, що за замовчуванням зірки мають властивість emissive, яка дає їм змогу світитися, але через те, що об’ємні хмари є напівпрозорими, зірки просвічували крізь них, що виглядало доволі антикліматично.

Рис. 3.55 Зірки просвічують крізь хмари

Тому було створено таймлайн Stars, який знижує інтенсивність сяйва зірок за допомогою зменшення коефіцієнту emissive матеріалу, доки зовсім не виключає його, при повній хмарності. Ця зміна, як і у випадку із хмарою, також досягається створенням динамічного інстансу матеріалу, та зміною скалярного параметра, що відповідає за emissive. Змінна Emissive зменшується від 1,0 до 0,0 протягом 30 секунд

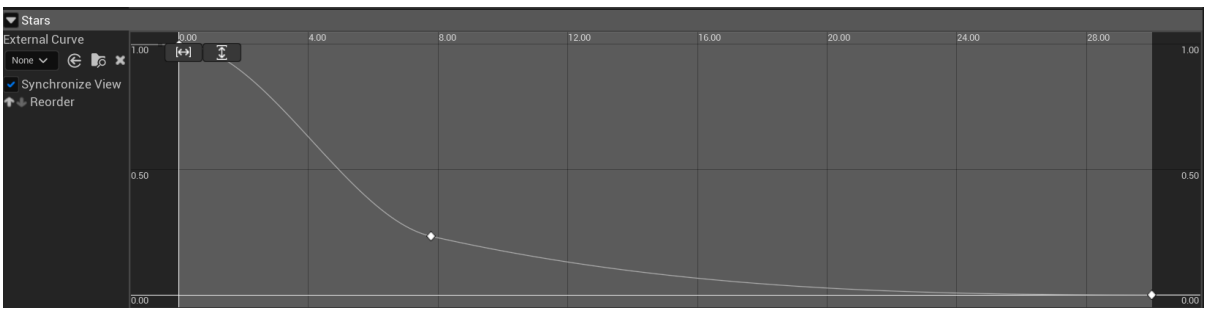


Рис. 3.56 Таймлайн змінної Stars

Після того, як усі таймлайни налаштовані, залишається останній крок – створення активації зворотнього процесу налагодження погоди. За це відповідає функція, яка перевіряє умови, зворотні до активації, тобто позитивність факту того, що пройшло 30 секунд дощу, та дощ активний, та спрацьовує після повторного натиснення кнопки F, після чого запускається Reverse вхід таймлайн функції Rainy_Weather_Timeline, що у свою чергу запускає усі вище описані таймлайни із кінця в початок.

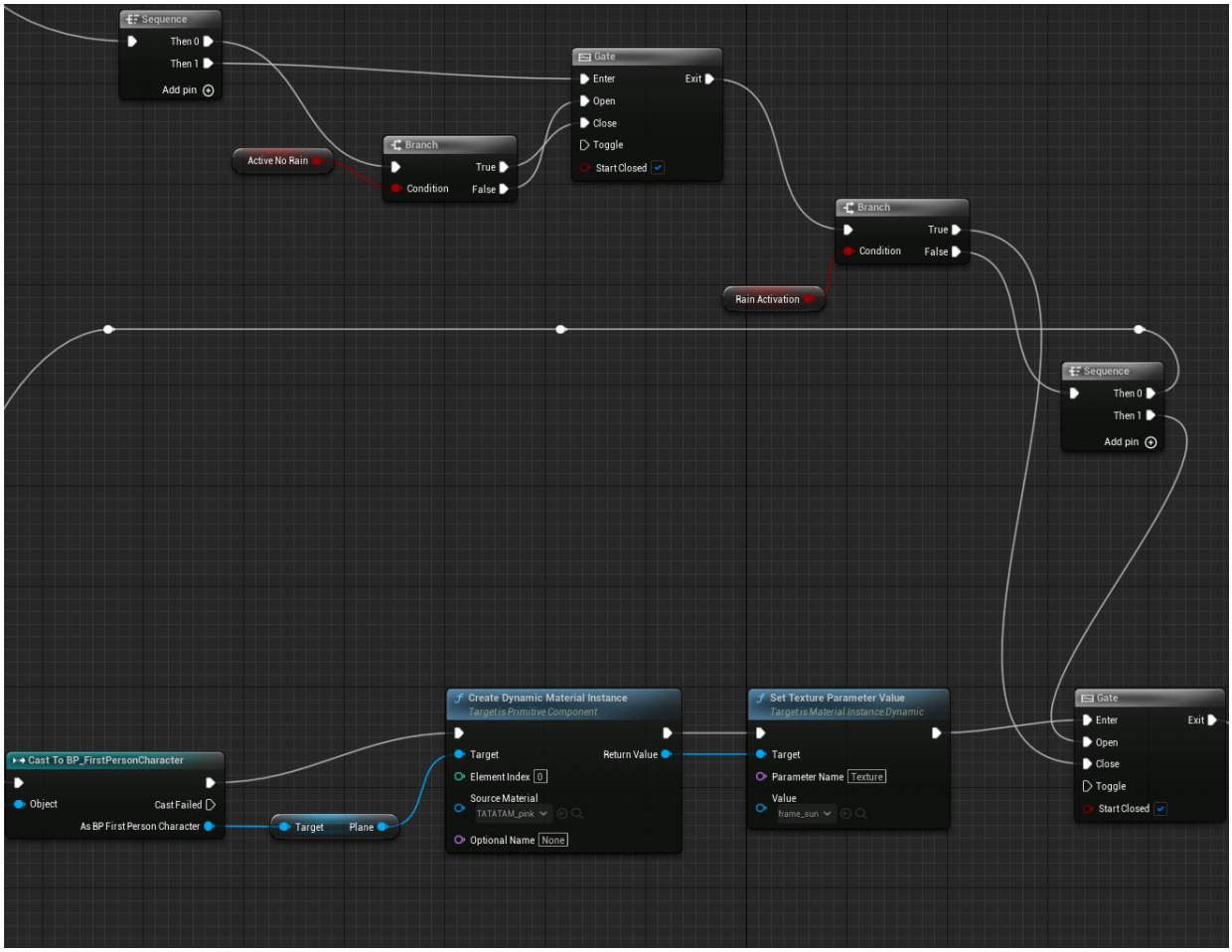


Рис. 3.57 Таймлайн змінної Stars

Кінцевий результат функції дощу можна побачити на Рис. 3.58-3.61. Завдяки єдиному таймлайну, який регулює усі процеси одночасно, зміна погоди відбувається гармонічно, і відображає усі необхідні зміни відповідно до індивідуального таймлайну, налаштованого для неї.

Рис. 3.58 Соняна погода

Рис. 3.59 Збільшення хмарності

Рис. 3.60 Початок дощу

Рис. 3.61 Сильний дощ

3.1.7. Пост процесінг

Наступний необхідний крок – створення об’єкту пост-процесінгу. Пост процесінг відповідає за налаштування «камери», якою користувач дивиться на сцену, вона включає до себе фільтри, освітлення, шейдери та інші аспекти, які роблять сцену більш привабливою в залежності від обраного сценарія.

Для початку, до сцени додається елемент PostProcessVolume.

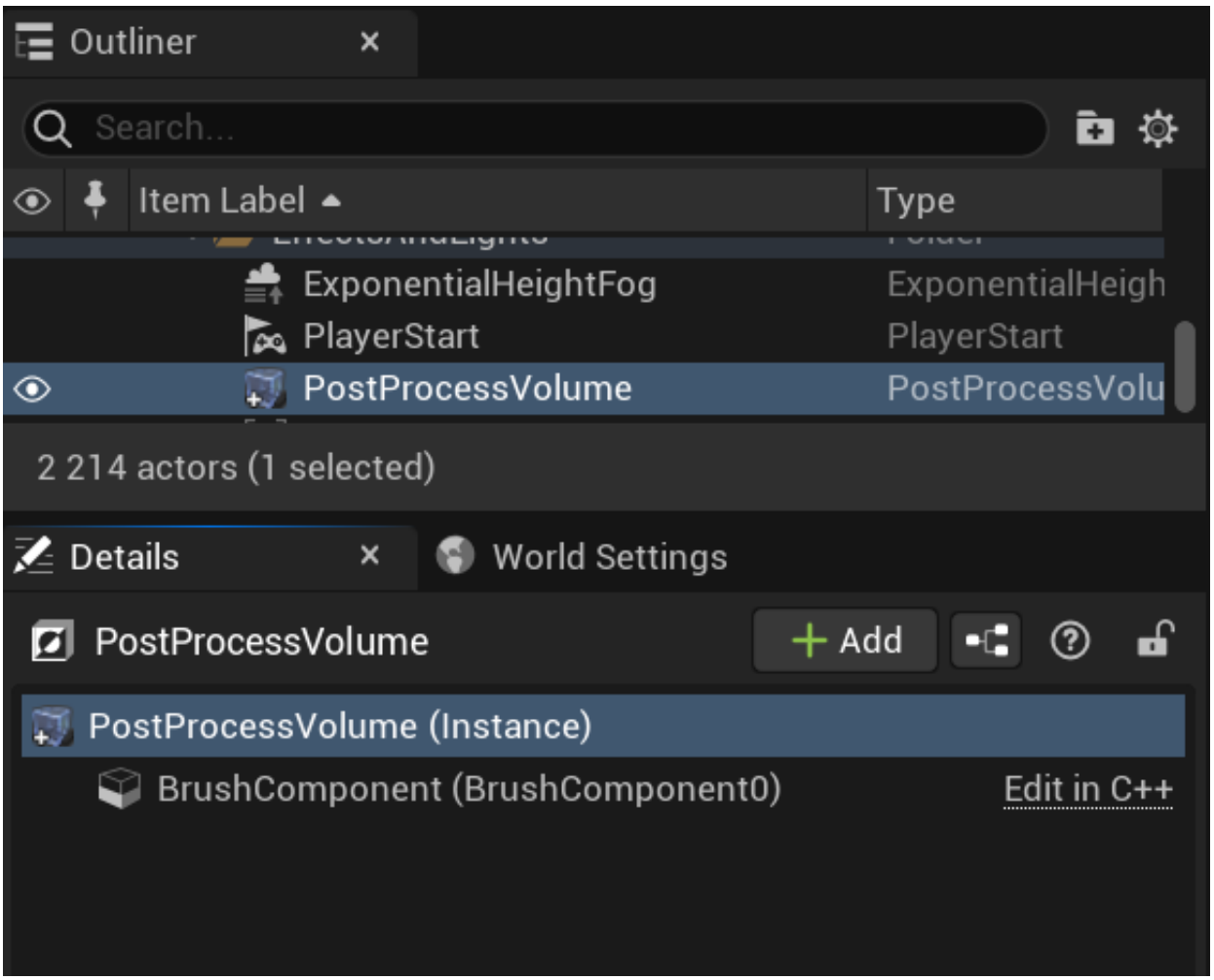


Рис. 3.62 PostProcessVolume

За замовчуванням, елемент Пост Процесінгу покриває лише обрану область, обозначену прямокутною зоною. Так як у даному випадку налаштовується уся сцена, включається опція Infinite Extent (Unbound) у Post Process Volume Settings. Тепер пост процесінг активний на всій сцені, незалежно від місця.

