МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет

імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інженерії програмного забезпечення _____ Є. О. Давиденко
підпис

«17» лютого 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

Комп'ютерна 3D гра з процедурною генерацією ігрового середовища

Спеціальність «Інженерія програмного забезпечення»

121 –КРМ – 608м.21710829

Студент _____ Д. В. Яшников
підпис
«17» лютого 2023 р.

Керівник PhD, старший викладач _____ І. О. Кандиба
підпис
«17» лютого 2023 р.

Консультант професор _____ Л. І. Григор'єва
підпис
«17» лютого 2023 р.

Миколаїв – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет імені Петра Могили
Факультет комп'ютерних наук
Кафедра інженерії програмного забезпечення

ЗАТВЕРДЖУЮ

Зав. Кафедри _____ Є.О. Давиденко

«04» листопада 2023 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи магістра

Видано студенту групи 608м факультету комп'ютерних наук
Яшников Денис Вячеславович
(прізвище, ім'я, по батькові студента)

1. Тема кваліфікаційної роботи
Комп'ютерна 3D гра з процедурною генерацією ігрового середовища

Затверджена наказом по ЧНУ від «03» листопада 2023 р. № 200

2. Строк представлення кваліфікаційної роботи «24» лютого 2023р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні
Очікуваним результатом є комп'ютерна 3D гра в жанрі покрокових стратегій в середовищі Unreal Engine 4 з процедурною генерацією.

4. Перелік питань, що підлягають розробці
Аналіз предметної галузі, дослідження аналогів; моделювання та проєктування; розробка процедурної генерації ігрового середовища; розробка комп'ютерної 3D гри; тестування.

5. Перелік графічних матеріалів

Слайди презентації

6. Завдання до спеціальної частини
Оцінка умов праці на робочому місці

7. Консультанти:

Консультант	Кафедра (організація)	Частина роботи
Л. І. Григор'єва	екології	Спеціальна частина з охорони праці

Керівник роботи: PhD, старший викладач Кандиба І. О.
(посада, прізвище, ім'я, по батькові)

_____
(підпис)

Завдання прийнято до виконання

Яшников Д. В.
_____

(прізвище, ім'я, по батькові студента)

_____

Дата видачі завдання «04» листопада 2023р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

Тема: Комп'ютерна 3D гра з процедурною генерацією ігрового середовища

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Розробка та затвердження завдання на виконання КРМ	04.11.2022	07.11.2022	Виконано
2.	Огляд літератури за темою роботи	12.11.2022	24.11.2022	Виконано
3.	Аналіз предметної галузі	24.11.2022	27.11.2022	Виконано
4.	Аналіз процедурної генерації	28.11.2022	01.12.2022	Виконано
5.	Аналіз інструментів створення процедурної генерації	01.12.2022	16.12.2022	Виконано
6.	Моделювання об’єкту та предмету дослідження	19.12.2022	24.12.2022	Виконано
7.	Розробка комп’ютерної 3D гри	25.12.2022	02.01.2023	Виконано
8.	Розробка процедурної генерації	02.01.2023	25.01.2023	Виконано
9.	Розробка спеціальної частини з охорони праці	25.01.2023	26.01.2023	Виконано
10.	Аналіз отриманих результатів	26.01.2023	01.02.2023	Виконано
11.	Відгук керівника КРМ	01.02.2023	01.02.2023	Виконано
12.	Оформлення КРМ	01.02.2023	06.02.2023	Виконано
13.	Оформлення презентації	06.02.2023	06.02.2023	Виконано
14.	Попередній захист	07.02.2023	12.02.2023	Виконано
15.	Рецензування			Виконано
16.	Завершення оформлення КРМ та презентації	12.02.2023	17.02.2023	Виконано
17.	Захист кваліфікаційної роботи	24.02.2023	24.02.2023	Виконано

Розробив студент _____Яшников Д. В._____ _____
(*прізвище, ім’я, по батькові)* *(підпис)*

«__» _____ 20__ р.

Керівник роботи ___PhD, старший викладач Кандиба І. О.___ _____
(*посада, прізвище, ім’я, по батькові)* *(підпис)*

«__» _____ 20__ р.

АНОТАЦІЯ

до кваліфікаційної роботи магістра

«Комп'ютерна 3D гра з процедурною генерацією ігрового середовища»

Студент 608м гр.: Яшников Денис Вячеславович

Керівник: старший викладач Кандиба І. О.

Мета роботи – покращення ігрового процесу за рахунок процедурної генерації ігрового середовища шляхом використання системи Blueprint.

Завдання: аналіз предметної галузі, дослідження алгоритмів процедурної генерації ігрового контенту, моделювання та проєктування гри, розробка комп’ютерної 3D гри з процедурною генерацією, тестування гри, дослідження результатів застосування процедурної генерації ігрового середовища.

Об'єктом дослідження є процес процедурної генерації ігрового середовища.

Предметом дослідження є технологія Blueprint в Unreal Engine 4 у якості ігрового рушія для 3D комп'ютерної гри з процедурною генерацією ігрового середовища.

У вступі розкривається актуальність використання процедурної генерації, програмного рушія Unreal Engine та системи Blueprint, ставиться проблема, мета і завдання дослідження, визначаються об'єкт та предмет дослідження.

У першому розділі роботи проводиться системний аналіз обраної предметної області та, на його основі, формулюється постановка завдань та специфікація вимог до гри.

У другому розділі описується моделювання та проєктування гри.

У третьому розділі описується реалізація процедурної генерації, освітлення, робота з ландашфтом та ігровим середовищем.

У четвертому розділі описується реалізація гри, а саме: атаки персонажів, система досвіду та рівнів, зберігання гри, робота з ефектами та анімаціями та тестування продуктивності гри.

У висновках проводиться аналіз роботи та отриманих результатів.

У спеціальній частині з охорони праці та безпеки в надзвичайних ситуаціях описано техніку безпеки при роботі в приміщеннях із комп'ютерним обладнанням.

КРМ викладена на 83 сторінки, вона містить 4 розділи, 95 ілюстрацій, 1 таблицю, 25 джерел в переліку посилань.

Ключові слова: розробка ігор, процедурна генерація, Blueprint, Unreal Engine, ігровий продукт, HUD.

ABSTRACT

to the master's qualification work

"3D computer game with procedural generation of the game environment"

Student of group 608m: Yashnykov Denys Vyacheslavovych

Supervisor: Senior Lecturer Kandyba I.O.

The goal of the work is to improve the game process due to the procedural generation of the game environment by using the Blueprint system.

Tasks: analysis of the subject area, research of algorithms for the procedural generation of game content, modeling and design of the game, development of a computer 3D game with procedural generation, game testing, research of the results of the application of the procedural generation of the game environment.

The object of research is the process of procedural generation of the game environment.

The subject of the study is Blueprint technology in Unreal Engine 4 as a game engine for a 3D computer game with procedural generation of the game environment.

The introduction reveals the relevance of using procedural generation, Unreal Engine and the Blueprint system, sets the problem, goal and task of the research, defines the object and subject of the research.

In the first section of the work, a systematic analysis of the selected subject area is carried out and, on its basis, the statement of the problem and the specification of the requirements for the game are formulated.

The second chapter describes the modeling and design of the game.

The third chapter describes the implementation of procedural generation, lighting, working with the landscape and game environment.

The fourth chapter describes the implementation of the game, namely: character attacks, the experience and level system, saving the game, working with effects and animations, and testing the performance of the game.

In the conclusions, an analysis of the work and the obtained results is carried out.

The special part on occupational health and safety in emergency situations describes safety techniques when working in premises with computer equipment.

MQW is laid out on 83 pages, it contains 4 sections, 95 illustrations, 1 table, 25 sources in the list of links.

Keywords: game development, procedural generation, Blueprint, Unreal Engine, game product, HUD.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

AI – Artificial intelligence

BP – blueprint

PS – Photoshop

PFS – Procedural Foliage Spawner

STD – State Transition Diagrams

LUT – Lookup Table

UE4 – Unreal Engine 4

UI – User Interface

UML – Unified Modeling Language

ЦП – Центральний процесор

ВСТУП

У сучасному світі відеоігри одна із складових індустрії розваг. Сукупна капіталізація в ігровій індустрії перевершує кіновиробництво і великий спорт разом узяті. А світове зростання ринку відеоігор щороку зростає в середньому на 11%. Згідно з даних аналітичного агентства Newzoo, кожна четверта людина на планеті так чи інакше грала в ігри, а кожна третя з них хотіла б працювати в цій індустрії.

В даний момент найбільш популярними в ігровій індустрії рушії для розробки ігор є Unity 3D та Unreal Engine. У двох рушіїв великі набори інструментів, що включають: редактор ландшафтів, симуляцію фізики, анімацію, покращення освітлення, підтримку VR і багато іншого. Але останнім часом можна помітити, що багато розробників з проектами виконаними в Unity, починають перемикатися на продукт Epic Games. Щоб досягти візуального ефекту, схожого на UE4, доведеться змінити систему рендерингу з нормальною на HDRP, замінити світло та скайбокс. Але навіть після трансформації такий самий результат на Unity не вийде. В основному це пов'язано з тим, що Epic Games, як розробник, накопичили великий досвід і щоразу створювали щось нове, а іншим доводилося наздоганяти [1]. Unity більше орієнтований на невеликі проекти, зроблені здебільшого у 2D та для мобільних пристроїв. Тому за рахунок графічних можливостей, що дають змогу створювати відкриті світи більш реалістичними та за рахунок великого набору інструментів є сенс розробляти комп'ютерні ігри саме на Unreal Engine [2].

У нас час відкриті світи в іграх стали на стільки великими, що пробляти їх вручну дуже довго та дорого. Для цього все частіше використовуються спеціальні алгоритми штучного створення світу – процедурна генерація. Процедурна генерація полегшує роботу, тому що допомагає заповнювати 80 відсотків ігрового світу, полегшуючи процес

створення декорацій оточення художникам та геймдезайнери, щоб перевіряти механіки на начерку світу і на ходу додавати нові. При правильному підході процедурна генерація дозволяє зменшити витрати, на кожному етапі розробки, тому що дозволяє зменшити кількість персоналу та збільшити різноманіття локацій. За рахунок цього, процедурна генерація є актуальною на сьогоднішній день.

Мета роботи – покращення ігрового процесу за рахунок процедурної генерації ігрового середовища шляхом використання системи Blueprint.

Досягнення мети можливо шляхом вирішення поставлених завдань:

– аналіз предметної галузі;

– дослідження алгоритмів процедурної генерації ігрового контенту;

– моделювання та проєктування гри;

– розробка комп’ютерної 3D гри з процедурною генерацією;

– тестування гри;

– дослідження результатів застосування процедурної генерації ігрового середовища.

Об'єктом дослідження є процес процедурної генерації ігрового середовища.

Предметом дослідження є технологія Blueprint в Unreal Engine 4 у якості ігрового рушія для 3D комп'ютерної гри з процедурною генерацією ігрового середовища.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ІСНУЮЧИХ АНАЛОГІВ

1.1 Огляд існуючих аналогів

Покрокова стратегія – піджанр стратегічних ігор, у яких ігровий процес складається з послідовності ходів (або кроків), під час яких гравці здійснюють свої дії. Основною характеристикою покрокових стратегічних ігор є дискретність ігрового процесу. Ігри складаються з фіксованих у часі моментів (кроків або ходів), які завершуються тільки за командою гравця. Під час цих ходів гравець здійснює свої дії. У більшості покрокових стратегій гравці здійснюють ходи по черзі, як у таких класичних настільних іграх, як шахи або монополія. Такі ігри розвивають логічне мислення гравців.

Аналогами є Heroes of Might and Magic та The Battle for Wesnoth. У процесі гри гравець керує героями – ігровими персонажами, які досліджують глобальну карту та персонажами свого загону під час бою. Виграючи битви, герої набувають досвіду, і, набравши достатню його кількість, переходять на наступний рівень, збільшуючи свої параметри.

Heroes of Might and Magic – фентезійна серія комп'ютерних ігор у жанрі покрокової стратегії з елементами RPG [3]. Порядок руху пов'язаний з атрибутом швидкості юнітів – чим вище швидкість, тим швидше (і далі) буде рухатися юніт (рис. 1.1). На порядок рухів також можуть впливати певні артефакти та заклинання, які збільшують або зменшують швидкість.

Рисунок 1.1 – Поле бою у грі Heroes of Might and Magic

Вся бойова статистика від атаки монстра/іншого героя (рис. 1.2) іншим героєм – переміщується на невелике поле бою, розділене на сіткоподібні поля (гекси), що представляють місця, у яких можна розмістити одиниці. Атрибут швидкості, який впливає на порядок руху, також визначає радіус дії юніта – юніт з 10 балами швидкості зможе подолати відстань у 10 гексів. Усе поле бою має 15 гексів завширшки та 11 гексів заввишки. Крім того, на полі бою в основному з'являються перешкоди місцевості, які сильно обмежують або навіть перешкоджають руху в заданому напрямку. Більшість одиниць у грі займають 1 гекс, але деякі з них можуть мати 2 гекси.

Рисунок 1.2 – Бойова статистика

Крім того, на екрані є панель інтерфейсу, яка дозволяє керувати різноманітними ігровими механізмами, такими як:

– Параметри бою – після натискання на нього з’явиться екран, на якому можна налаштувати параметри, пов’язані з боєм, наприклад швидкість анімації юнітів.

– Варіант капітуляції – варіант, який дозволяє здатися під час бою. Герой, який вирішив здатися, повинен заплатити «штраф», який є частиною загальної вартості його одиниць.

– Опція втечі – після її активації герой втече з поля бою, залишивши всі юніти.

– Діалогове вікно – це вікно розташоване в нижній, центральній частині екрана. Там можна знайти всю важливу інформацію про хід битви, наприклад, кількість шкоди, завданої різними істотами, події, що відбулися під час битви (низький/високий моральний ефект) тощо.

– Книга заклинань – дозволяє отримати доступ до заклинань, які є у розпорядженні героїв.

– Опція «Почекати» – дозволяє юніту зберегти свій хід на кращий момент (кінець раунду). Після того, як усі інші підрозділи перемістяться, настає час для тих, які використовували опцію «чекати».

– Опція «Захищатися» – частина втратить хід, але натомість отримає значний прискорений атрибут захисту.

Битву виграє ворог, що залишився – програє той, хто втратить усі одиниці. Після закінчення битви переможець отримує очки досвіду та будь-які артефакти, які були в інвентарі переможеного героя.

Недоліками гри є недостатньо розвинена система покращення характеристик юнітів, застаріла графіка та набридливий ракурс камери.

Наступним аналогом є гра The Battle for Wesnoth (рис 1.3).

Рисунок 1.3 – Інтерфейс The Battle for Wesnoth

The Battle for Wesnoth – це покрокова стратегічна комп'ютерна гра у жанрі фентезі [4]. Графічне оформлення гри двовимірне, з анімованими персонажами та деякими деталями ландшафту. Ігровий процес містить елементи як класичної покрокової стратегії, і рольової гри. Перед початком кожної нової битви показується вікно, де викладено умови перемоги та поразки, а часом і корисну пораду. Кожен юніт у грі має певні бойові якості, запас здоров'я (hit points), власне ім'я (крім механізмів і монстрів), іноді також з власним характером та історією, що розкриваються по ходу сюжету.