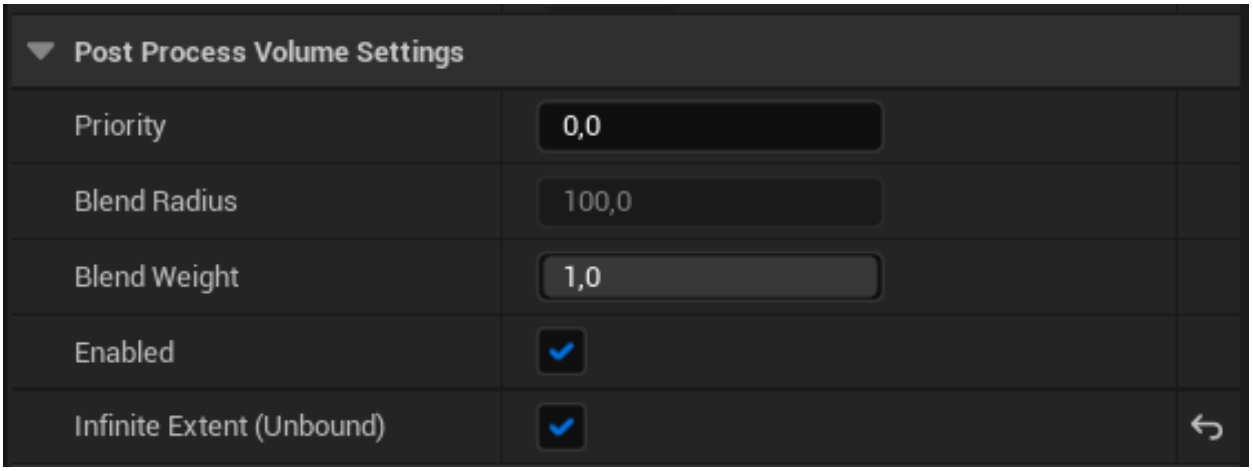


Рис. 3.63 Налаштування PostProcessVolume

Після цього також включається ігнорування колізії та активується створення об’єкта пост-процесінгу на сцені при будь-яких умовах. Це дозволить пост процесінгу працювати завжди однаково не дивлячись на те, що на сцені відбувається, за виключенням освітлення та тіней.

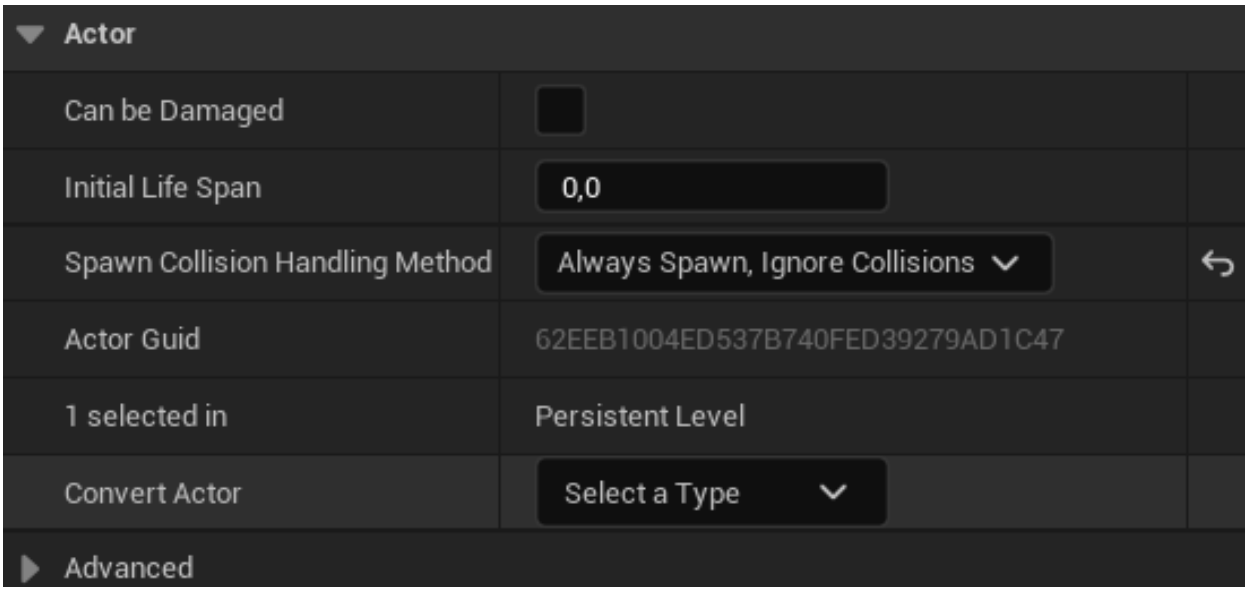


Рис. 3.64 Налаштування Actor

Після цього, налаштовуються температура та контрастність освітлення сцени, для створення більш живого та стилізованого освітлення.

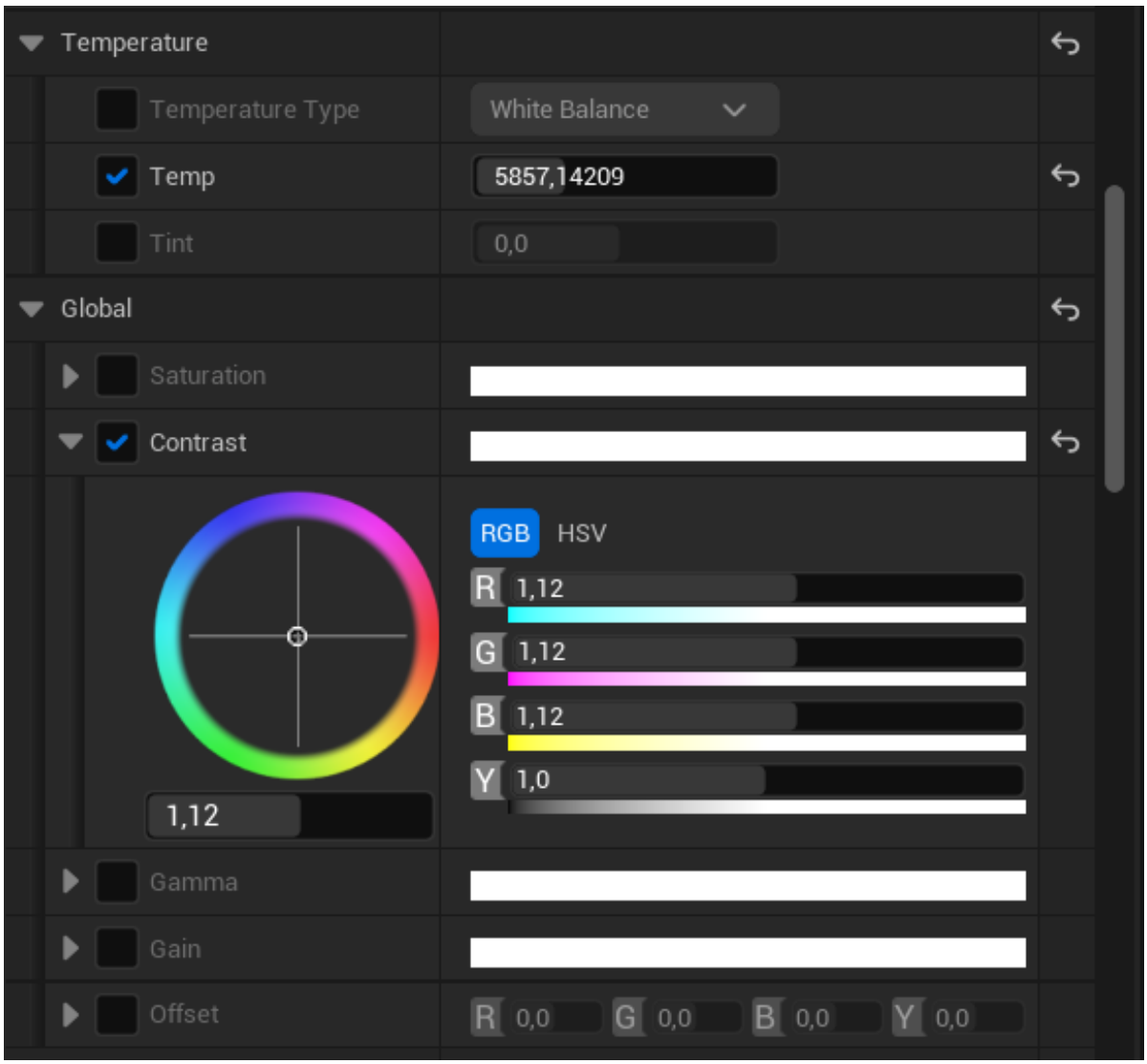


Рис. 3.65 Налаштування Температури та Контрастності

Нарешті, останній параметр – у категорії «Film», а саме Slope, який регулює нахиленність зображення, яке створює більш реалістичний ефект.

Рис. 3.66 Налаштування «фільмовості»

Після усіх налаштувань, на Рис. 3.66-3.67 можна побачити кінцевий результат активації пост-процесінгу

Рис. 3.67 Ефект (праворуч) пост-процесінгу удень

Рис. 3.68 Ефект (праворуч) пост-процесінгу уночі

3.2. Створення користувацького інтерфейсу

Так, як функціонал програми готовий, наступним кроком необхідно створити користувацький інтерфейс, який дозволить бачити візуальне відображення активації функції погоди, та швидкості зміни циклу дня та ночі. Для початку, треба створити матеріал, який буде виповнювати роль інтерфейсу.

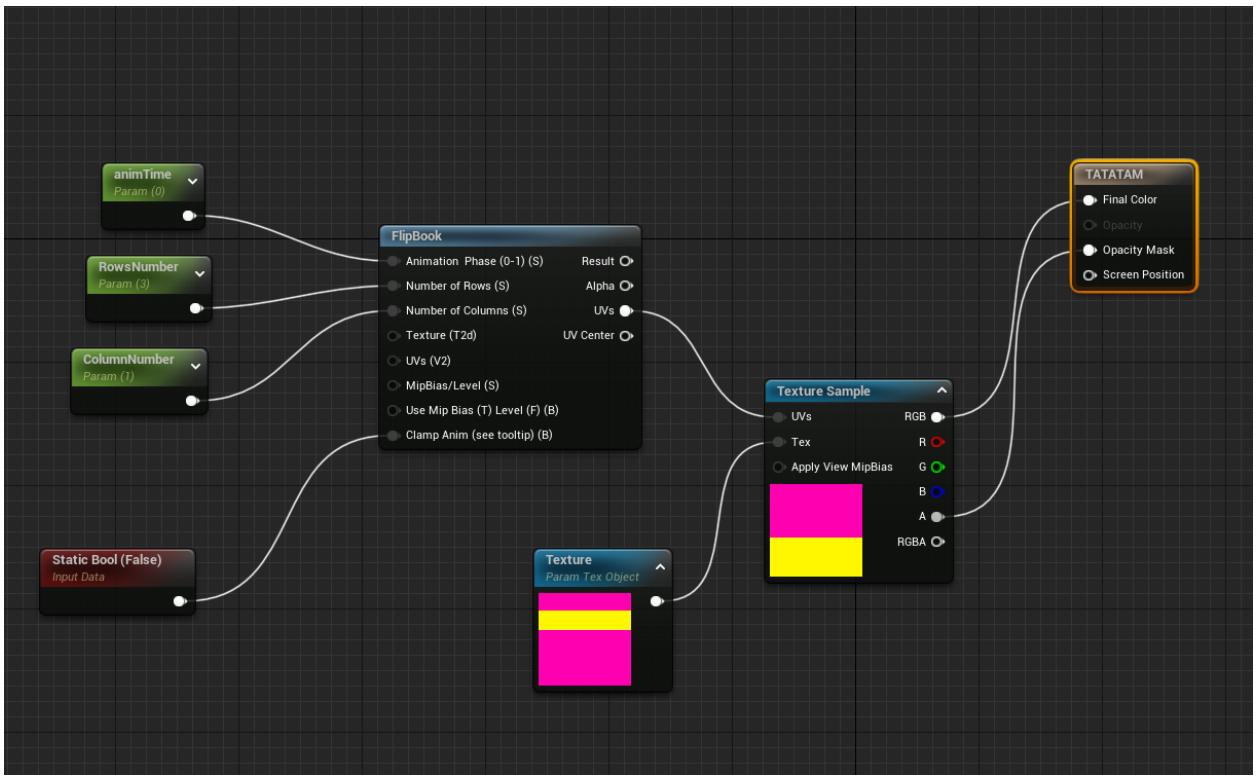


Рис. 3.69 Матеріал інтерфейсу

Цей матеріал використовує функцію FlipBook, щоб створити анімацію завдяки перемиканням текстур окремих кадрів. Таких чином, створивши потрібні кадри, можна використовувати їх як функціональні візуальні об’єкти, які змінюються відносно до активації різних клавіш та функцій.