У ході битв (рис. 1.4), а особливо – перемагаючи супротивника, боєць набуває досвіду, який після досягнення певного значення дозволяє йому перейти на новий рівень і перетворитися на більш живучого і важко озброєного воїна. Результат кожного конкретного поєдинку окремо взятих бійців залежить багатьох чинників: час доби, ландшафт, тип зброї (атаки), раса, клас бійця, індивідуальні особливості, досвід, стан здоров'я. Також є елемент випадковості, який іноді повністю нівелює ці відмінності. Тим не менш, у середньому за грою відхилення від прогнозованих значень влучень незначні, і виграти цілу партію на чистому везінні практично неможливо.

Рисунок 1.4 – Процес битв у грі

Вороги відрізняються непоганим інтелектом: застосовують атаки з кількох напрямків, вибирають війська, що найбільш підходять до ситуації, атакують найслабших бійців (або ключових персонажів, знищення яких означає поразку гравця) і відводять назад своїх потерпілих у бою воїнів. Тому метод примітивного «тиску числом» проти комп'ютера не працює, необхідно ретельно продумувати тактику. Однак комп'ютерний гравець ніколи повністю не переходить до оборони, продовжує атакувати, навіть зазнавши великих втрат і опинившись у меншості, що полегшує перемогу над ним. Економічна сторона у грі розвинена слабо. На початку кожного сценарію дається мінімальна сума грошей. У кампаніях при достроковому завершенні попереднього сценарію гравець отримує на додачу грошовий бонус за кожен невикористаний хід. Кожне контрольоване поселення при стандартних налаштуваннях дає по дві одиниці золота в скарбницю, а також один золотий для утримання бійців. Іноді у сценаріях зустрічаються грошові скарби. Зібрані таким чином кошти витрачаються на вербування солдатів та виплату їм платні.

Недоліком гри є недостатньо розвинена система розвитку персонажів та застаріла графіка.

Перевагою розроблюваної є створення системи покращення характеристик героїв за рахунок системи досвіду гравця. Для жанру покрокових стратегій ніколи не використовувалась процедурна генерація, тому це робить розроблювану гру більш якісною.

1.2 Аналіз ігор з процедурною генерацією

Наразі відкриті світи стали настільки великими та опрацьованими, що створювати їх вручну дуже довго та дорого. Сучасна процедурна генерація – це складна система з безліччю умов і параметрів. А щоб результат виглядав переконливо, ця система завжди контролюється живою людиною.

Корисна властивість процедурної генерації, якою часто користуються розробники – економія місця [5]. При розміщенні вручну кожен об'єкт на карті

отримує свою координату, і в сумі ці значення можуть зайняти багато ресурсів. Згенерована рослинність зберігається у вигляді окремої карти та займає менше оперативної пам’яті.

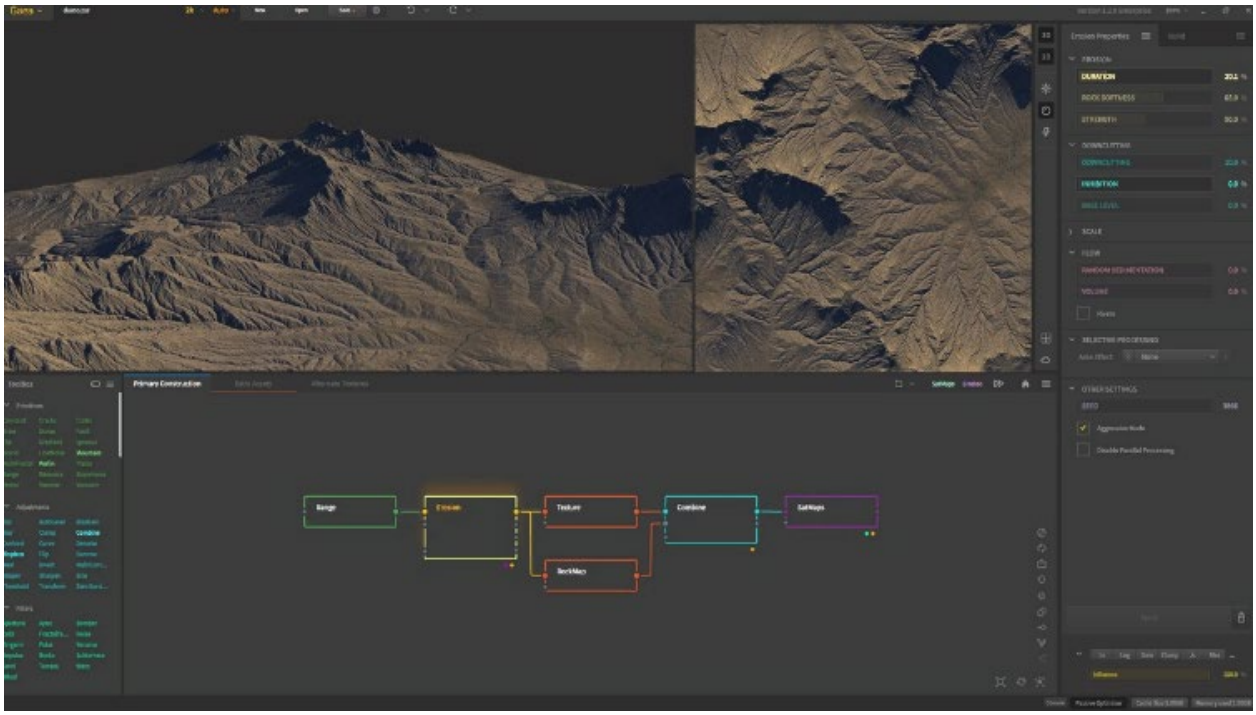


Рисунок 1.5 – Процедурна генерація ландшафту

Команда розробників The Witcher розробила систему, яка дозволяла легко створювати величезні лісові простори, на ходу змішуючи текстури та розставляючи асети. Наприклад, алгоритм «умів» створювати плавні переходи між матеріалами: на стиках двох поверхонь використовували одразу дві текстури, з якими система працювала індивідуально.

Програмісти навчили алгоритм розпізнавати два типи поверхонь: до природних система відносила ґрунт, пісок, кам'янисту місцевість, а до рукотворних – цеглу, бруківку, зроблені з колод стіни. Це допомагало по-різному накладати ефекти на різні типи матеріалів, наприклад, на стіні будинку сніг виглядав не так, як на землі. Завдяки цьому й виходили плавні переходи – з обох боків дороги генератор складав рукотворну текстуру бруківки та природну текстуру землі так, що залишалися видно контури окремих каменів (рис. 1.6).

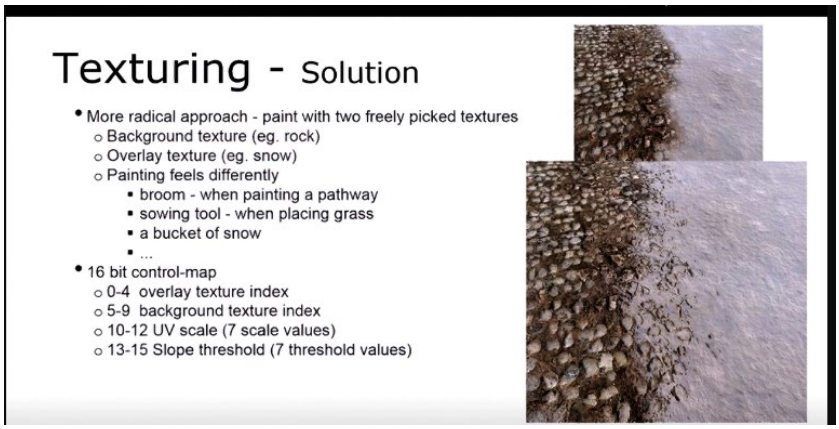


Рисунок 1.6 – Перехід при процедурній генерації

Процедурна генерація також розподіляла поверхнею текстур невеликі об'єкти – каміння чи траву. Художникам потрібно було намалювати лише десять видів трави, а алгоритми підфарбовували їх під колір землі. Для цього левел-артисти використали пігментну карту – текстуру локації з видом зверху в низькій роздільній здатності. Колір конкретного поля або луки накладався на траву градієнтом, щоб ближче до коріння рослинність була кольору землі, а вище зберігався оригінальний колір і не порушувався відтінок плодів та суцвіть.

При цьому процедурно згенерована трава не перекриває деталі вихідної текстури, а підкреслює їх. Для цього програму навчили виділяти на текстурі лише ті місця, де щось може рости. Наприклад, якщо на вимощеній бруківкою старій дорозі місцями проступає текстура землі, то трава виростає лише на цих ділянках (рис 1.7).

Рисунок 1.7 – Процедурно згенерована трава

Алгоритми пензлів розпізнавали особливості рельєфу. Наприклад, вони враховували, в яких місцях вода під час дощу стікає по схилах у долині – там

ліс виростав густішим. У процесі система порівнювала обраний рельєф із рельєфом сусідніх локацій, щоб прорахувати траєкторію сонця та обчислити, скільки місцевість отримує світла.

Усі зібрані дані порівнювалися з типами рослинності, які художники закладали на пензель. Набір дерев у кисті густого північного лісу сильно відрізнявся від пензлів для світлих гаїв на півдні, а ті, у свою чергу, від інструменту, розрахованого на гірську рослинність.

Якщо для конкретного дерева не вистачало води чи сонця, то пензель виключав цей тип із локації. Але навіть якщо ресурсів було достатньо, об'єкт все одно порівнювався із заданими «ідеальними умовами», та його розмір змінювався відповідно. Тому один і той самий китиця, застосована на долину біля озера і на скелястий посушливий пагорб, видавала різний результат.

Також ці пензлі давали різний результат за різної кількості застосувань. Якщо художник використовував пензель на конкретній ділянці один раз, у нього виходив легкий підлісок: чагарник, дерева, що окремо стоять. Але якщо він повторював прийом, ефект ставав помітнішим – чагарники розросталися, дерев ставало більше, а самі вони росли вище.

Працюючи над Horizon Zero Dawn, програмісти Guerrilla Games також використали процедурну генерацію. В результаті студія розробила систему, яка піддавалася налаштуванню і рідко помилялася з розміщенням об'єктів. В основі цієї системи були слої.

У Horizon Zero Dawn художники створили карти доріг, висот, річок та рослинності. Залежно від навколишнього середовища між цими картами вибудовується певна логіка взаємодії. В них зберігається інформація про щільності: чорні значення означають, що у конкретній точці не може бути рослинність, білі – що у ній має бути максимальна кількість рослинності.

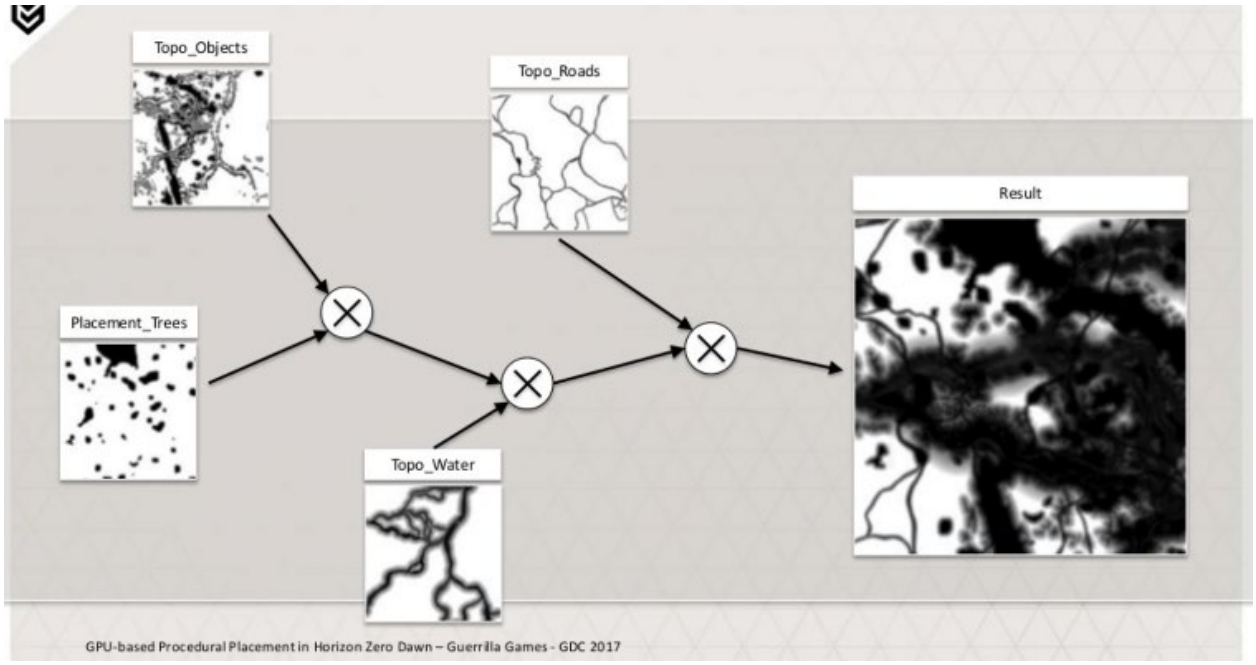


Рисунок 1.8 – Карта шарів для процедурної генерації

Головним шаром служила карта рослинності, на якій художники відзначали, де має рости ліс і наскільки густим він має бути. Ця карта накладалася інші карти місцевості, після чого алгоритм порівнював значення у конкретних точках. Якщо навіть через дуже густий ліс проходила дорога чи річка, система сама змінювала параметри так, щоб під водою не росла трава, а кущі поступово рідшали на підходах до стежки.

Три людини у Guerrilla Games намалювали 500 типів рослин, а шаблонами займався один технічний художник. Налаштувавши параметри генерації один раз, художники студії могли легко створювати певні поєднання рослинності у різних частинах локації. Алгоритм однаково успішно працював і з густими лісами, і з полями, що складаються з трави і чагарника.

Щоб моделі рослинності не заважали одна одній, для кожної вказувався footprint - відбиток, що задає мінімальну відстань між об'єктами одного типу. А щоб результат не виглядав штучно, художники налаштовували окремий параметр, який хаотично розкидував цифри та змінював логіку розміщення. Найчастіше локацію потім віддавали художникам, щоб ті могли її вручну відредагувати.

Рисунок 1.9 – Згенерований світ у Horizon Zero Dawn

Щоб побудувати велике та реалістичне місто у The Sinking City, студія Frogwares створила інструмент процедурної генерації для Unreal Engine 4. Процедурна генерація міста починалася з ручної роботи. Художники створювали «сітку» (рис. 1.10), – макет основних вулиць та орієнтирів і вказували, які типи будинків можуть бути в тому чи іншому районі [6]. А ландшафт, рельєф та ширина вулиць визначалися лише попередньо.