Наступним кроком, треба намалювати сам інтерфейс. Для цього використовується Photoshop, через те, що в ньому можна доволі гнучко налаштовувати слої картинки, що дозволить створити інтерактивний інтерфейс без особливих проблем.

Для початку, треба взяти картинку із прикладом сцени, щоб можна було візуально представляти, де повинен бути користувацький інтерфейс.

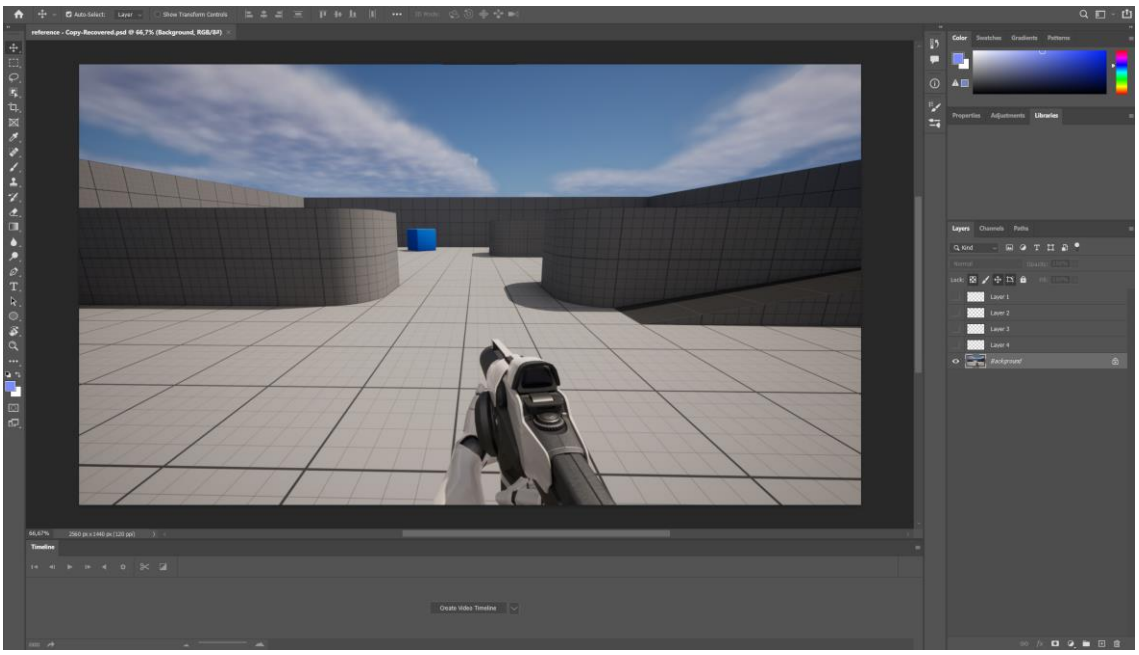


Рис. 3.70 Шаблон сцени у Photoshop

Після цього, малюються два інтерактивні елементи меню – переключення погоди між сонцем та дощем, та змінення швидкості зміни циклу дня та ночі.

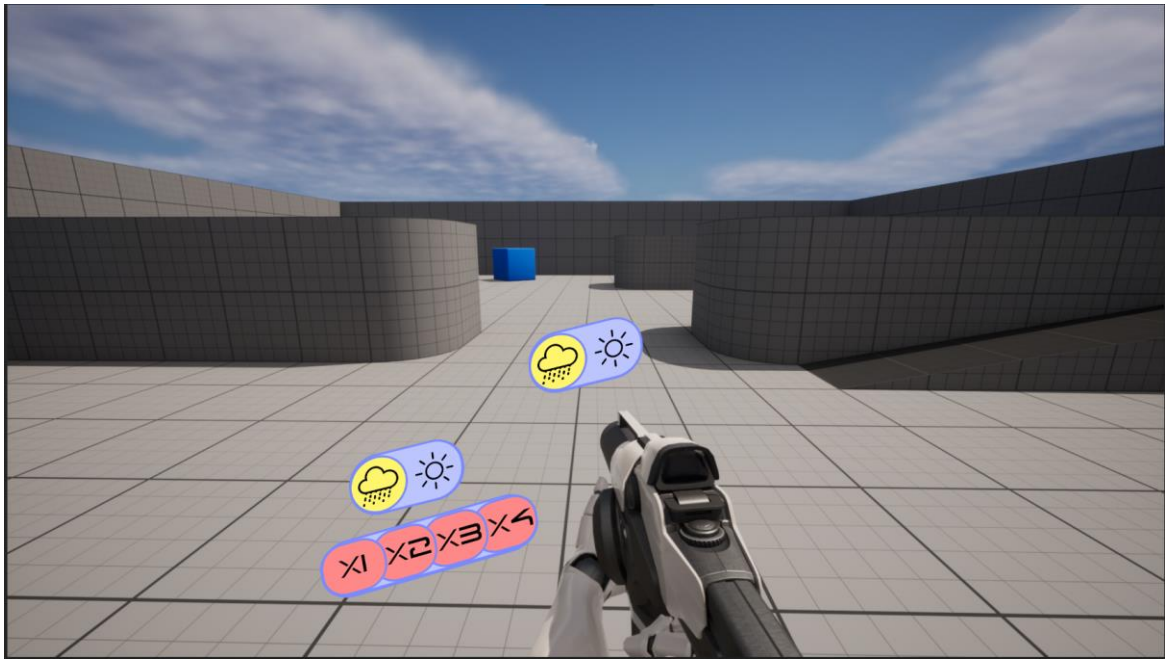


Рис. 3.71 Готові елементи користувацького інтерфейсу

Усі елементи розбиті на групи слоїв, які представляють кожен із станів інтерактивних об’єктів.

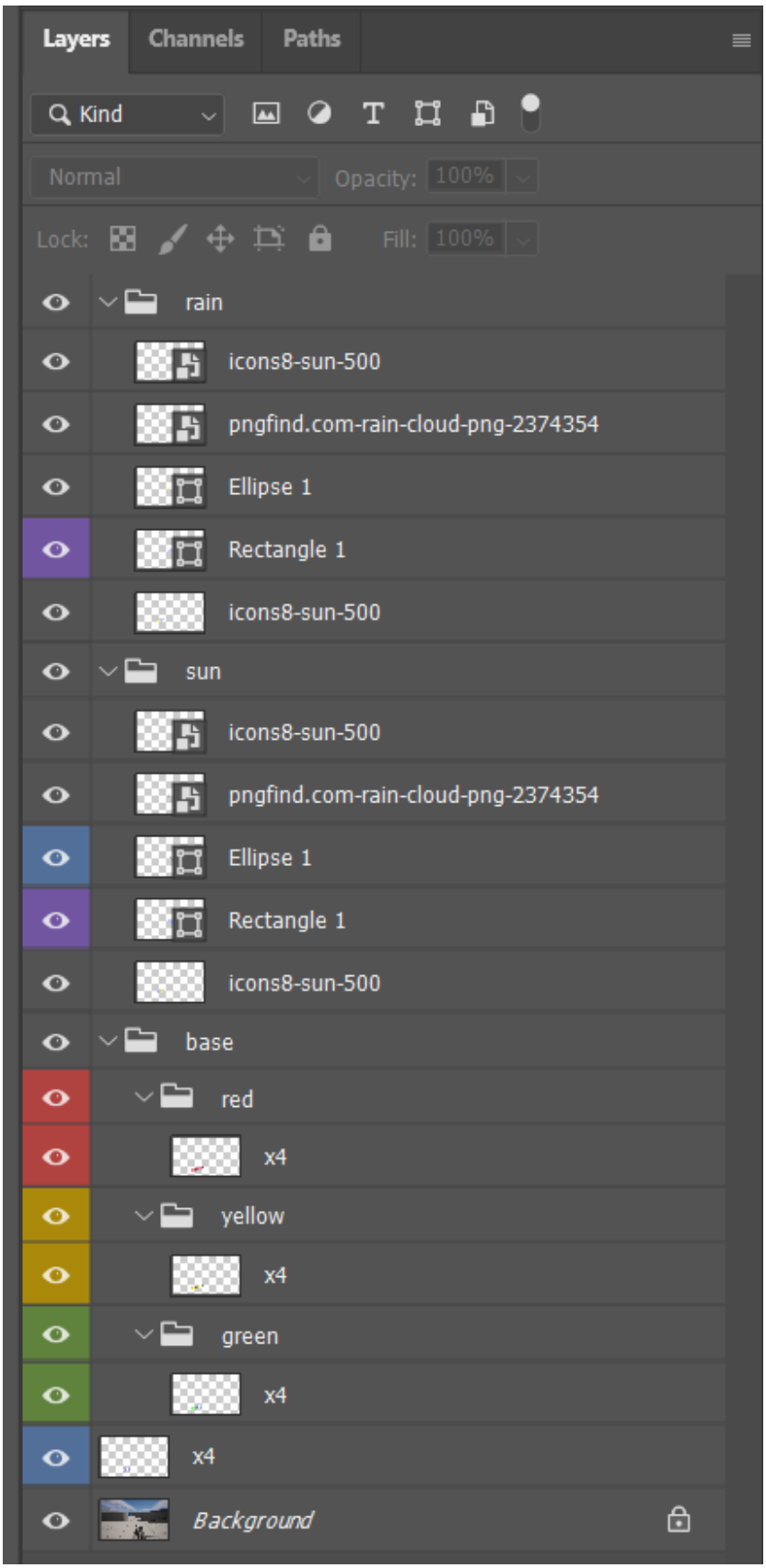


Рис. 3.72 Групи слоїв елементів у Photoshop

Після цього, кожна із груп слоїв зливається в один слой, який після цього експортується як PNG файл. Цей файл у подальшому буде використаний як текстура у Unreal Engine.