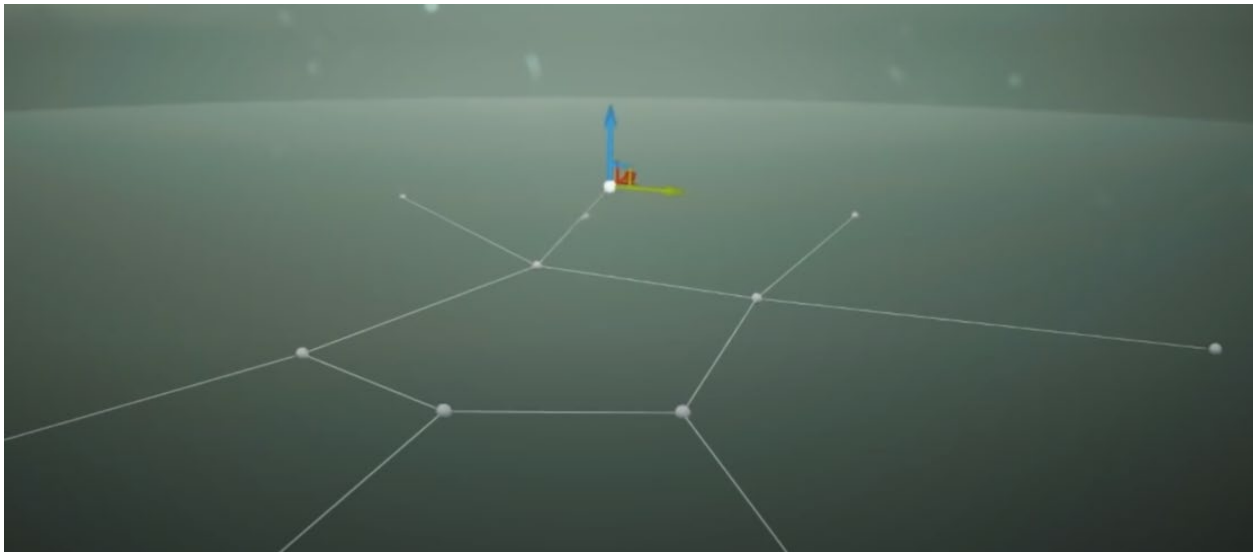


Рисунок 1.10 – Створення макету вулиць

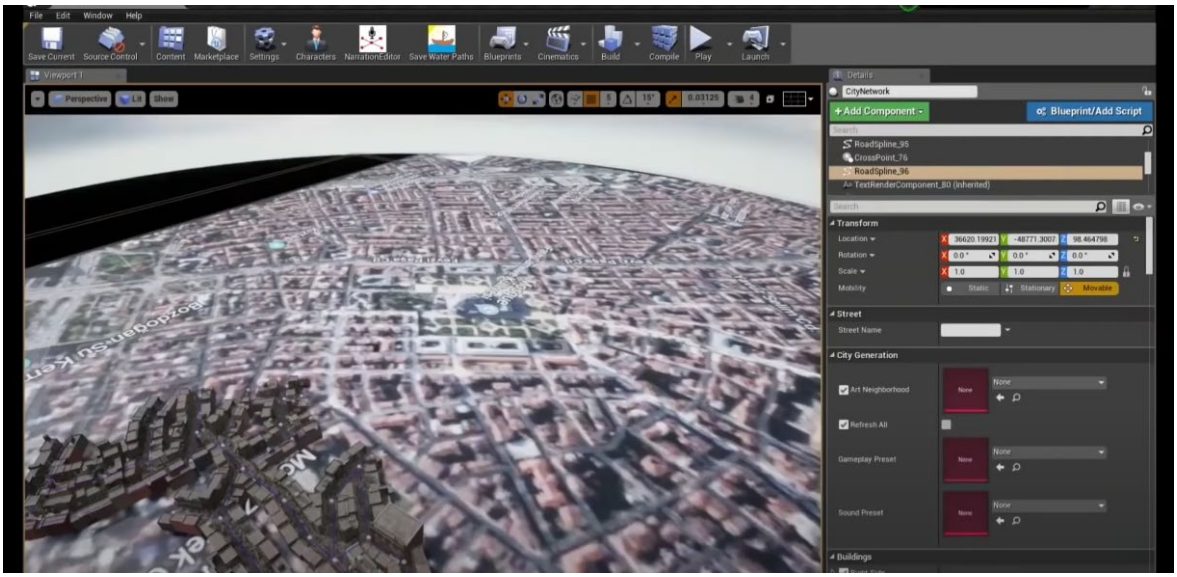


Рисунок 1.11 – Створення міста в Unreal Engine 4

Художник вибирав архітектурний стиль, висоту та ширину будівлі і після цього редактор збирав будинок. Потім моделер може внести редагування - змінити стіну, використати інше вікно, збільшити поверховість або ширину, додати деталей.

Коли всі правки були внесені, редактор об'єднував деталі будівлі на чотири фасади, які зберігав окремо (рис. 1.12). Фрагментоване зберігання будівель потім допомогло розробникам оптимізувати гру.

Рисунок 1.12 – Створення будинку

1.3 Методи процедурної генерації

Процедурна генерація ландшафту в іграх створюється за допомогою карти висот, яка складається з випадкових чисел. Для створення карти висот використовуються методи генерації шуму. Найбільш розвиненими є: шум Перліна, симплексний шум, шум Ворлі. шум-значення.

Шум Перліна – це примітив процедурної текстури, тип градієнтного шуму, який використовується художниками візуальних ефектів для збільшення реалістичності комп’ютерної графіки [7]. Функція має псевдовипадковий вигляд, але всі її візуальні деталі мають однаковий розмір. Ця властивість дозволяє ним легко керувати; численні масштабовані копії шуму Перліна можуть бути вставлені в математичні вирази для створення великої різноманітності процедурних текстур. Метод процедурної генерації за допомогою шуму Перліна може використовуватися для генерації ландшафту – чим темніше точка на текстурі, тим нижче буде знаходитись на самому ландшафті у рушії певна ділянка; чим світліше, тим вище.

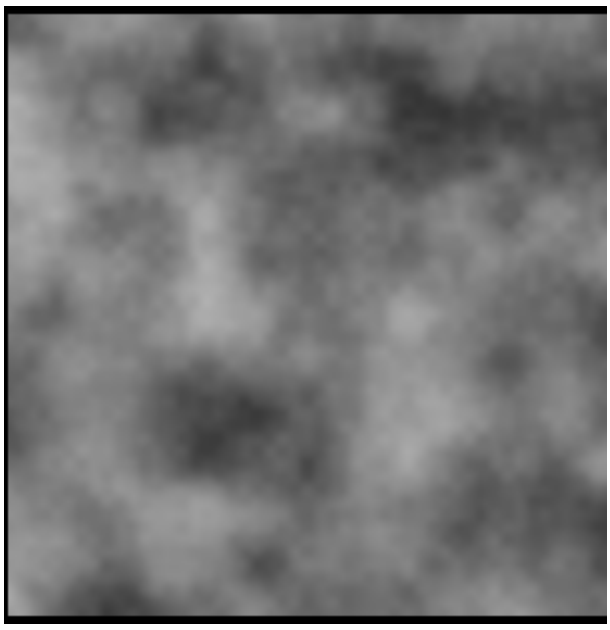


Рисунок 1.13 – Шум Перліна

Симплексний шум є результатом n-вимірної шумової функції, порівнянної з шумом Перліна («класичний» шум), але з меншою кількістю

спрямованих артефактів і, у вищих вимірах, меншими обчислювальними витратами (рис 1.12). Симплексний шум поділяє простір на симплекси [8]. Це зменшує кількість вузлів, проте при більше чотирьох вимірів, симплекси знаходяться недостатньо щільно, тим самим занулюючи функцію на вільних ділянках.

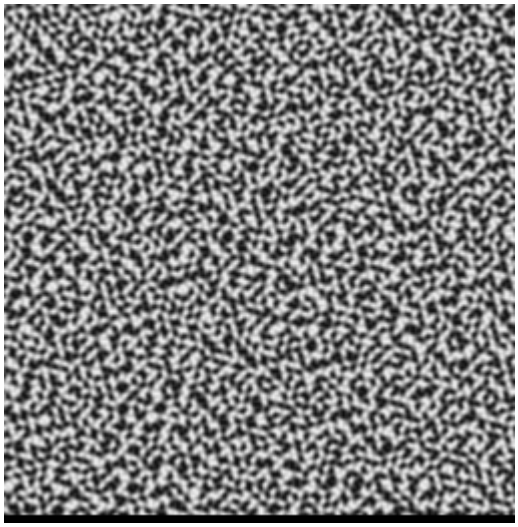


Рисунок 1.14 – Симплексний шум

Шум Ворлі – це шумова функція, представлена Стівеном Ворлі в 1996 р. [9]. У комп’ютерній графіці вона використовується для створення процедурних текстур, тобто текстур, які створюються автоматично з довільною точністю, які не потрібно малювати вручну. Шум Worley наближається до імітації текстур каменю, води або біологічних клітин. Алгоритм вибирає випадкові точки в просторі (2- або 3-вимірному), а потім для кожного розташування в просторі бере відстань dn до n-ї найближчої точки (наприклад, другої найближчої точки) і використовує їх комбінації для виведення інформації про колір (зауважте, що dn+1 > dn). Випадково розподіляє точки в просторі, організованому у вигляді клітинок сітки.

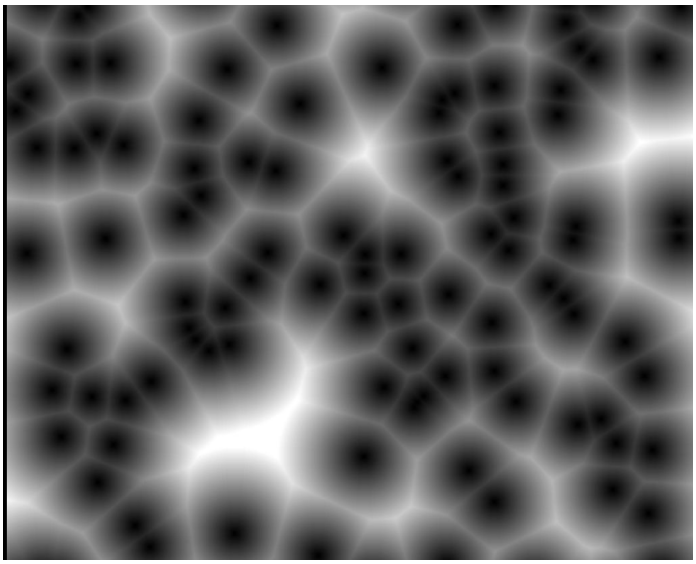


Рисунок 1.15 – Шум Ворлі

Під час створення береться відстань dn від заданого місця до n-ї найближчої початкової точки. Це можна зробити ефективно, обравши поточну комірку та сусідні.

Шум-значення – це тип шуму, який зазвичай використовується як примітив процедурної текстури в комп’ютерній графіці. Він концептуально відрізняється від градієнтного шуму, прикладом якого є шум Перліна та симплексний шум, і його часто плутають з ним. Цей метод полягає у створенні решітки точок, яким присвоюються випадкові значення. Потім функція шуму повертає інтерпольоване число на основі значень навколишніх точок решітки.

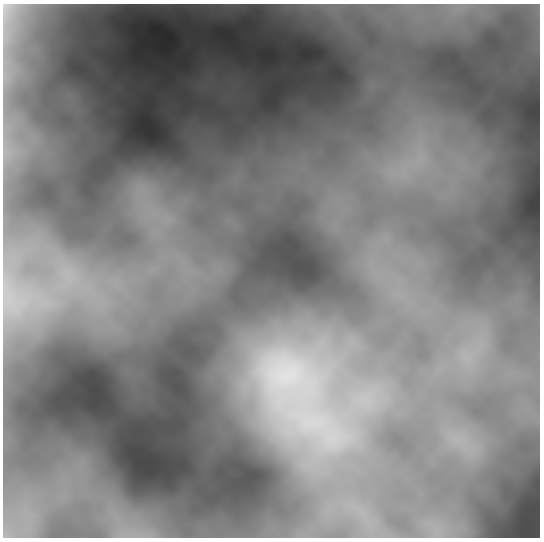


Рисунок 1.16 – Шум Значення

1.4 Інструменти створення процедурної генерації

Gaea – це потужна система генерації рельєфу та сцен, яка створює пейзажі й сцени для мобільних пристроїв, віртуальної реальності, консолей і комп’ютерів за лічені хвилини [10]. Він забезпечує просте, швидке та якісне формування ландшафту, текстурування, насадження та процедурне розміщення вмісту. Gaea заощаджує час, включаючи все, що потрібно, щоб створювати студійні ландшафти, рельєфи та сцени за години та дні замість тижнів і місяців. Gaea дозволяє:

– вибирати розмір світу та цільову платформу для створення рельєфу, виділяючи гори, пагорби, височини, озера та річки;

– створювати біоми, щоб текстурувати, заводити та наповнювати світ процедурно;

– налаштовувати освітлення, вітер, воду, контролер програвача та постфлекс.

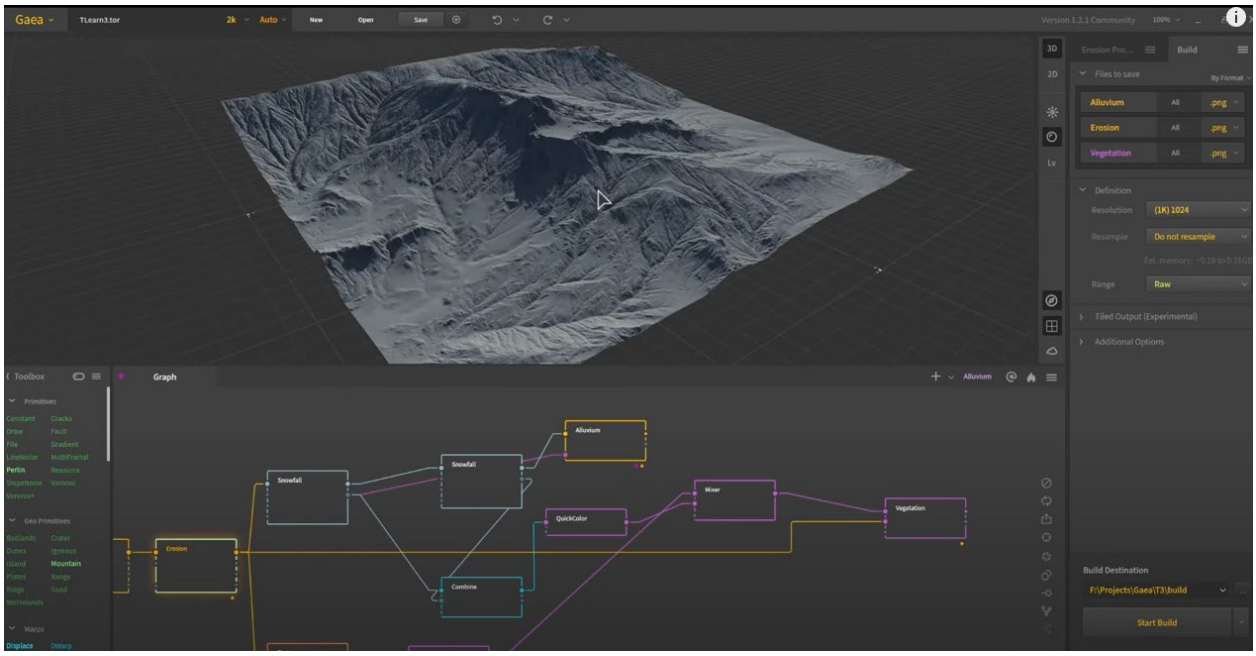


Рисунок 1.17 – Програмне забезпечення Gaea

World Creator – це інструмент для створення рельєфу та ландшафту в режимі реального часу, який виконує процедурні та проектні процеси повністю на графічному процесорі, використовуючи тисячі ядер, поєднуючи

процедурну потужність із творчою свободою та ефективністю робочого процесу в реальному часі [11]. Унікальний і потужний генератор у реальному часі у World Creators дозволяє застосовувати та поєднувати багато різних типів фільтрів, щоб змінювати та адаптувати рельєф. Він дозволяє розмивати, створювати річки та озера, трансформувати, стилізувати, імітувати потік води, у режимі реального часу.

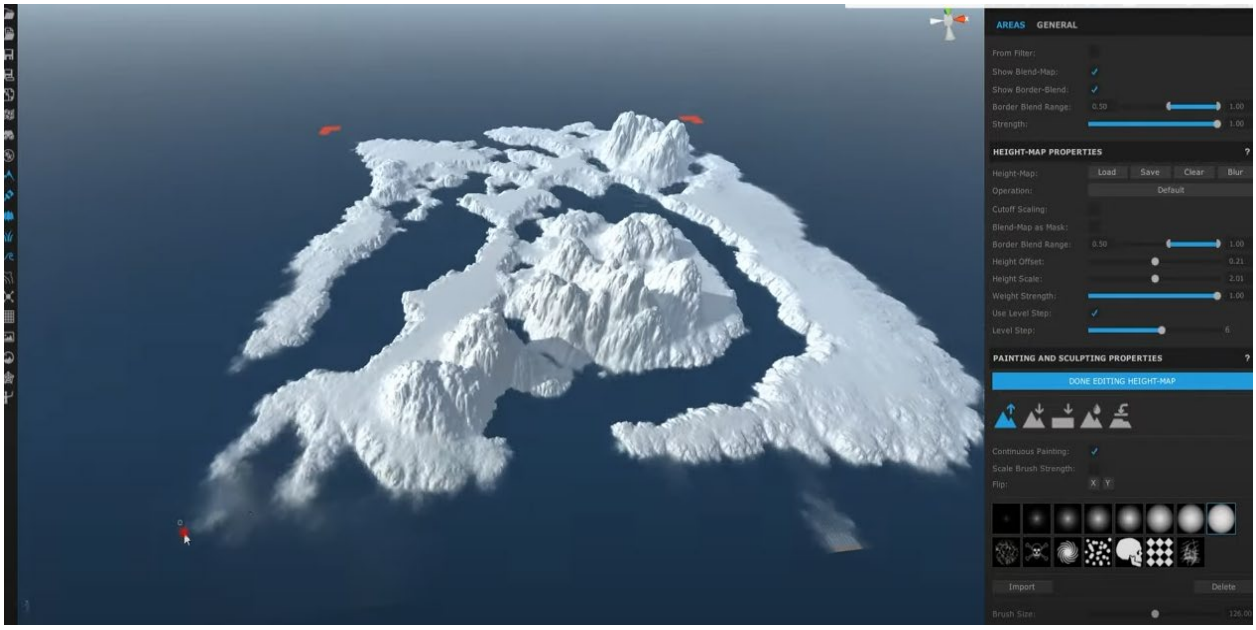


Рисунок 1.18 – Програмне забезпечення World Creator

World Machine – програма, що створює реалістичний 3D-рельєф. World Machine поєднує процедурне створення рельєфу, моделювання природи та інтерактивне редагування для швидкого та легкого створення реалістичного рельєфу [12]. Він дає можливість створення та експортування поля висот високої роздільної здатності, текстури та сітки для гри чи програмного забезпечення для візуалізації.

Природний світ формується не просто математикою. Вітер, вода та інші хаотичні процеси відповідають за більшу частину зовнішнього вигляду нашого світу. World Machine пропонує провідні в галузі інструменти проти ерозії, які допомагають створювати природні ландшафти.

1.5 Специфікація вимог до гри

1 ПРИЗНАЧЕННЯ ТА МЕЖІ ПРОЄКТУ

1.1 Гра розробляється для використання у розважальних цілях.

2 ЗАГАЛЬНИЙ ОПИС

2.1 Сфера застосування: ігрова індустрія та ігрова діяльність

2.2 Характеристики користувачів: гравці, зацікавлені у покрокових стратегіях

3 ФУНКЦІЇ СИСТЕМИ

3.1 Функція системи атаки персонажів

3.1.1 Опис функції: функція, що дає змогу наносити пошкодження

3.1.2 Вхідна і вихідна інформація: здоров’я та сила пошкодження

3.1.3 Функціональні вимоги: персонаж має бути не знищеним

3.2 Функція системи анімацій та ефектів

3.2.1 Опис функції: функція анімацій та доданих до них ефектів персонажів

3.2.2 Вхідна і вихідна інформація: анімації та ефекти

3.2.3 Функціональні вимоги: персонаж має бути не знищеним

3.3 Функція системи спеціальних атак

3.3.1 Опис функції: функція, що дає змогу наносити пошкодження персонажам за допомогою спеціальних атак

3.3.2 Вхідна і вихідна інформація: здоров’я та сила пошкодження

3.3.3 Функціональні вимоги: персонаж має бути не знищеним

3.4 Функція системи збільшення показників персонажів

3.4.1 Опис функції: функція, що підвищує характеристики для персонажів

3.4.2 Вхідна і вихідна інформація: показники персонажа

3.5 Функція системи атаки ворогів

3.5.1 Опис функції: функція, що дає змогу ворогам наносити пошкодження персонажам

3.5.2 Вхідна і вихідна інформація: здоров’я та сила пошкодження

3.6 Функція системи використання спеціальних предметів

3.6.1 Опис функції: функція, для використання спеціальних предметів після битви

3.6.2 Вхідна і вихідна інформація: спеціальні предмети

3.6.3 Функціональні вимоги: персонаж має бути не знищеним

3.7 Функція системи отримання досвіду

3.7.1 Опис функції: функція, що дає змогу отримувати досвід

3.7 Функція системи рівнів

3.7.1 Опис функції: функція, що дає змогу збільшувати рівень та показники

3.7.2 Вхідна і вихідна інформація: досвід, рівень, показники

4 ВИМОГИ ДО ТЕХНІЧНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Процесор: Чотирьох ядерний процесор Intel або AMD з тактовою частотою 2,5 ГГц або вище

4.2 Пам’ять: 8 гігабайт оперативної пам’яті

4.3 Відеокарта: сумісна з DirectX 11 чи DirectX 12

5 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Мова і технологія розробки ПЗ

5.1.1 Мова програмування – Blueprint

5.1.2 Ігровий рушій Unreal Engine 4

5.1.3 Середовище розробки Unreal Editor

5.1.4 Maya – редактор тривимірної графіки

5.1.5 Adobe Photoshop 2020 - графічний редактор

5.1.6 Megascans – бібліотека матеріалів

5.1.7 Substance Painter – середовище текстурування 3д моделей

6 ВИМОГИ ДО ЗОВНІШНІХ ІНТЕРФЕЙСІВ

6.1 Інтерфейс користувача

Інтерфейс гри складається зі сцени на якій містяться персонажі, до кожного персонажа відображаються характеристики, також інтерфейс містить випадаюче меню з вибором покращення характеристик. Під час початку ходу герой має меню вибору ворога для атаки. Після завершення кожної битви меню отримання досвіду. Рівень гри складається з процедурно згенерованої рослинності – дерев та трави; також рівень має налаштоване освітлення з пост-обробкою.

7 ВЛАСТИВОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

7.1 Доступність

Гра безкоштовна та доступна користувачам комп’ютерів.

7.2 Продуктивність

Гра повинна дуже швидко відповідати на кожну дію гравців та мати стабільну кількість FPS за для забезпечення комфортної гри.

Висновки до розділу 1

Досліджено аналоги розроблюваної гри – Heroes of Might and Magic та The Battle for Wesnoth. Виявлено недоліки існуючих ігор, а саме недостатньо розвинена система розвитку персонажів та застаріла графіка. Перевагою гри, що розробляється є створення системи покращення характеристик героїв за рахунок системи досвіду гравця та процедурної генерації. Було досліджено процедурну в сучасних іграх та методи її створення. Розглянуто декілька алгоритмів створення випадкових чисел: шум Перліна, симплексний шум, шум Ворлі, шум значення. Представлено специфікацію вимог до гри. Також визначено обмеження, мову та технології розробки комп’ютерної 3D гри. Виходячи із наведеного аналізу існуючих аналогів та їх недоліків тема роботи є актуальною.

2 МОДЕЛЮВАННЯ ІГРОВОГО ПРОЦЕСУ КОМП’ЮТЕРНОЇ 3D ГРИ

2.1 Діаграми використання та класів

Діаграма варіантів використання (сценаріїв поведінки, прецедентів) є вихідним концептуальним поданням системи в процесі її проєктування і розробки. Дана діаграма складається з акторів, варіантів використання і відносин між ними.

Діаграма варіантів використання представлена на рисунку 2.1

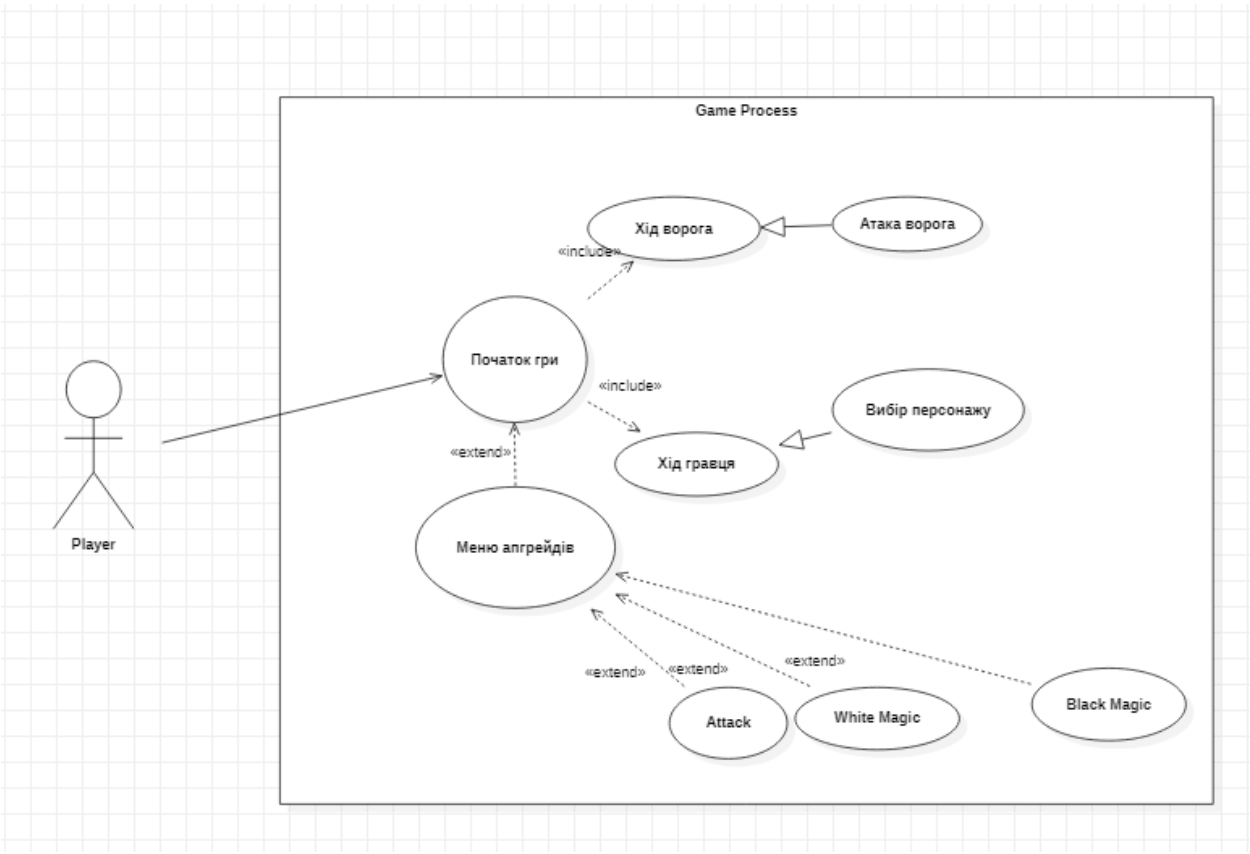


Рисунок 2.1 – Діаграма варіантів використання

Після початку гри відбуваються покрові ходи гравця та його ворога. Хід гравця містить у собі вибір одного з трьох персонажів та можливість покращувати їх характеристики. Гравець обирає одного з трьох ворогів кому нанести пошкодження. Після цього відбувається хід супротивника в якому обирається на якого персонажа йому напасти.

Діаграма класів – це статичне представлення структури моделі. Відображає статичні (декларативні) елементи, Такі як: класи, типи даних, їх зміст та їх відношення. Зазначена діаграма для гри, що розроблюється представлена на рис. 2.2.

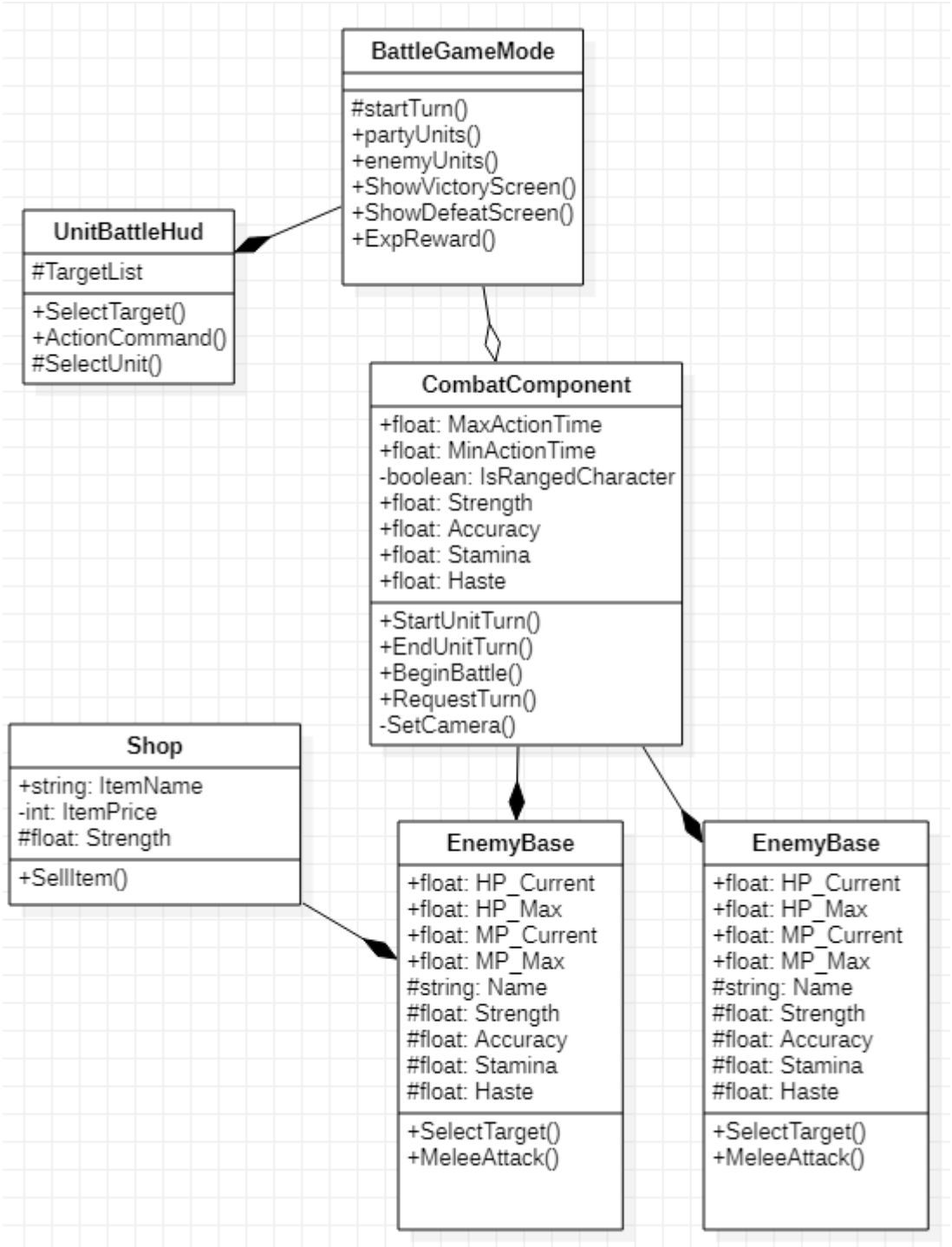


Рисунок 2.2 – Діаграма класів

2.2 Діаграми послідовностей та життєвий цикл об’єктів

Діаграма послідовності – UML-діаграма, на якій для деякого набору об'єктів на єдиній тимчасової осі показаний життєвий цикл об'єкта (створення-діяльність-знищення якоїсь сутності) і взаємодія акторів (дійових осіб) інформаційної системи в рамках прецеденту.