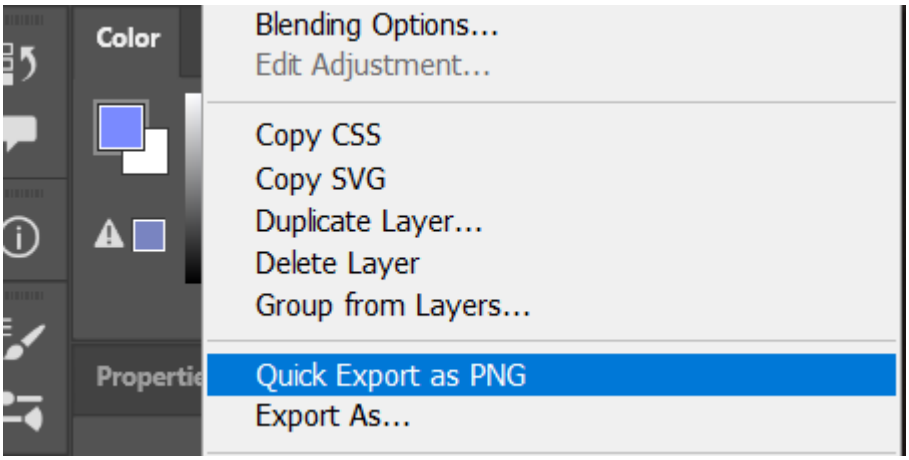


Рис. 3.73 Експортування слоя у Photoshop

Після того, як усі картинки інтерфейсу готові, вони імпортуються до Unreal Engine.

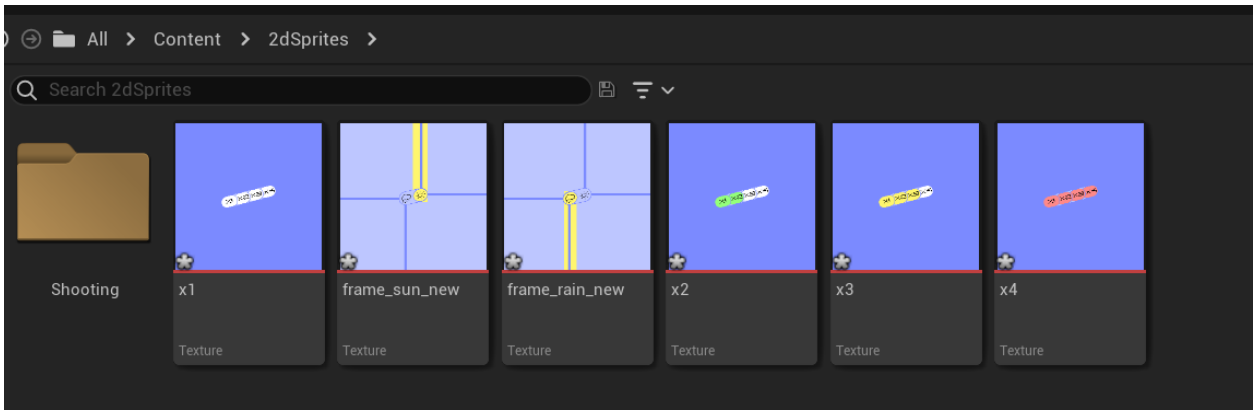


Рис. 3.74 Імпортовані картинки інтерфейсу

Однак одразу після імпорту вони непрозорі, тому не можуть бути використані. При спробі використання увесь екран буде покритий фіолетовим фоном. Для того, щоб зробити ці текстури пригідними до використання, треба застосувати до них спеціальні опції спрайтів, а саме Apply Paper2D Texture Settings.

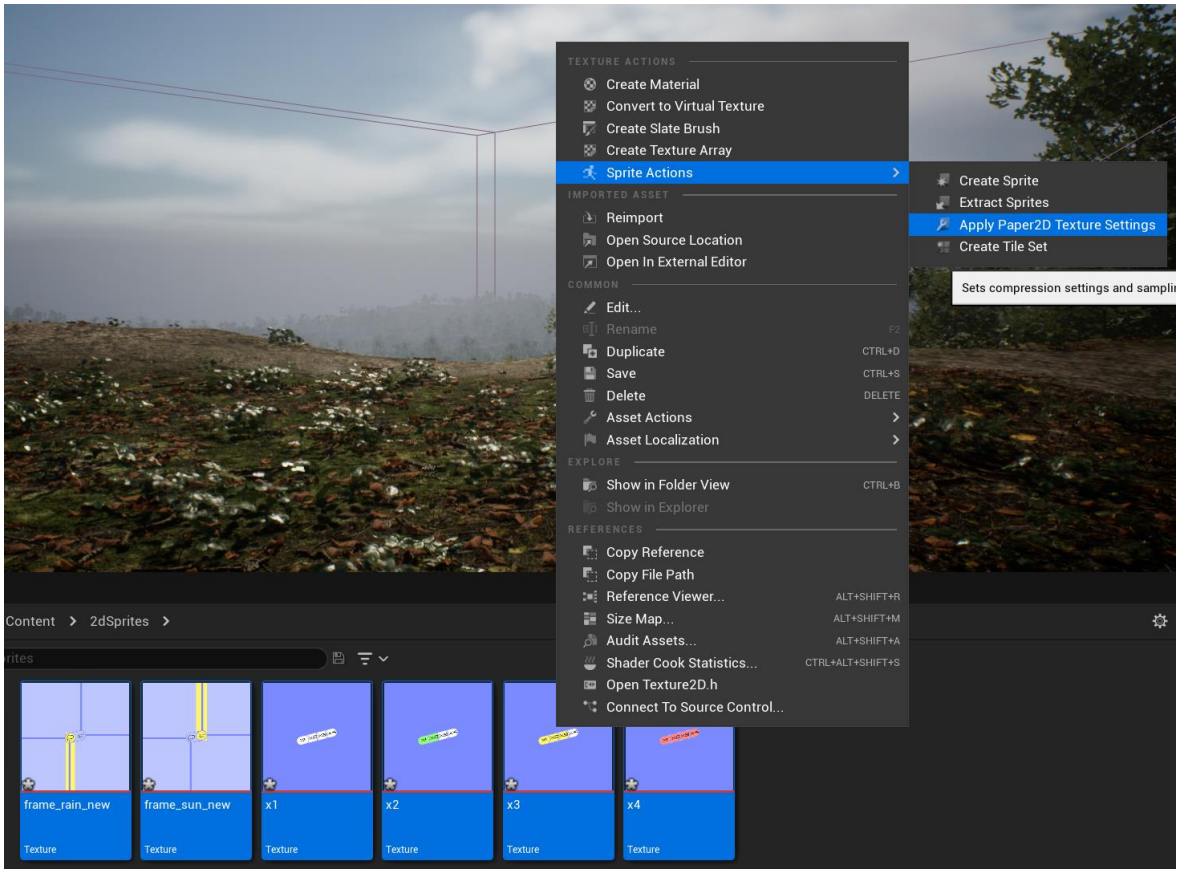


Рис. 3.75 Застосування опції Apply Paper2D Texture Settings

Після обробки, картинки тепер мають прозорий задній фон, як і потрібно для інтерфейсу.

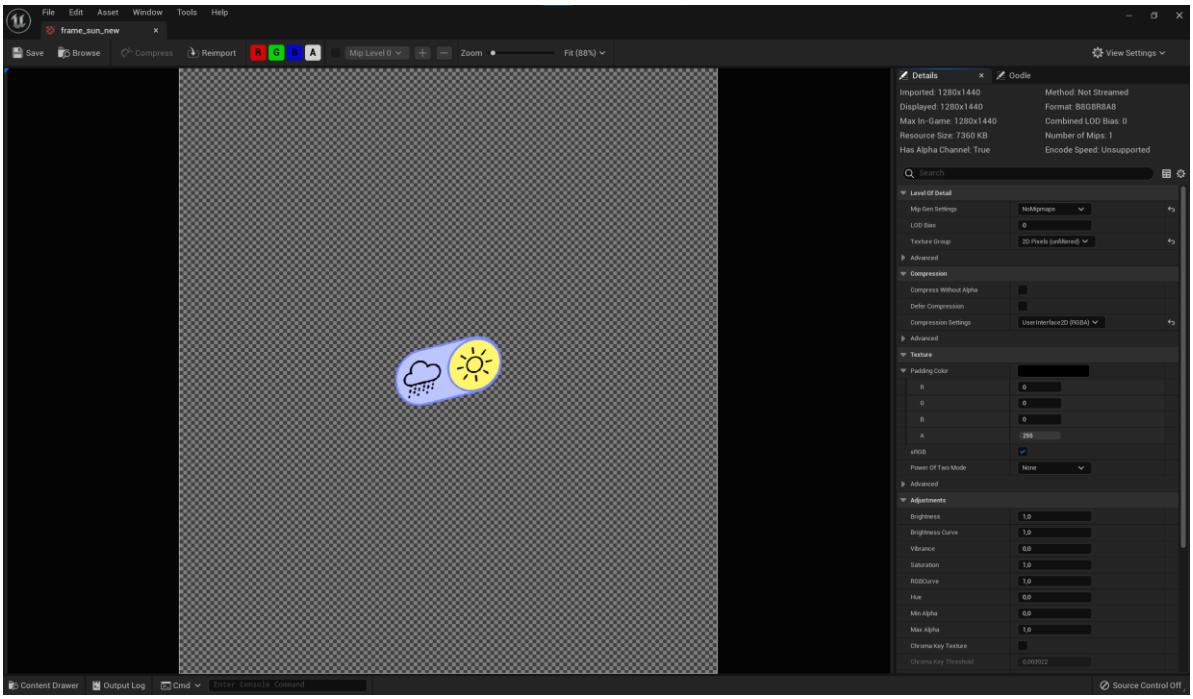


Рис. 3.76 Оброблена картинка користувацького інтерфейсу

Після цього, треба налаштувати відображення цього інтерфейсу у гравця. Для цього, у класі BP_FirstPersonCharacter, який відповідає за переміщення гравця на сцені, додати до моделі гравця два об’єкта Plane, щоб вони покривали простір перед камерою.

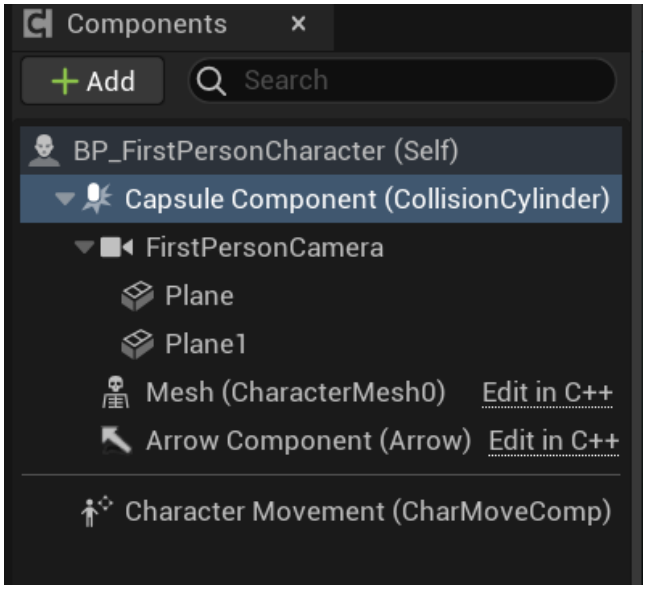


Рис. 3.77 Об’єкти Plane, прив’язані до камери у об’єкту класу BP_FirstPersonCharacter

Після цього, створюються два інстанси вище створеного матеріалу, по одному для кожного створеного елементу інтерфейсу. Відповідно до елементу інтерфейсу, виставляється потрібну текстуру у інстансі, а саме опції інтерфейсу за замовчуванням – x1 та frame_sun_new.

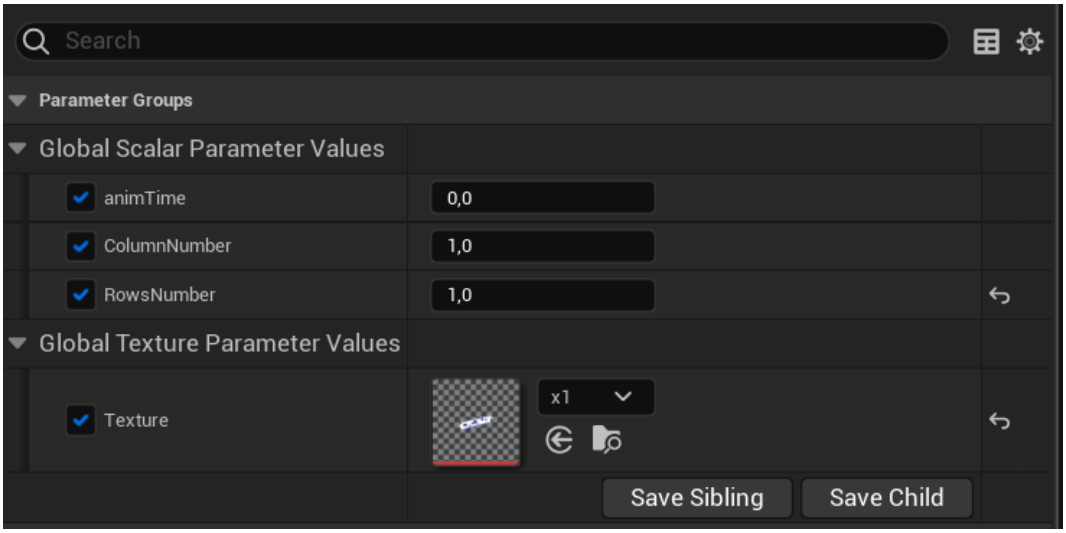


Рис. 3.78 Параметри інстансу матеріалу інтерфейсу

До вище створених об’єктів застосовуються відповідні інстанси матеріалу. Текстури цих матеріалів прозорі, за виключенням самих елементів інтерфейсу, що створює потрібний ефект. Після того, як матеріали застосовані, потрібно мануально налаштувати знаходження цих елементів інтерфейсу перед камерою.

Рис. 3.79 Об’єкти Plane, прив’язані до камери у об’єкту класу BP_FirstPersonCharacter

Із цим, користувацький інтерфейс готовий, наступний крок – зробити його функціональним. Перш за все, треба створити дві функції перемикання індикатора погоди. Обидві ці функції будуть створені у класі BP_Custom_SunSkyNew, де вони будуть спрацьовувати одночасно із функціями, які запускають основний та реверсивний процес дощу, та активуються при натисканні кнопки F. Ці функції працюють аналогічно до функції модифікації хмар та зірок, через створення динамічного інстансу

матеріалу, та змінення його параметру, у даному випадку – текстури. Значення текстури перемикається між frame_run_new та frame_sun_new.

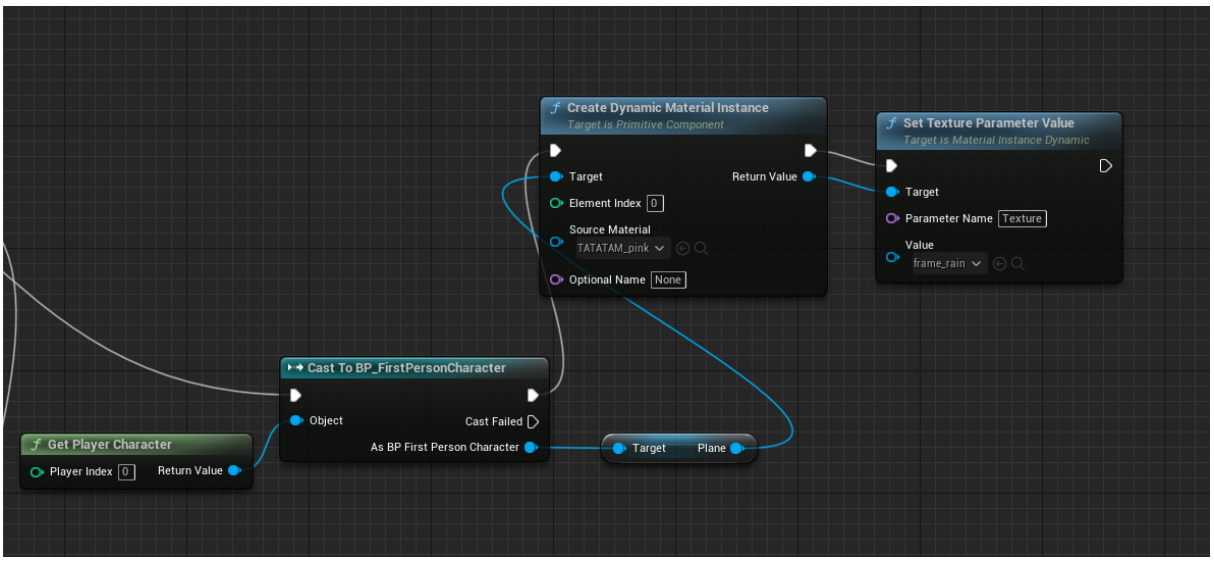


Рис. 3.80 Функія відображення активації дощу

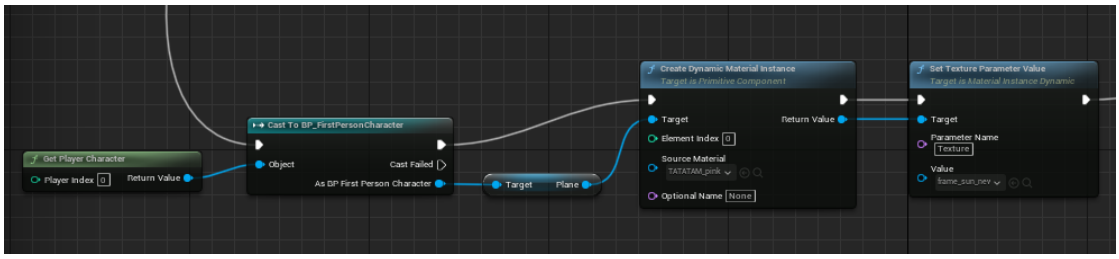


Рис. 3.81 Функція відображення деактивації дощу

Рис. 3.82 Відображення роботи інтерфейсу погоди

Готовий результат роботи інтерфейсу можна побачити на Рис. 3.80.

Аналогічно до цього, створюються функції перемикання швидкості зміни циклу дня та ночі. До кожної функції зміни швидкості приєднана активація потрібної текстури, що відображає поточний стан, створюючи потрібний динамічний інстанс матеріалу.

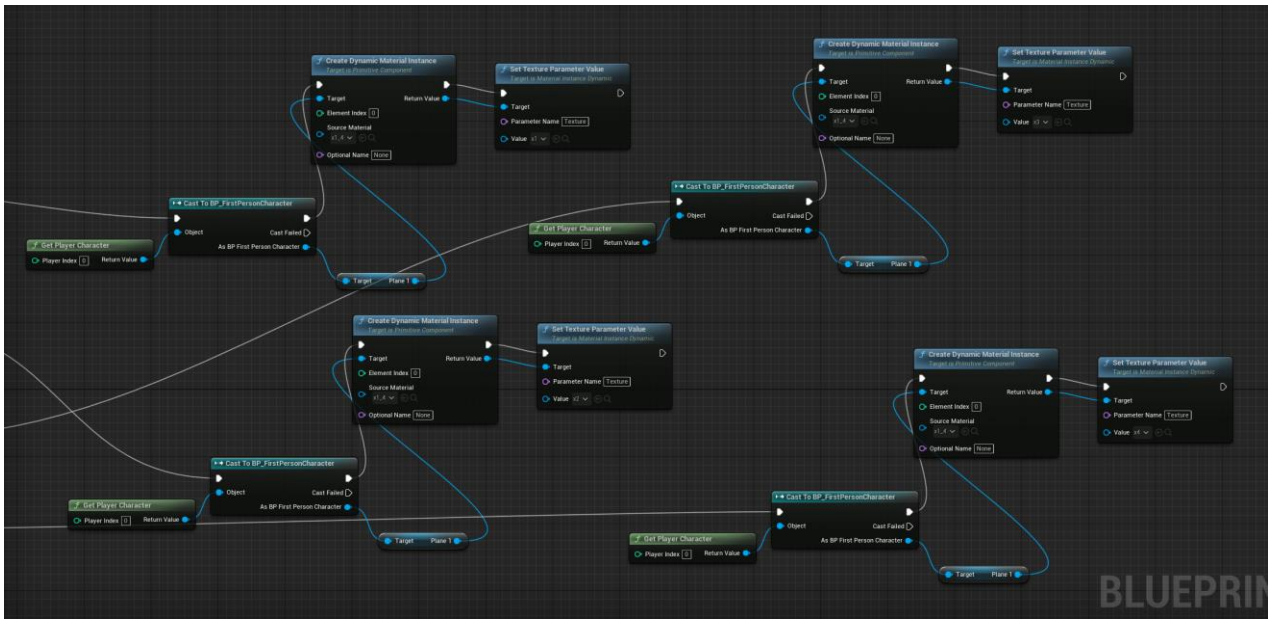


Рис. 3.83 Функція зміни відображення швидкості зміни циклу дня та ночі

Готовий результат роботи інтерфейсу можна побачити на Рис.3.82.

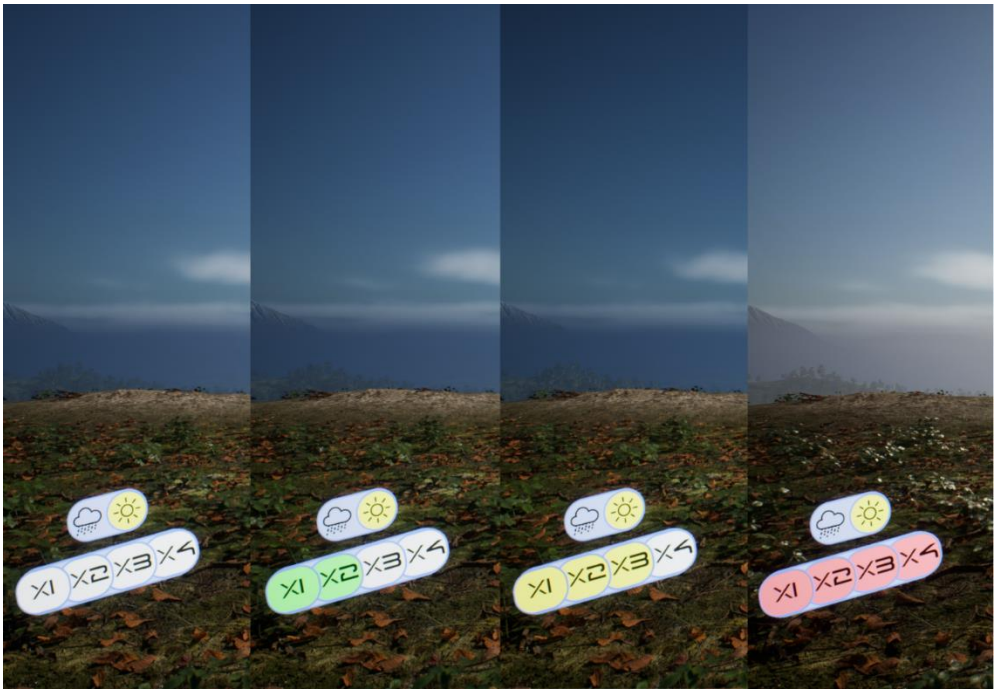


Рис. 3.84 Відображення роботи інтерфейсу зміни швидкості

3.3. Оптимізація

3.3.1. Nanite

Тоді як візуально і функціонально програма виглядає і працює згідно із цілями проекту, швидкодія не є ідеальною. Тому наступний крок – оптимізація тих аспектів, які можливо зробити більш оптимальними.