Основними елементами діаграми послідовності є позначення об'єктів (прямокутники з назвами об'єктів), вертикальні «лінії життя», що відображають плин часу, прямокутники, що відображають діяльність об'єкта або виконання ним певної функції (прямокутники над пунктирною «лінії життя»), і стрілки, що показують обмін сигналами або повідомленнями між об'єктами.

На рисунку 2.3 показано діаграму послідовності покрокової системи гри. Після початку гри першим ходить гравець, після нього ходить ворог.

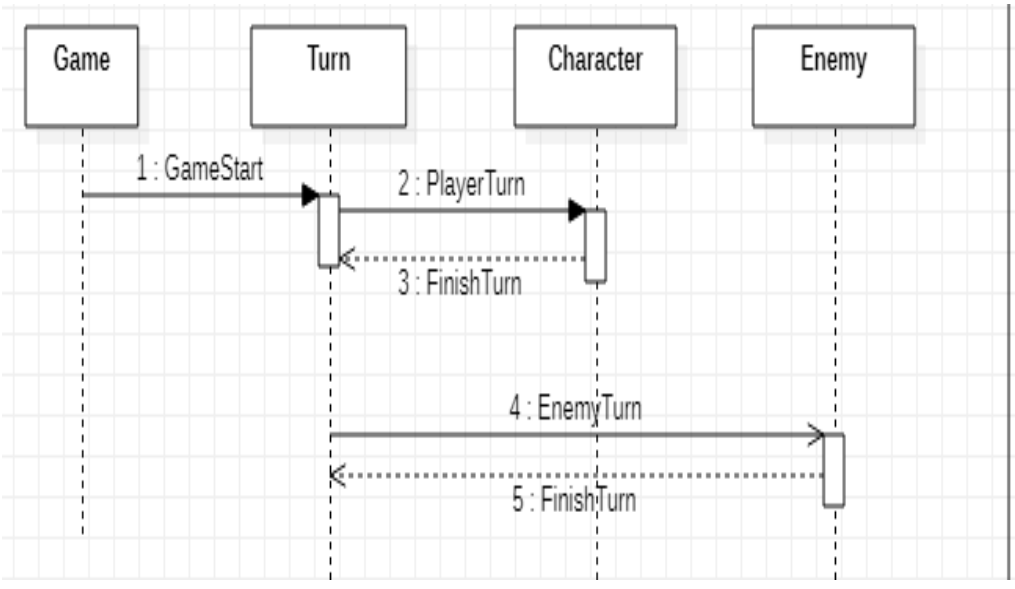


Рисунок 2.3 – Діаграма послідовності покрокової системи

На рисунку 2.4 діаграма послідовності для атаки героя. Гравець обирає тип атаки: рукопашна атака чи магічна. Після вибору типу атаки гравець

обирає ворога – ціль. Під час атаки відбувається в залежності від типу спеціальна анімація. При нанесенні пошкоджень магічною атакою відбувається зменшення кількості мани.

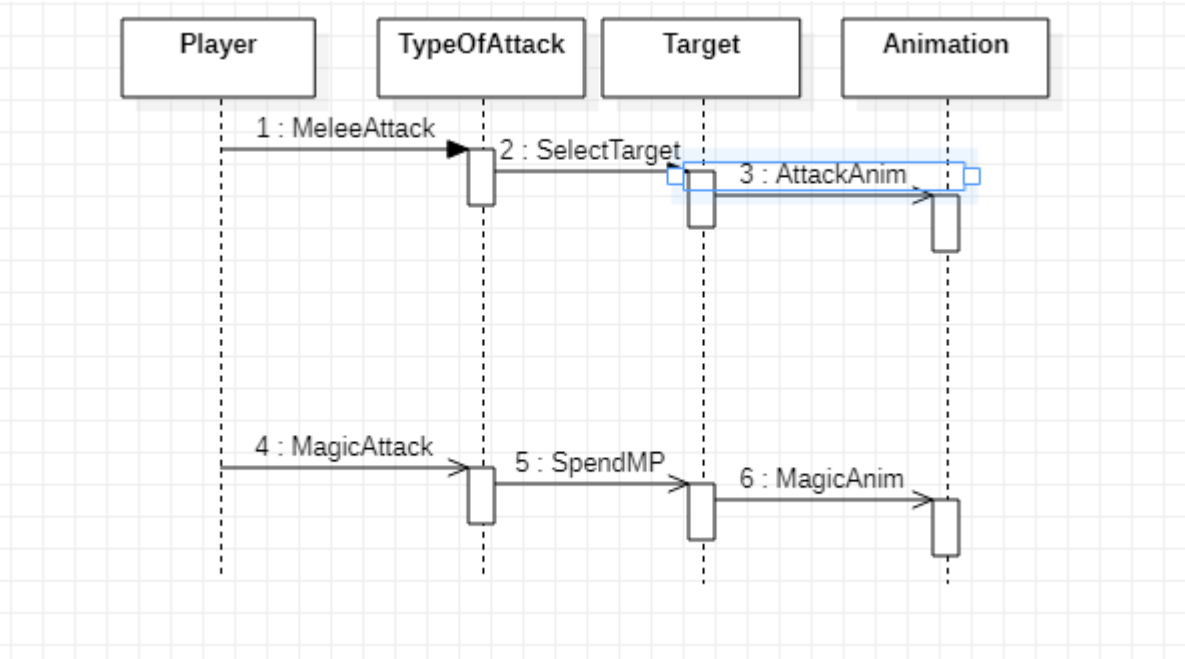


Рисунок 2.4 – Діаграма послідовності для атаки героїв

На рисунку 2.5 показано діаграму послідовності для атаки ворога, Після ходу гравця, відбувається хід противника при якому випадково обирається ціль для атаки. Після атаки на персонажа, у нього відбувається зменьшення здоров’я. Після цього закінчується хід ворога.

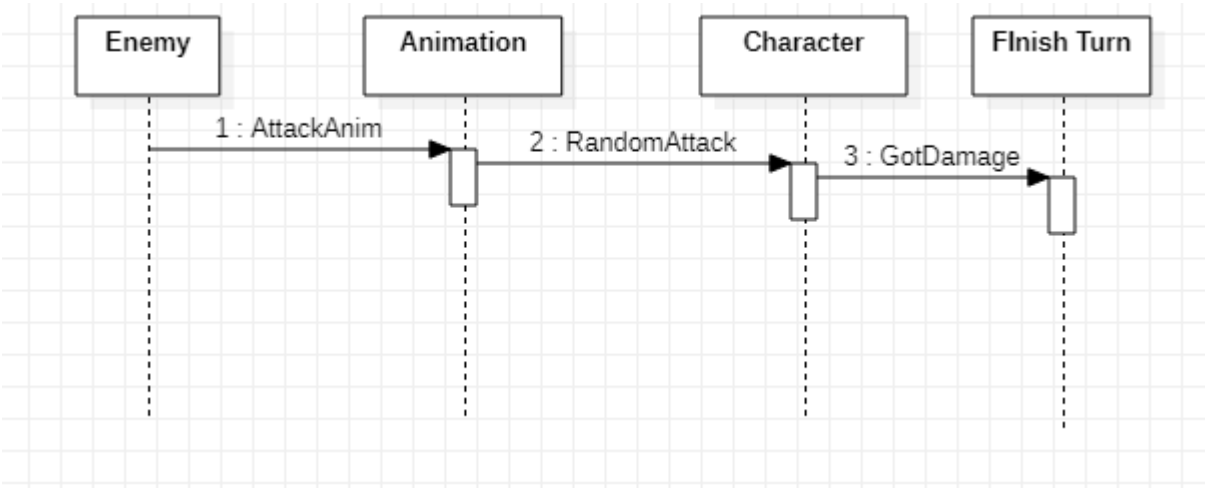


Рисунок 2.5 – Діаграма послідовності для атаки ворога

На рисунку 2.6 показано діаграму послідовності для атаки магією. Гравець обирає персонажа яким буде ходити та може атакувати магією. В грі є магія двох типів: біла і чорна. В залежності від обраного героя показники білої і чорної магії різні. Після цього обирається ціль яку буде атакувати персонаж під час ходу.

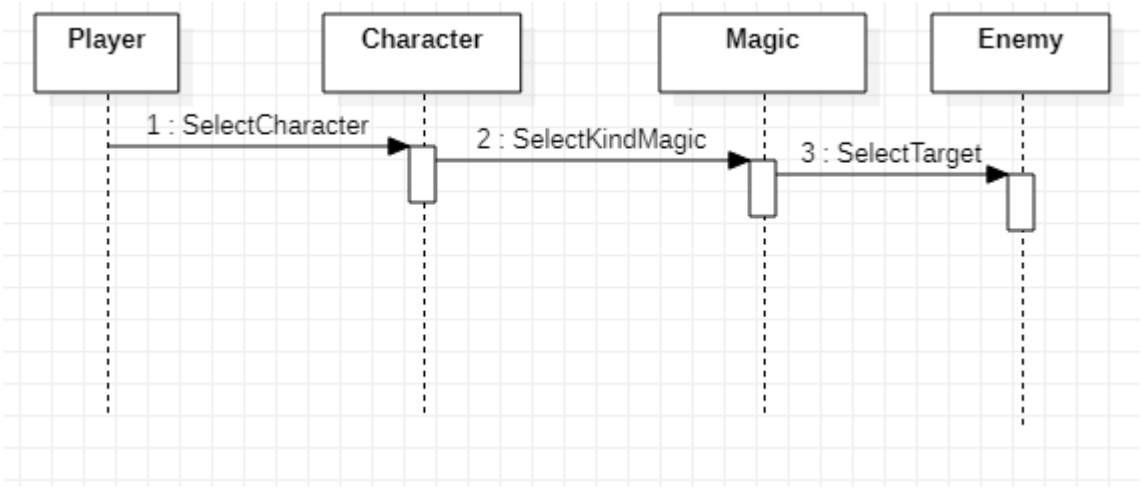


Рисунок 2.6 – Діаграма послідовності для атаки магією

На рисунку 2.7 показано діаграму послідовності для процесу використання магії. Після того як гравець її використовує відбувається анімація та зменшення шкали мани.

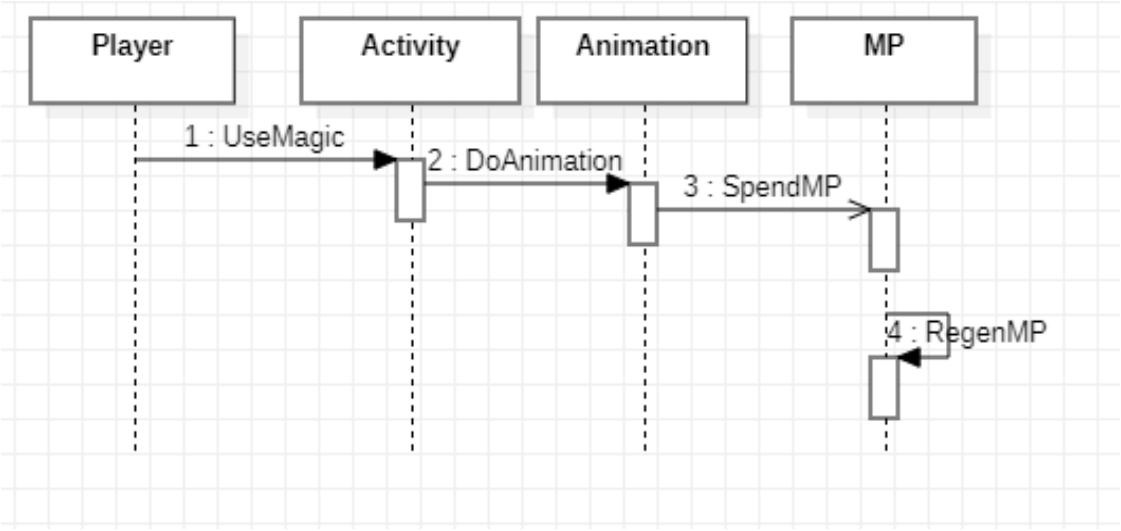


Рисунок 2.7 – Діаграма послідовності для процесу використання магії

На рисунку 2.8 показано діаграму послідовності для меню предметів. Гравець під час свого ходу може відкрити меню з предметами, які покращають

характеристики персонажа. Після цього купити предмет витративши на нього зароблені монети.

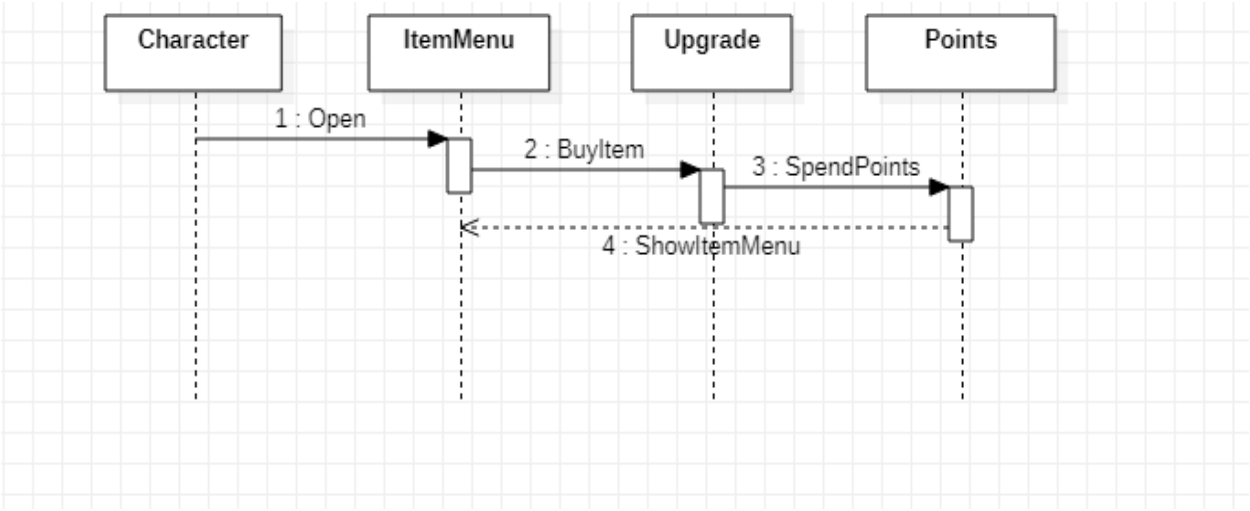


Рисунок 2.8 – Діаграма послідовності для меню предметів

На рисунку 2.9 діаграма послідовності при процесі завершення гри. Якщо гравець знижує всіх ворогів, то відбувається перемога та відкривається вікно перемоги, а якщо всі герої гравця буде знищено - вікно поразки.