Один із таких аспектів – технологія Nanite. Вона дозволяє робити потік текстури у такому числі полігонів, яке відповідає розширенню дісплею, що робить навіть дуже деталізовані об’єкти досить легковісними для швидкодії. Для активації Nanite, усі статичні об’єкти сцени треба перемкнути на потрібний режим.

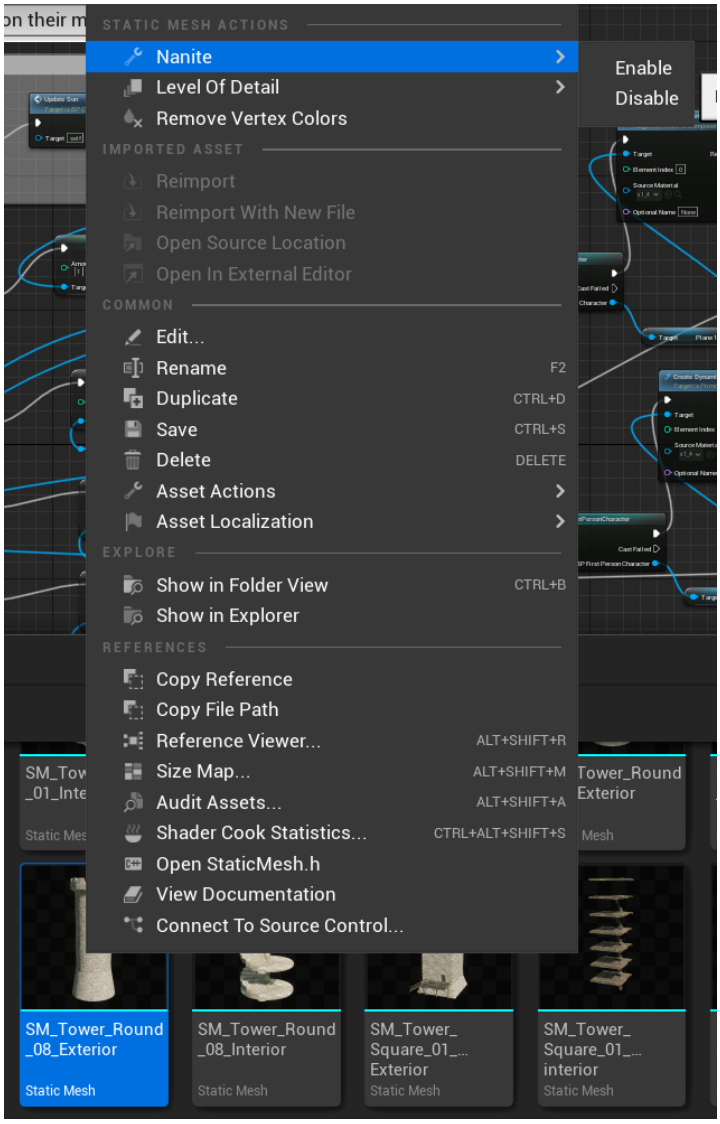


Рис. 3.85 Активація технології Nanite

3.3.2. Видалення зайвих асетів

Інша проблема, яка негативно впливає на рушій і швидкодію проекту – велика кількість зайвих асетів, тобто залишкових моделей, матеріалів, текстур, анімацій, тощо. При створенні часто використовуються готові пакети програм, ідентичні копії вже наявних асетів, які доволі тяжко згодом видаляти, щоб не завдати шкоди потрібним асетам. Тоді як можна видаляти усі ці асети вручну, куди більш оптимальне рішення – спеціальний плагін ProjectCleaner. Цей плагін сам сканує увесь контент проекту на предмет асетів, які не використовуються.

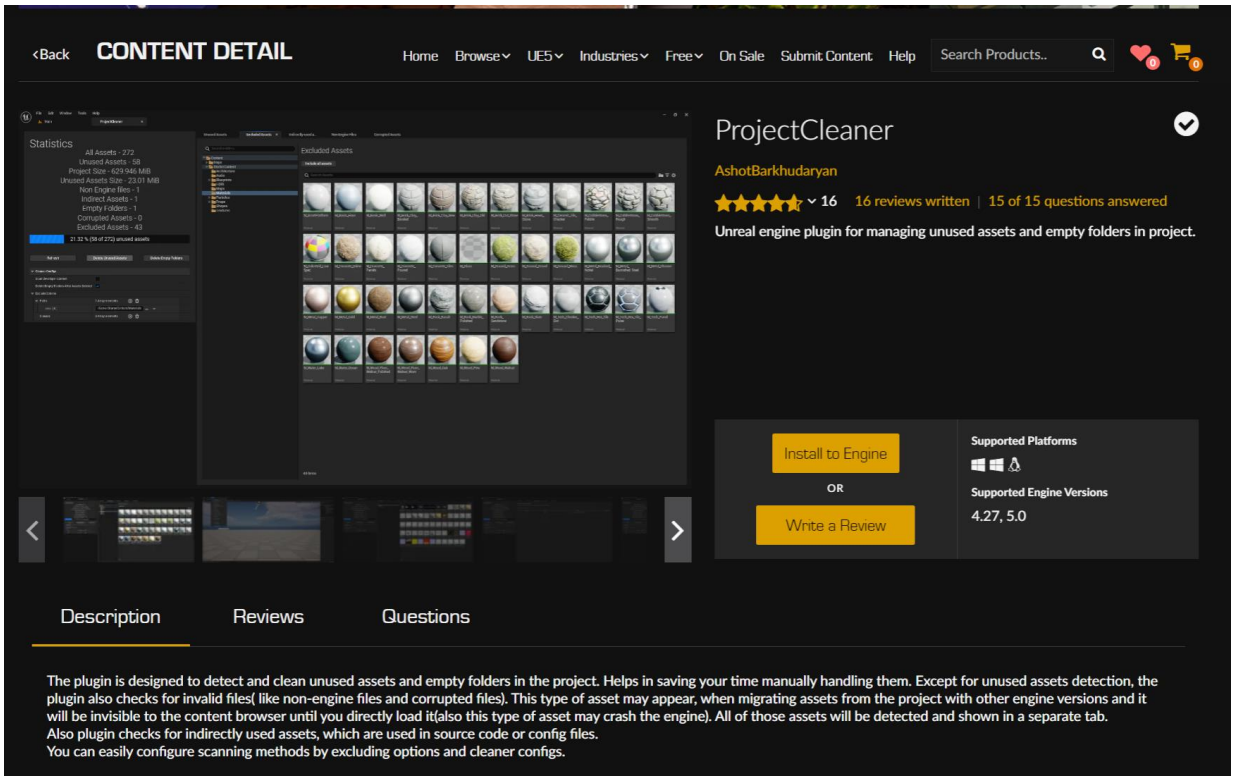


Рис. 3.86 Сторінка плагіну Project Cleaner

Після сканування усіх необхідних папок, у кваліфікаційному проекті було виявлено понад 1600 асетів, що не використовуються. Враховуючи, що сумарна кількість асетів – 2205, це гігантське число «сміття» у проекті, яке доволі негативно впливає на швидкодію проекту. Окрім цього, було також з’ясовано, що проект має 65 пустих папок, які також негативно впливають на

обрахунки проекту. Та завдяки плагіну, після певного часу обробки, усі непотрібні асети були видалені, а проект полегшав приблизно на 1.5 гігабайт.

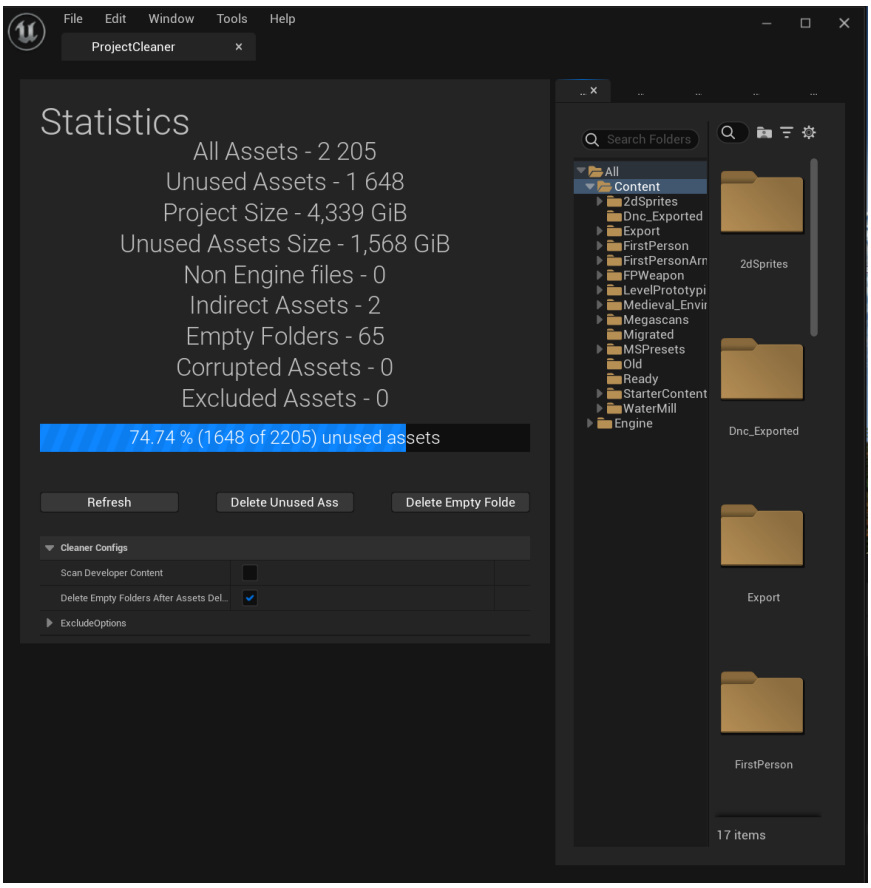


Рис. 3.87 Сторінка плагіну Project Cleaner

ВИСНОВКИ ДО РОЗДІЛУ 3

В даному розділі було створено проект симуляції зміни циклу дня та ночі. Основа проекту була написана на графічній мові програмування Blueprint, був використаний ряд засобів візуалізації ефектів на базі Unreal Engine 5, серед яких Niagara System для створення анімації дощу, Post Processing для налаштування освітлення, Nanite для оптимізації проекту, та Volumetric Clouds для анімації хмар. Для створення користувацького інтерфейсу був використаний додаток Photoshop.

Усі функції та об’єкти були інкапсульовані в одному класі, задля легкості імпортування у інші проекти. Готовий демонстраційний проект був зкомпільований для запуску на Windows OS.

ВИСНОВКИ

Під час кваліфікаційної роботи було детально розглянуто специфіку створення тривимірних рушіїв, елементів реалізму, які застосовуються у сфері кіно та відеоігр, та важливість освітлення у цьому процесі безпосередньо. Було обрано рушій Unreal Engine 5, як найбільш прогресивний рушій із найбільшим вибором інноваційних технологій та загальний юзер френдлі підхід.

Основні технології, які були вирішальним аргументом за Unreal Engine, були нові елементи п’ятої ітерації рушія. Було розглянуто технології віртуальної геометрії Nanite, як системи стримінгу моделей нового покоління, що дозволяє нехтувати LOD-ами для статичних моделей, і дозволяє використовувати безліч моделей з величезною кількістю полігонів без втрати швидкодії.

Також було розглянуто систему динамічного глобального освітлення та відображень Lumen, яка змагається із Ray Tracing, та здатна оброблювати світло та відображення у реальному часі без допомоги RTX ядер.

Розглянуто продукти сім’ї Quixel, яку нещодавно придбала Epic Games, компанія – власник Unreal Engine. Серед них Quixel Megascans – портал із безкоштовними відсканованими із реального світу моделями, текстурами та матеріалами фотореалістичної якості, Quixel Bridge – плагін для Unreal Engine із функціоналом Megascans, Quixel Mixer – програму для фарбування та корегування текстур на об’ємних моделях, безкоштовному аналогу Adobe Substance Painter. Усі ці утиліти є безкоштовними при застосуванні із Unreal Engine 5, і є одними із найкращих в індустрії.

Також було розглянуто технології освітлення, які застосовуються у Unreal Engine, а саме налаштування рівнів Альбедо, технології зображень динамічного діапазону HDRI, спрямованого сяйва та його налаштувань, тіней та їх впливу на аутентичність зображення, важливість туману як елементу оптимізації та візуальної реалістичності, канали освітленості як засіб створення унікально виглядаючих сцен та оточення, корегування світла за

допомогою Lightmass portals, та створення динамічних погодних умов за допомогою Volumetric Clouds.

Після дослідження усіх вищезгаданих аспектів, було розроблено проект симуляції зміни циклу дня та ночі на базі Unreal Engine 5, було використано сцену, на якій було протестовано та реалізовано цілий ряд систем освітлення та симуляції, було створено функцію дощу із хмарами, для тестування для демонстрації більш широкого спектру ситуацій для симуляції поведінки світла при зміні циклу дня та ночі.

Усі аспекти були детально налаштовані для найбільшої реалістичності та аутентичності. Було також створено користувацький інтерфейс, намальований у Photoshop, та безпечно і безпомилково прив’язаний до функцій проекту.

Усі основні об’єкти проекту, що стосуються освітлення, зміни циклу дня та ночі, систем погоди, а також класи та їх функції, були максимально інкапсульовані у одному класі, який можна зручно експортувати в інші проекти, що виповнює задачу створення Standalone-плагіну. Такий підхід дозволяє використовувати майже усі функції освітлення та погоди в інших проектах усього – навсього додавши створений клас до потрібної сцени.

Після усіх налаштувань проект також був оптимізований для покращення швидкодії та ергономіки, усі активні статичні моделі були адаптовані для використання із технологією Nanite для зменшення навантаження на процесор та пам’ять комп’ютера, було видалено понад 1600 непотрібних файлів, із 2100 файлів проекту загалом, що зробило кінцевий проект найбільш доступним до використання.

Таким чином, цілі кваліфікаційної роботи були досягнуті, проект функціонує як заплановано, та проведена робота над його оптимізацією.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

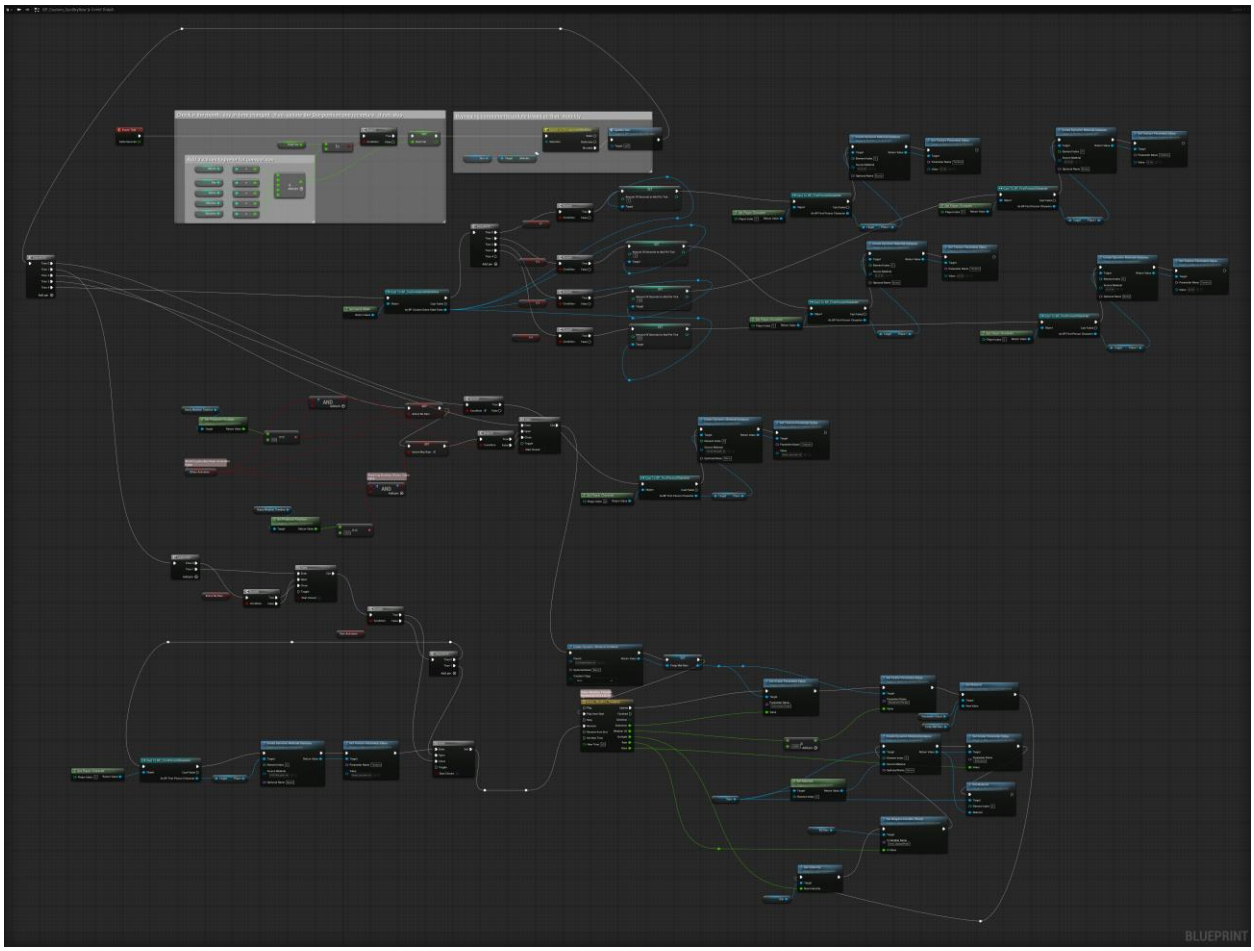
1. Графіка як у фільмі. Або чому технології Unreal Engine 5 – це майбутня революція у геймдеві. dev.ua. URL: https://dev.ua/news/unreal-engine-1661669748 (дата звернення: 08.09.2022).
2. 25 порад для Unreal Engine 4 - 3DAS. 3DAS. URL: https://3das.com.ua/25-porad-dlya-unreal-engine-4/ (дата звернення: 08.09.2022).
3. Unreal Engine: що варто знати новачку про «занадто складне» ПЗ, на якому створюються шедеври • Voki Games. Voki Games. URL: https://vokigames.com/ua/unreal-engine-scho-varto-znatu-novachku-pto-zanadto-skladne-pz-na-yakomu-stvoruutsya-skhedevru/ (дата звернення: 08.09.2022).
4. Nanite virtualized geometry. Unreal Engine 5 Documentation | Unreal Engine 5.0 Documentation. URL: https://docs.unrealengine.com/5.0/en-US/nanite-virtualized-geometry-in-unreal-engine/ (дата звернення: 08.09.2022).
5. Lumen global illumination and reflections. Unreal Engine 5 Documentation | Unreal Engine 5.0 Documentation. URL: https://docs.unrealengine.com/5.0/en-US/lumen-global-illumination-and-reflections-in-unreal-engine/ (дата звернення: 08.09.2022).
6. Quixel Megascans - The world's largest scanned 3D asset library. Quixel. URL: https://quixel.com/megascans (дата звернення: 08.09.2022).
7. Quixel Bridge - Manage 3D content and export with one click. Quixel. URL: https://quixel.com/bridge (дата звернення: 08.09.2022).
8. Quixel Mixer - All-in-one texturing & material creation tool. Quixel. URL: https://quixel.com/mixer (дата звернення: 08.09.2022).
9. Albedo. Unreal Engine 5 Documentation | Unreal Engine 5.0 Documentation. URL: https://docs.unrealengine.com/5.0/en-US/API/Runtime/Engine/Materials/UMaterialExpressionStrataVolumet-/Albedo/ (дата звернення: 15.09.2022).
10. HDRI backdrop. Unreal Engine 5 Documentation | Unreal Engine 5.0 Documentation. URL:https://docs.unrealengine.com/4.27/en-