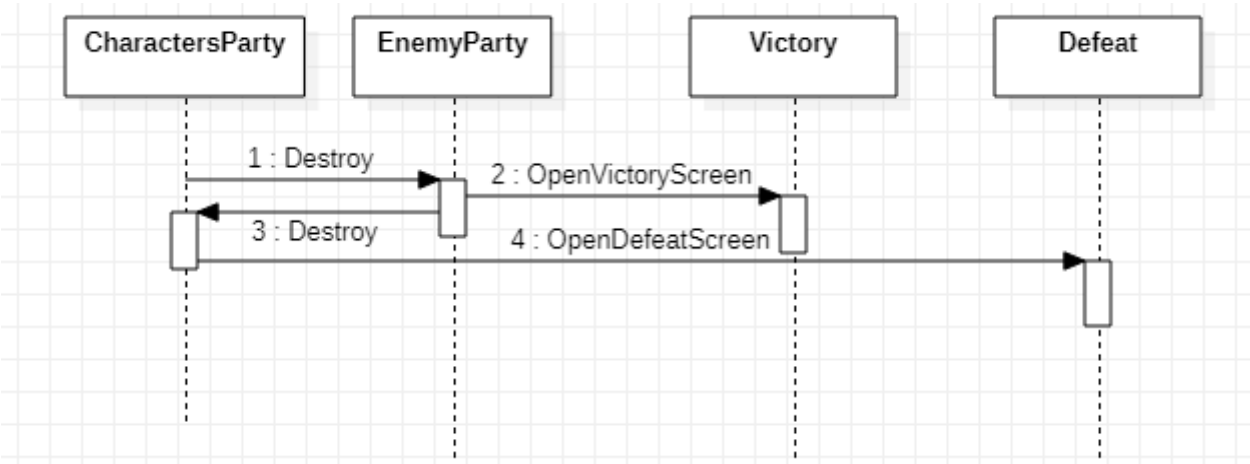


Рисунок 2.9 – Діаграма послідовності для процесу завершення гри

На рисунку 2.10 показано діаграму послідовності для отримання досвіду. Після того, як гравець знищить всіх ворогів відбувається перемога та кожен персонаж отримує досвід та збільшення рівню, який впливає на збільшення характеристик героїв.

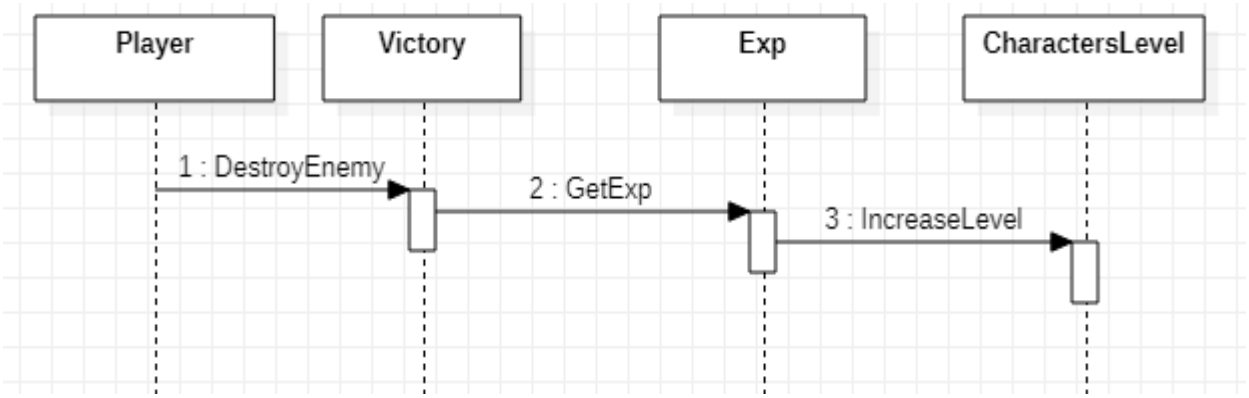


Рисунок 2.10 – Діаграма послідовності отримання досвіду

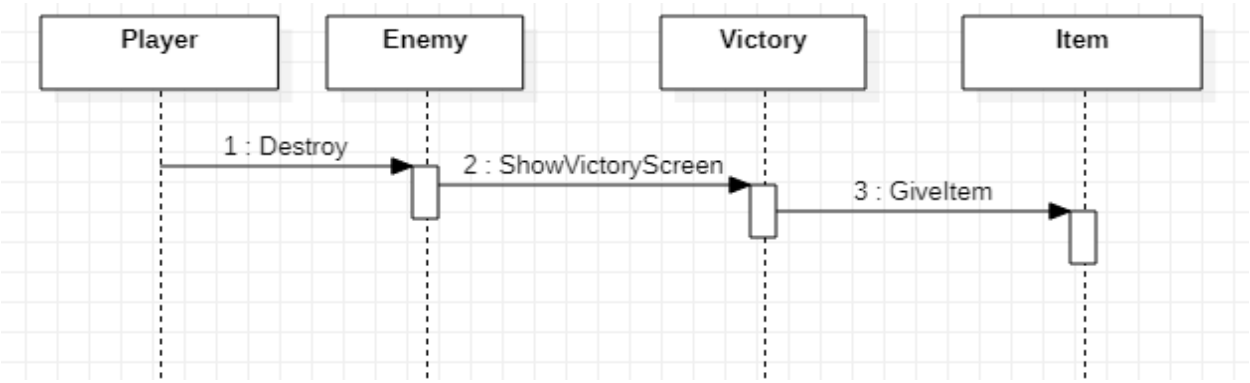


Рисунок 2.11 – Діаграма послідовності отримання предметів

Після перемоги над ворогом, окрім досвіду та підвищення рівня, герой отримує предмет, який також буде підвищувати характеристики героя.

2.3 Діаграми переходів та діяльності

Діаграма переходів станів (State Transition Diagrams, STD) – це UML діаграма, що демонструє поведінку розроблюваної програмної системи при отриманні керуючих впливів (ззовні). За допомогою діаграм переходів станів можна моделювати подальше функціонування системи на основі її попереднього і поточного функціонування.

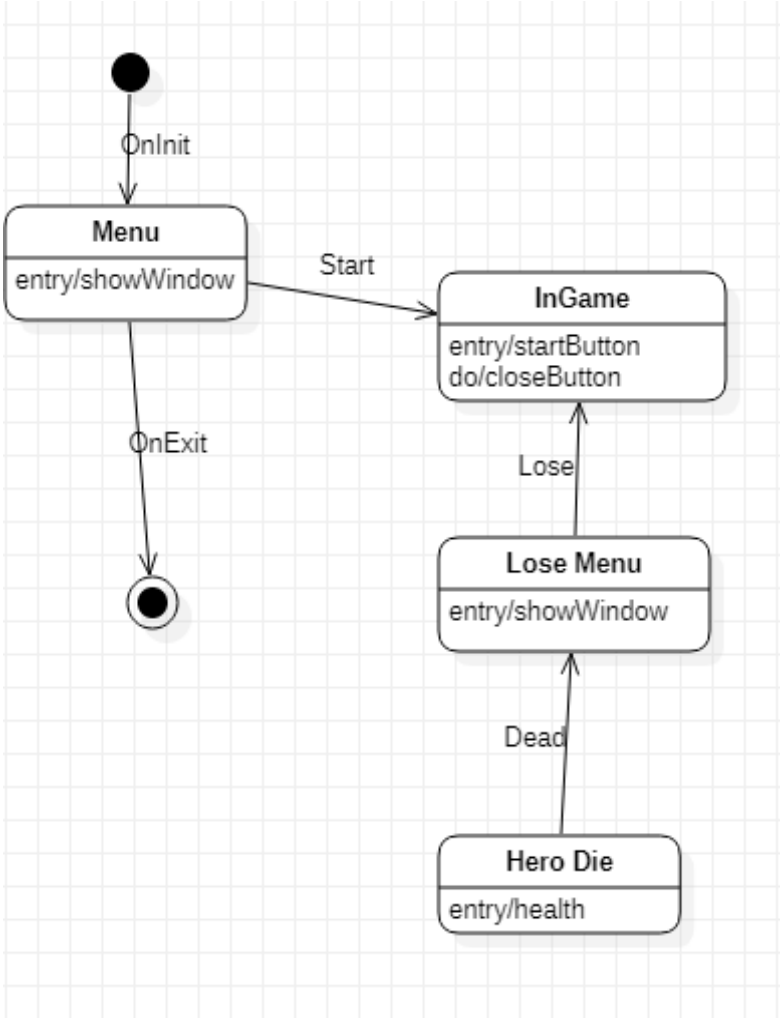


Рисунок 2.12 – STD поразки

На діаграмі переходів станів (рис 2.12) показано процес гри, в якому після смерті героя відкривається меню програшу зі статистикою за матч і гравець потрапляє на сторінку початку гри [4].

На наступній STD (рис. 2.13) показано процес гри. Після старту обирається герой для ходу, який наносить пошкодження команді противника, після знищення ворога герою дається досвід для збільшення рівня та гроші за які можна купувати предмети в магазині. При знищенні всієї команди ворога, відкривається вікно перемоги, а при втрати всієї команди героїв – поразка.

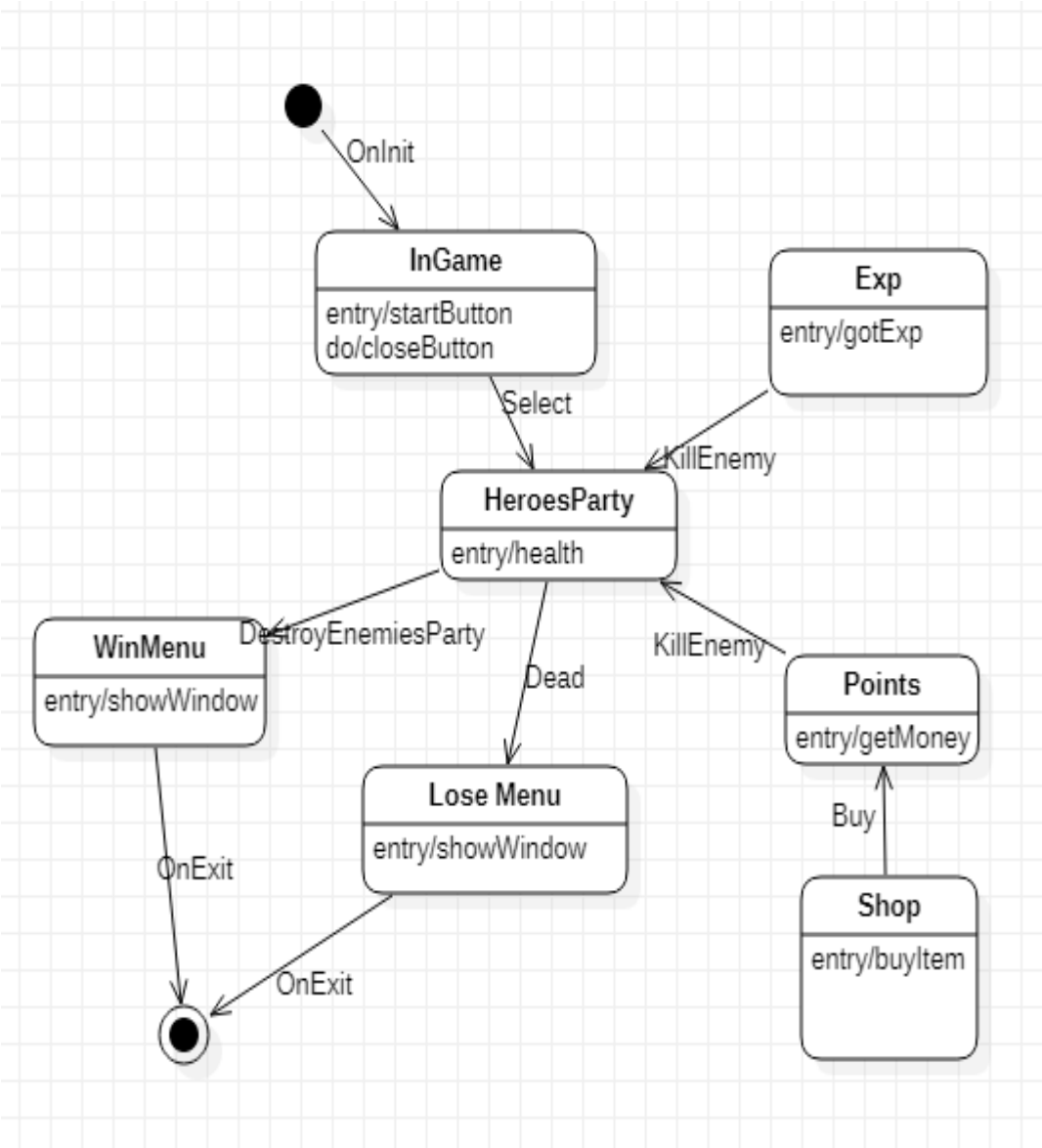


Рисунок 2.13 – STD ігрового процесу

На рисунку 2.14 діаграма переходів станів для героя гри. Для всіх персонажів у грі для кожної дії існує своя анімація. Після смерті команди відбувається поразка і гра закінчується.

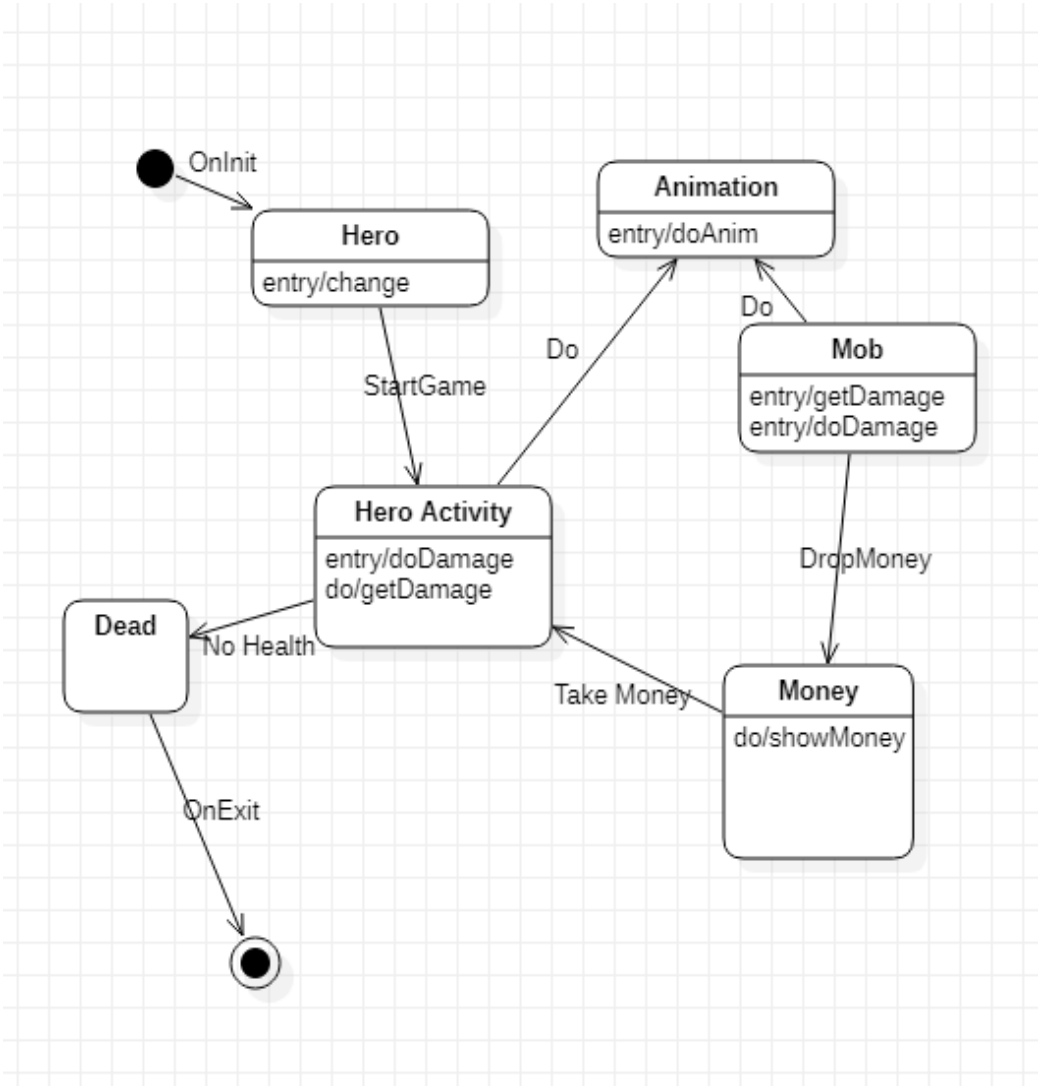


Рисунок 2.14 – STD для героя

Діаграма діяльності – це UML-діаграма, на якій показані дії, специфікація виконуваної поведінки у вигляді координованого послідовного і паралельного виконання підлеглих елементів - вкладених видів діяльності і окремих дій, з'єднаних між собою потоками, які йдуть від виходів одного вузла до входів іншого. Діаграми діяльності складаються з обмеженої кількості фігур, з'єднаних стрілками (рис 2.15).

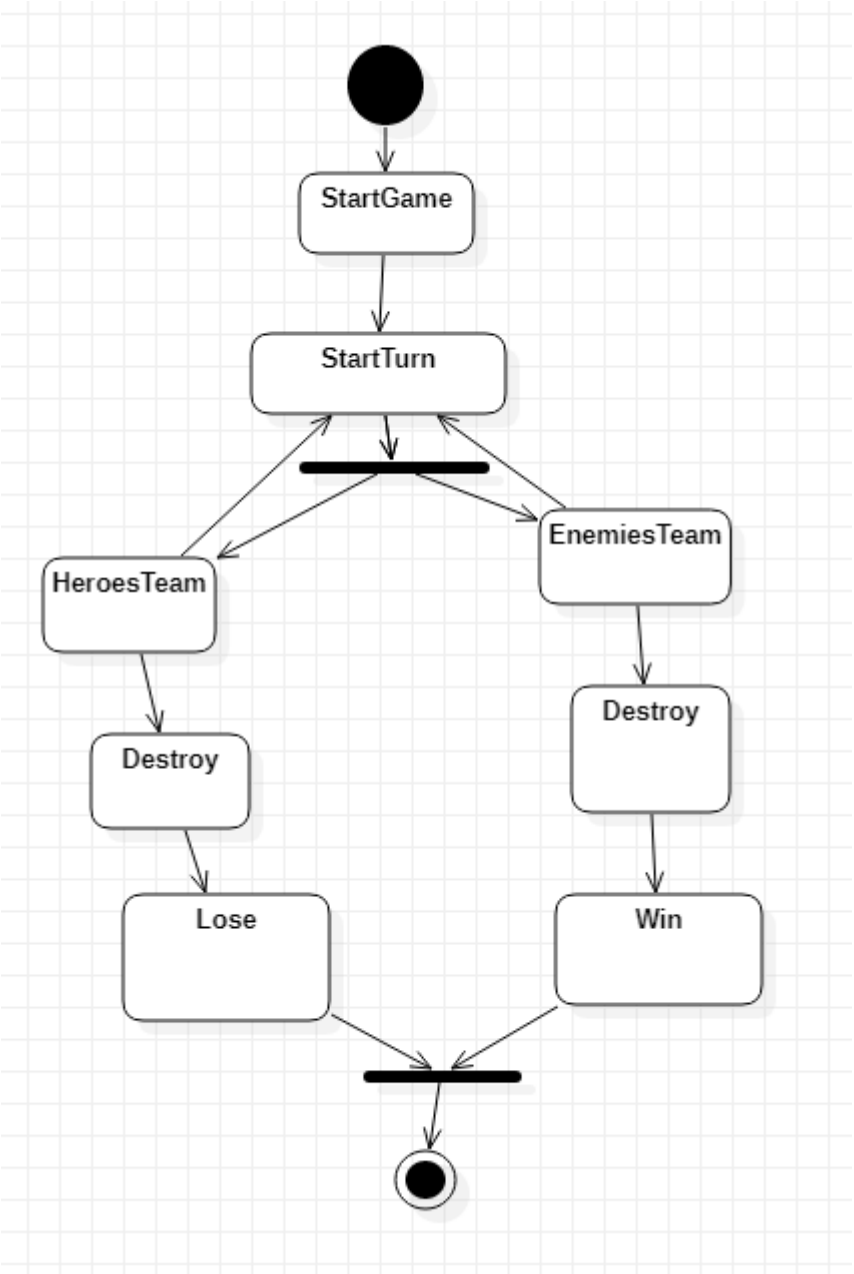


Рисунок 2.15 – Діаграма діяльності для гри

2.4 Розробка макетів інтерфейсу

Для створення макетів гри було використано програмне забезпечення Adobe Photoshop (PS). PS – це професійний графічний редактор, розроблений компанією Adobe Systems і призначений для створення і редагування цифрових зображень, обробки фотографій. Програма містить у собі набір інструментів для малювання, роботи з шарами, трансформації та редагування зображень, підтримує автоматизацію операцій за допомогою сценаріїв. При

необхідності є можливість додавання в бібліотеки Photoshop додаткових фільтрів, наборів квітів, кистей і стилів.

Перевагою Photoshop є те, що в програмі можна створювати елементи дизайну: логотипи, кнопки, іконки, банери для реклами, візитні картки, вітальні листівки.

Рисунок 2.16 – Макет інтерфейсу гравця

Макет інтерфейсу гравця (рис. 2.16) складається з двох вікон. У першому вікні випадаюче меню, яке містить ім’я вибраного персонажа, меню з атакою, магією, техніками та спелами. У другому вікні міститься інформацію про всю команду героїв: ім’я кожного, значення максимального та мінімального здоров’я та максимальне та мінімальне для мани, шкала мани.

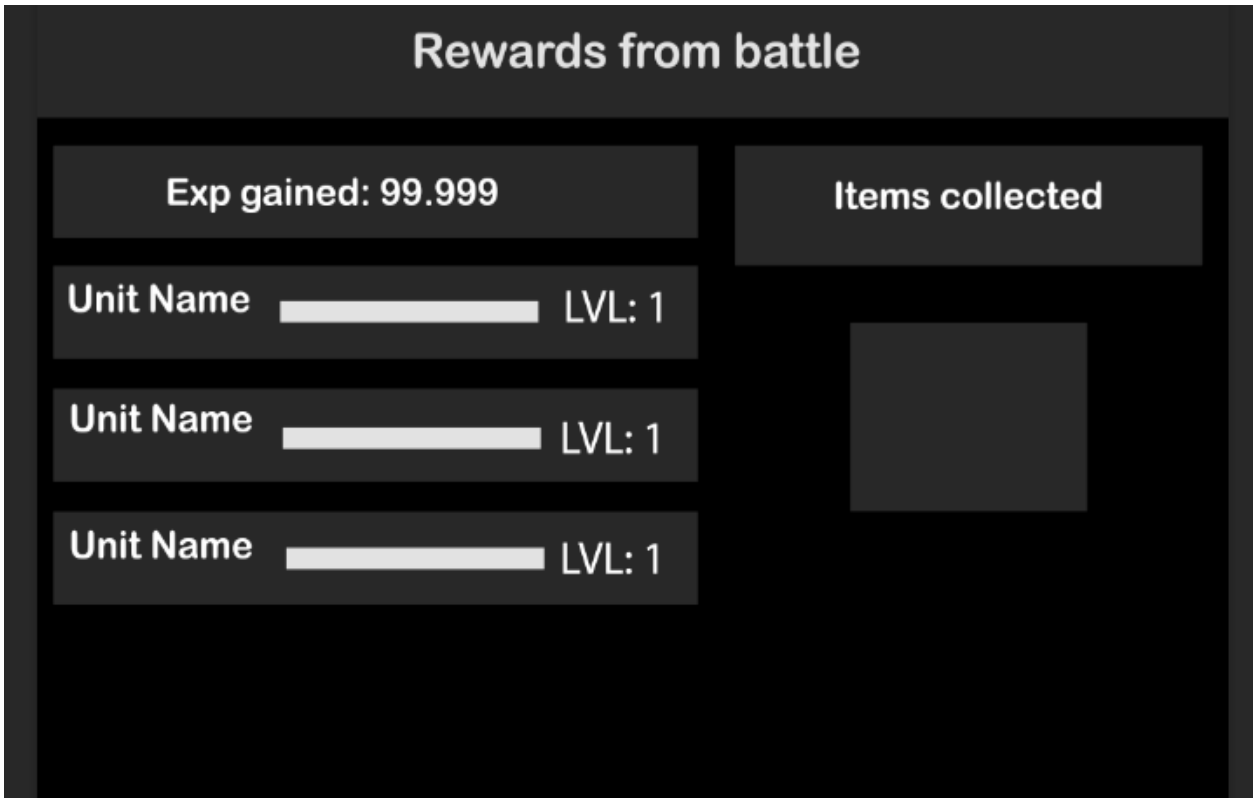


Рисунок 2.17 – Макет вікна перемоги

На макеті вікна перемоги (рис. 2.17) зображено отримані нагороди після закінчення бою перемогою. Після кожної перемоги для кожного героя дається досвід, який збільшує рівень героя. Також вікно перемоги містить предмети, які можна отримувати після закінчення бою.

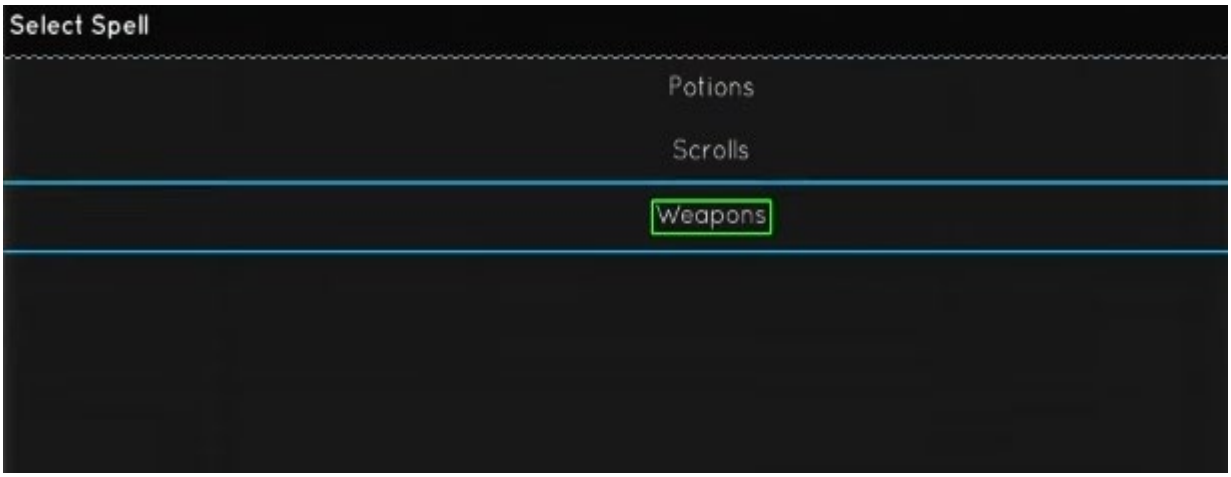


Рисунок 2.18 – Макет меню предметів

На рисунку 2.19 показано макет для предметів, які можна купувати або отримувати після перемоги у грі. Макет містить у собі 3 категорії:

– зброя;

– зілля;
– сувої.

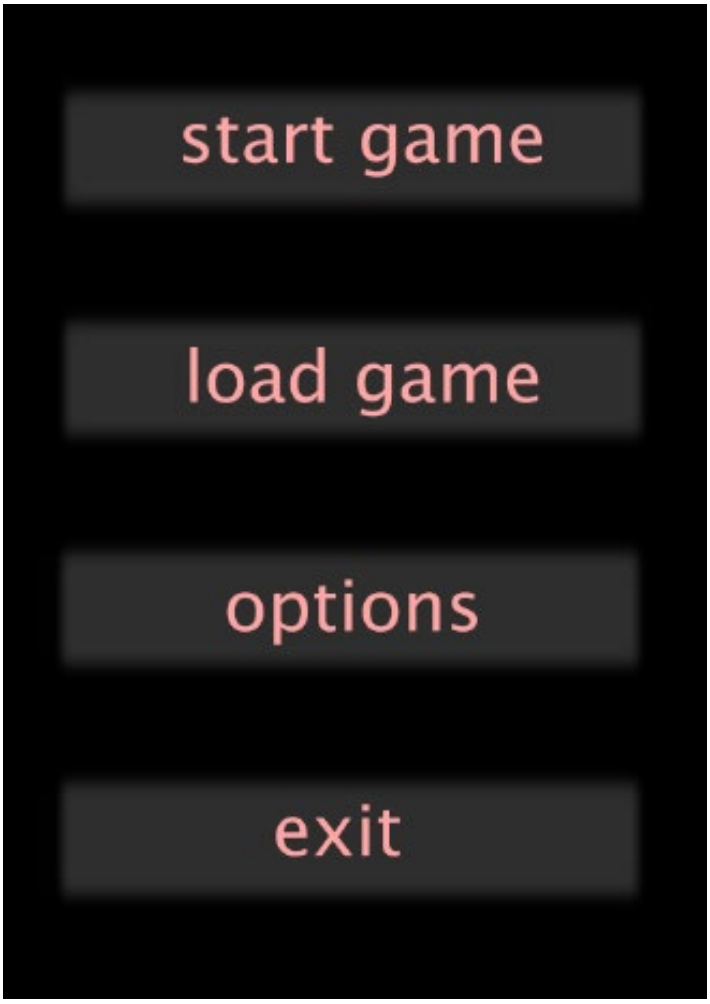


Рисунок 2.19 – Макет меню гри

Макет гри (рис. 2.19) містить у собі такі кнопки:

– початок нової гри;
– завантажити збережену гру;
– налаштування;
– вихід з гри.

Висновки до розділу 2

Змодельовано процес розробки комп’ютерної гри. Представлені UML-моделі включають в себе діаграми: класів, використання, послідовності, переходів та діяльності. Діаграма класів описує основні класи, їх функції та відношення. Діаграма використання – основний процес гри покрокової стратегії. Діаграми послідовності показують життєвий цикл об’єктів та взаємодії акторів. Розроблено діаграми переходів станів для процесу поразки, процесу самої гри та для самого героя. За рахунок діаграм переходів станів можна моделювати подальше функціонування системи на основі її попереднього і поточного функціонування. Також створено діаграму діяльності для процесу гри в покроковій стратегії.

В Adobe Photoshop розроблено макети для інтерфейсу гравця під час самої гри, макет вікна перемоги, макет вибору предметів, макет меню гри. Макети було розроблено для розуміння розроблюваних функцій та механік гри.

3 РЕАЛІЗАЦІЯ ПРОЦЕДУРНОЇ ГЕНЕРАЦІЇ

3.1 Створення 3D моделей рослинності

Для наповнення ігрового середовища за рахунок процедурної генерації створено декілька видів трави та дерев. Спочатку змодельовано високополігональні об’єкти в Maya 3D (рис. 3.1).