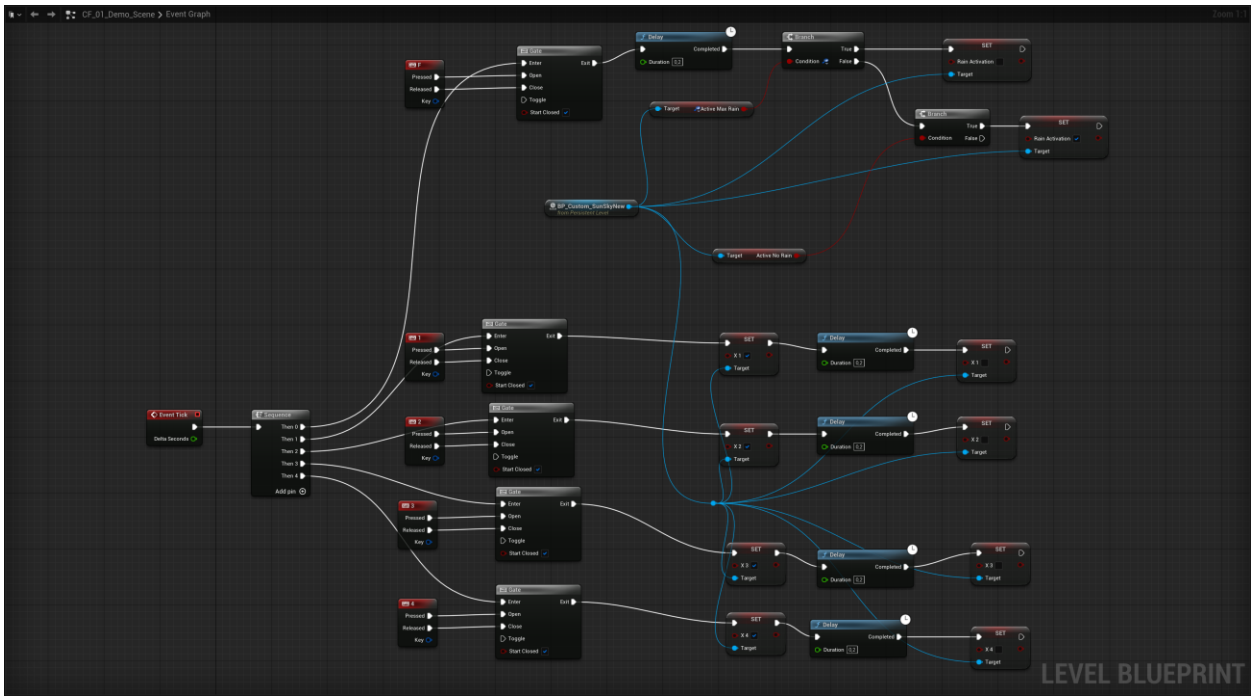
US/BuildingWorlds/LightingAndShadows/HDRIBackdrop/ (дата звернення: 15.09.2022).
11. Directional lights. Unreal Engine 5 Documentation | Unreal Engine 5.0 Documentation. URL:https://docs.unrealengine.com/4.27/en-US/BuildingWorlds/LightingAndShadows/LightTypes/Directional/ (дата звернення: 15.09.2022).
12. Shadow casting. Unreal Engine 5 Documentation | Unreal Engine 5.0 Documentation. URL: https://docs.unrealengine.com/4.27/en-US/BuildingWorlds/LightingAndShadows/Shadows/ (дата звернення: 15.09.2022).
13. Exponential height fog user guide. Unreal Engine 5 Documentation | Unreal Engine 5.0 Documentation. URL: https://docs.unrealengine.com/4.27/en-US/BuildingWorlds/FogEffects/HeightFog/ (дата звернення: 15.09.2022).
14. Lighting channels. Unreal Engine 5 Documentation | Unreal Engine 5.0 Documentation. URL: https://docs.unrealengine.com/4.27/en-US/BuildingWorlds/LightingAndShadows/LightingChannels/ (дата звернення: 15.09.2022).
15. Lightmass portals. Unreal Engine 5 Documentation | Unreal Engine 5.0 Documentation. URL: https://docs.unrealengine.com/4.27/en-US/BuildingWorlds/LightingAndShadows/LightmassPortals/ (дата звернення: 15.09.2022).
16. Volumetric clouds. Unreal Engine 5 Documentation | Unreal Engine 5.0 Documentation. URL: https://docs.unrealengine.com/4.27/en-US/BuildingWorlds/LightingAndShadows/VolumetricClouds/ (дата звернення: 15.09.2022).
17. Medieval castle modular vol 1 in environments - UE marketplace. Unreal Engine. URL: https://www.unrealengine.com/marketplace/en-US/product/medieval-castle-modular-vol (дата звернення: 01.11.2022).
18. Geographically accurate sun positioning. Unreal Engine 5 Documentation | Unreal Engine 5.0 Documentation. URL: https://docs.unrealengine.com/4.27/en-

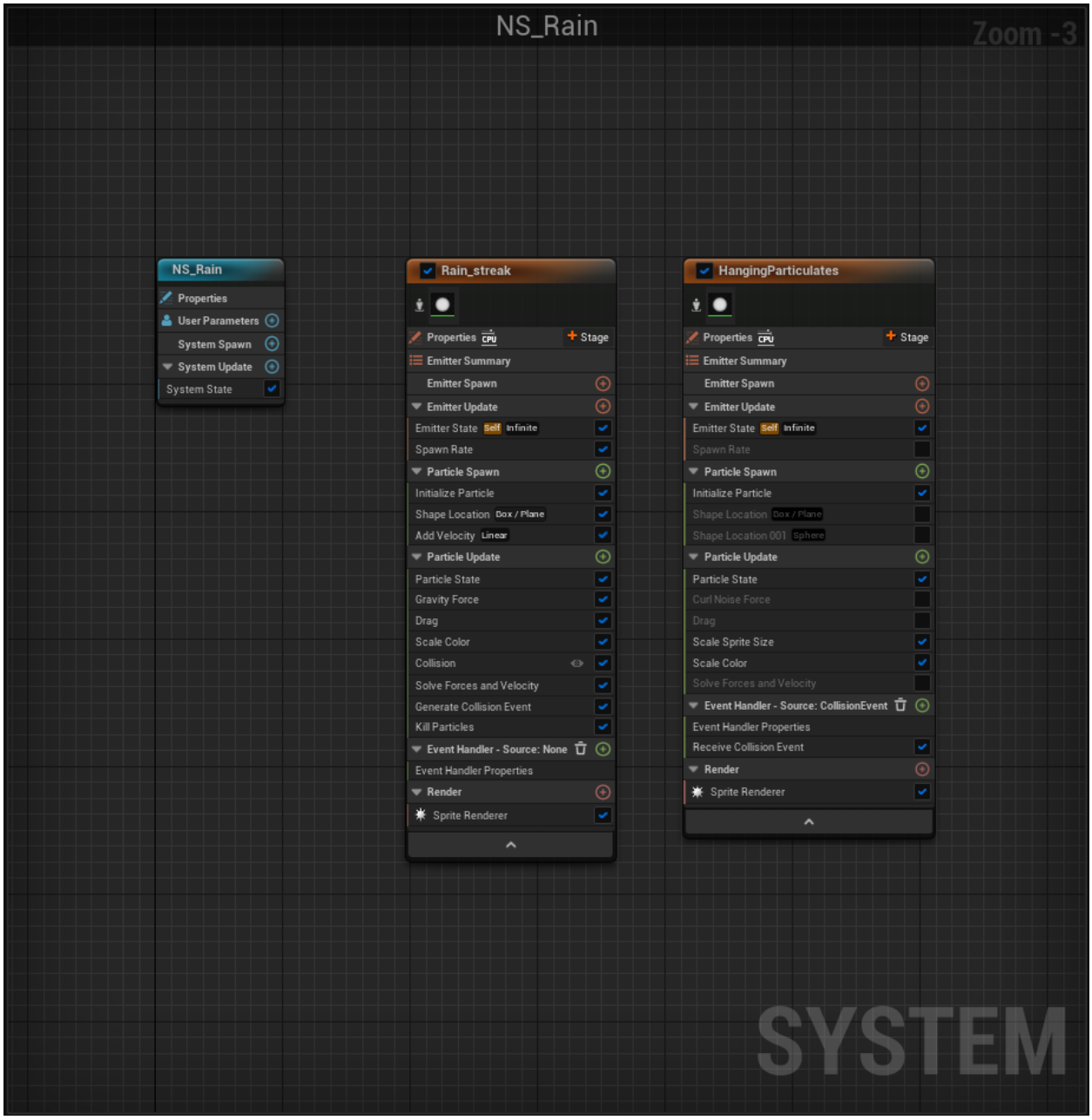
US/BuildingWorlds/LightingAndShadows/SunPositioner/ (дата звернення: 01.11.2022).

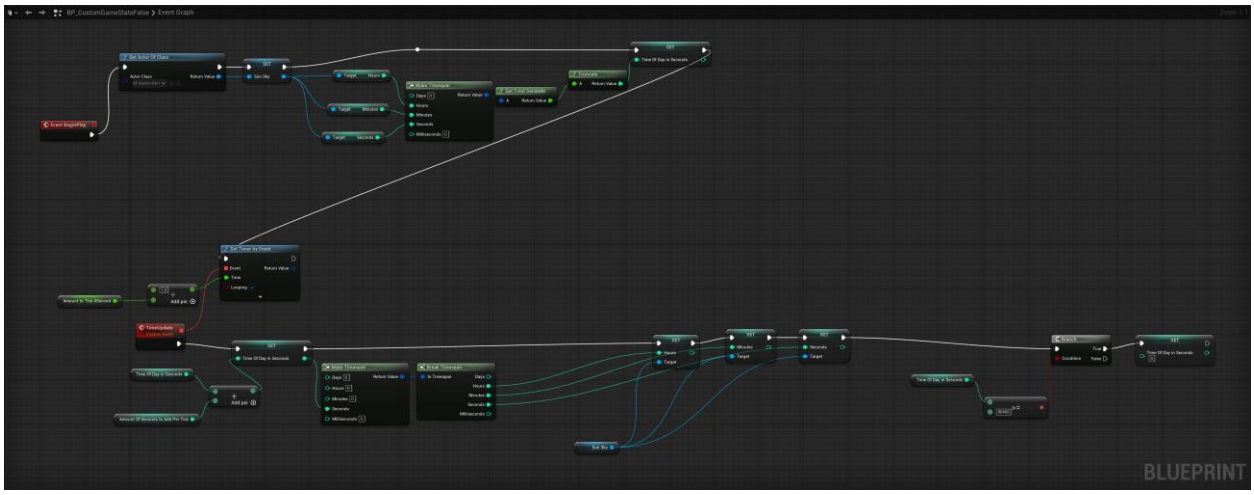
19. Getting the most out of noise in UE4. Unreal Engine. URL: https://www.unrealengine.com/en-US/tech-blog/getting-the-most-out-of-noise-in-ue4 (дата звернення: 01.11.2022).
20. Li J., Arevalo K., Tovar M. Creating games with unreal engine, substance painter, & maya. CRC Press, 2020. URL: https://doi.org/10.1201/9781003053101 (дата звернення: 09.11.2022).
21. Sanders A. An introduction to unreal engine 4. A K Peters/CRC Press, 2016. URL: https://doi.org/10.1201/9781315382555 (дата звернення: 09.11.2022).
22. Pv S. Beginning unreal engine 4 blueprints visual scripting. Berkeley, CA : Apress, 2021. URL: https://doi.org/10.1007/978-1-4842-6396-9 (дата звернення: 10.11.2022).
23. Newton P. Programming in unreal engine 4 with blueprints: the game changer. Pearson Education, Limited, 2018. 384 с. (дата звернення: 12.11.2022).
24. Shannon T. Unreal engine 4 for design visualization: developing stunning interactive visualizations, animations, and renderings. Addison Wesley, 2017. 384 с. (дата звернення: 12.11.2022).

ДОДАТОК






NS_Rain
Zoom -3
NS_Rain
Properties
User Parameters
System Spawn
System Update
System State
Rain_streak
Properties CPU
Stage
Emitter Summary
Emitter Spawn
Emitter Update
Emitter State Self Infinite
Spawn Rate
Particle Spawn
Initialize Particle
Shape Location Box / Plane
Add Velocity Linear
Particle Update
Particle State
Gravity Force
Drag
Scale Color
Collision
Solve Forces and Velocity
Generate Collision Event
Kill Particles
Event Handler - Source: None
Event Handler Properties
Render
Sprite Renderer
HangingParticulates
Properties CPU
Stage
Emitter Summary
Emitter Spawn
Emitter Update
Emitter State Self Infinite
Spawn Rate
Particle Spawn
Initialize Particle
Shape Location Box / Plane
Shape Location 001 Sphere
Particle Update
Particle State
Curl Noise Force
Drag
Scale Sprite Size
Scale Color
Solve Forces and Velocity
Event Handler - Source: CollisionEvent
Event Handler Properties
Receive Collision Event
Render
Sprite Renderer
SYSTEM


BLUEPRINT