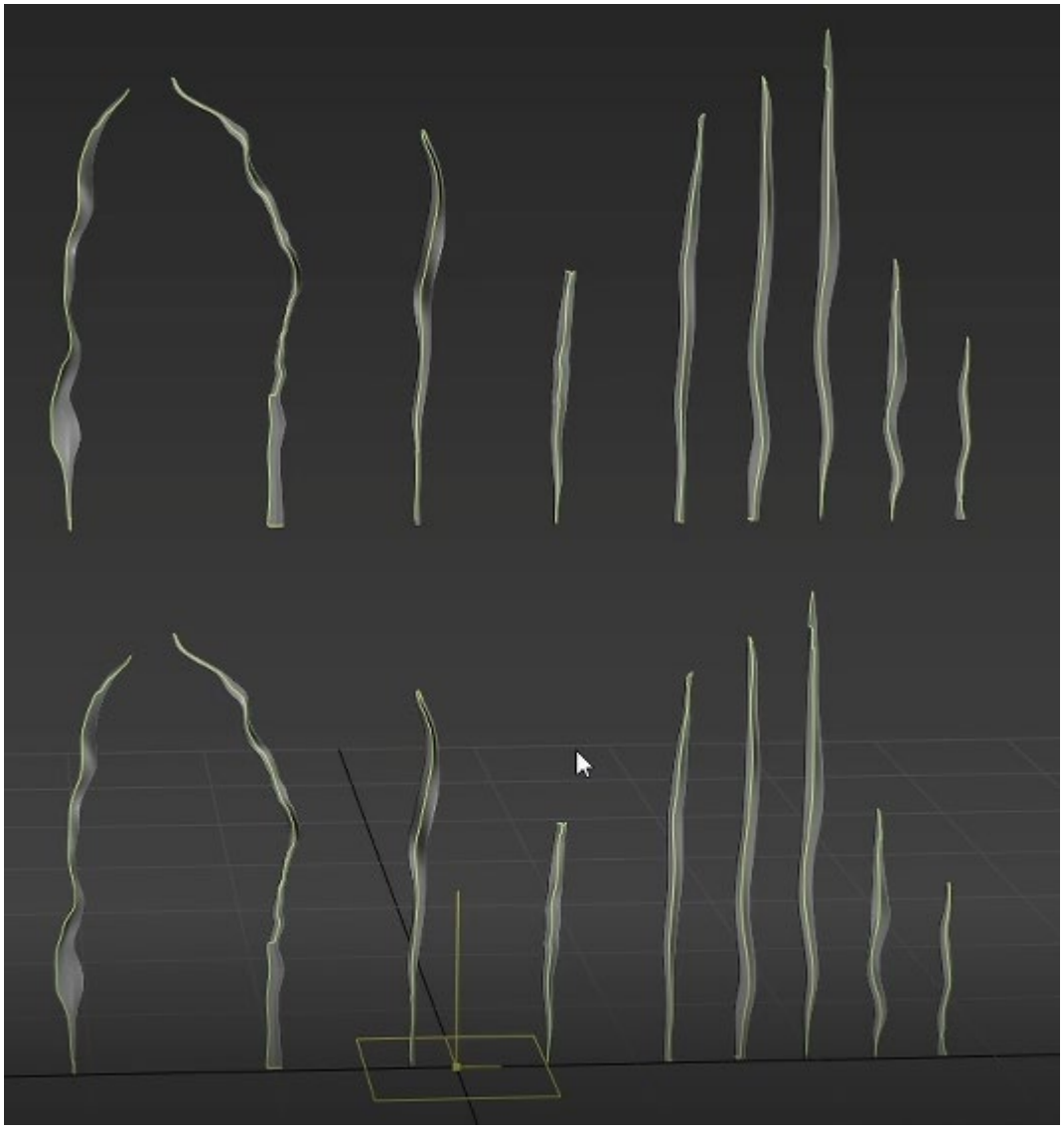


Рисунок 3.1 – Моделювання трави

Для того, щоб зробити текстури для трави створено UV-розгортку (рис. 3.2). UV розгортка – процес в 3D моделюванні, який полягає в накладанні двовимірного зображення на тривимірну модель. Літерами U і V позначають осі координат площини розгортки, оскільки літери X, Y і Z використовуються для позначення просторових координат.

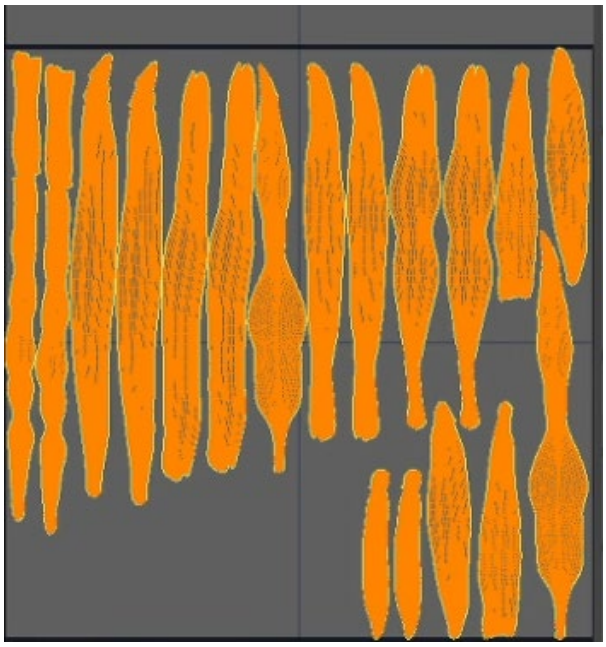


Рисунок 3.2 – UV-розгортка трави

Після UV-розгортки трави в Substance Painter 3D зроблено текстури трави двох відтінків (рис. 3.3). 3D-текстурування – це процес додавання текстури до 3D-об'єкта. Він включає створення текстур з фотографії або самостійно. Для створення градієнту було використано фільтр – linear gradient та планарну проєкцію. Для того, щоб зробити кольоровий перехід по лінію краю моделі використано генератор – metal edge.

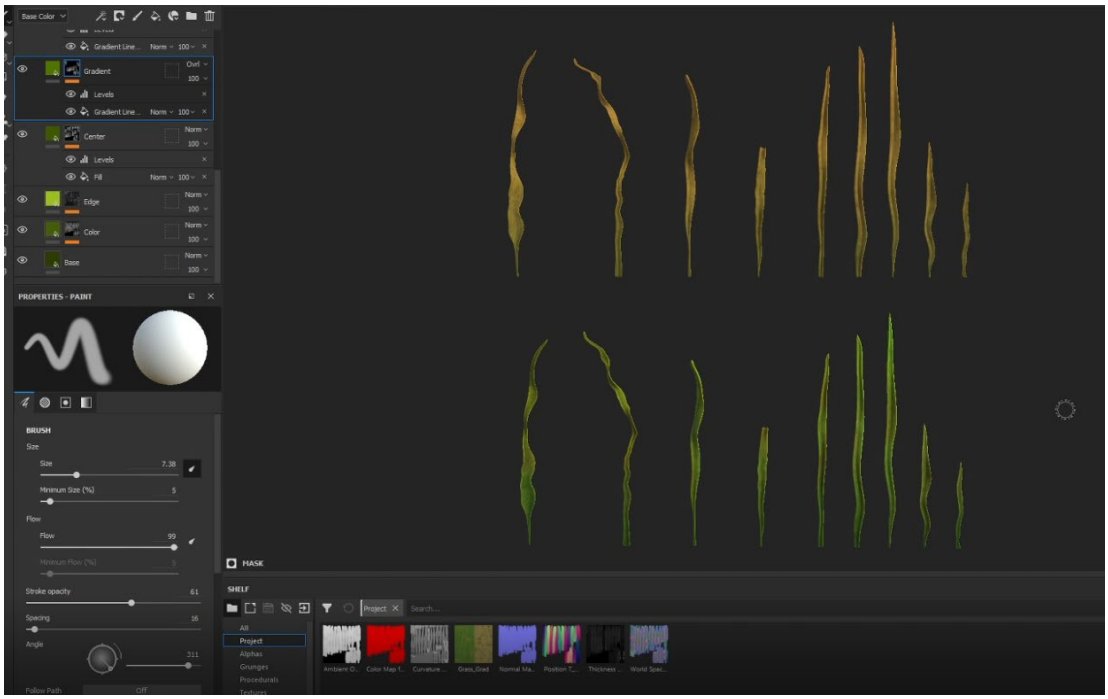


Рисунок 3.3 – Текстурування 3D моделі

Після цього створено набір з травинок, в який розміщуються створені моделі. Після цього для оптимізації та зменшення полігонів в ігровому середовищі створюється атлас – набір травинок та плейн на який в подальшому трава буде запікатись (рис. 3.4).

Рисунок 3.4 – Атлас трави

Після того як моделі було запечено на плейн, з декількох елементів текстури зібрано об’єкт (рис. 3.5) та створено декілька різновидів (рис. 3.6)

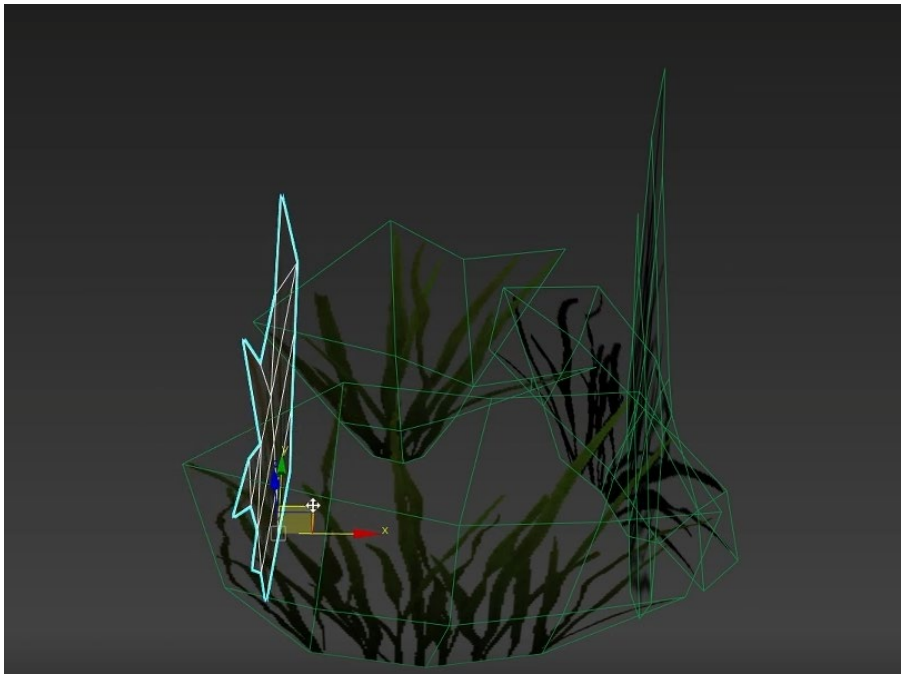


Рисунок 3.5 – Збирання об’єкту трави

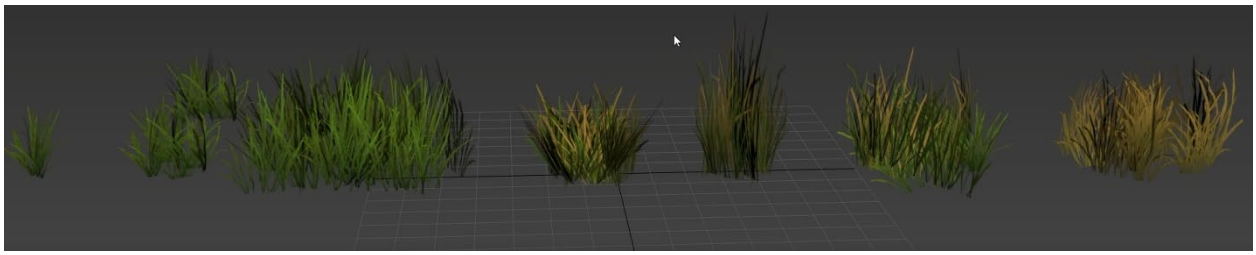


Рисунок 3.6 – Різновиди трави

З атласу трави було обрізано текстуровані зображення трави, що запікались з високополігональних об’єктів і після цього створено єдині об’єкти які мають карти за допомогою яких в Unreal Engine за допомогою інструменту, що додає вітер, будуть колихатись в залежності від налаштованих параметрів.

3.2 Створення процедурної генерації рослинності

У рушії Unreal Engine 4 створено ландшафт для ігрового рівня (рис. 3.7). Кожний компонент секції складається з однієї секції. Розмір ландшафту – 63 квади на одну секцію. Номер компоненту встановлює розмір ландшафту разом із розміром розділення. Це значення обмежене 32 на 32, оскільки кожен компонент має пов’язану з ним вартість навантаження на ЦП. Перевищення цього обмеження може призвести до проблем із продуктивністю ландшафту. Після цього за допомогою інструментів скульптингу на згенерованому ландшафті додано височини та низовини.

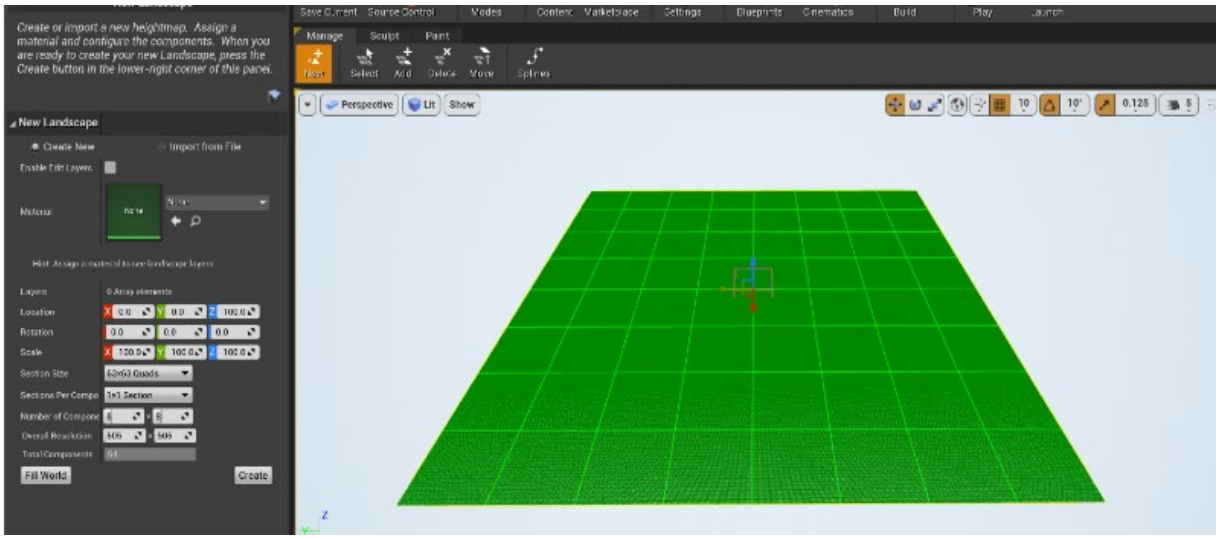


Рисунок 3.7 – Створення ландшафту

Для матеріалу ландшафту використано текстури з бібліотеки Megascans від Quixel. За допомогою їх поєднання створено основний матеріал землі (рис. 3.8). Матеріал землі складається з таких карт: Base Color, Roughness, Normal, Ambient Occlusion. Base Color містить в собі інформацію про колір, Roughness – жорсткість, Normal – не додаючи додаткові полігони за рахунок падіння освітлення відобразити деталі матеріалу, Ambient Occlusion відповідає за тіні між об’єктами.

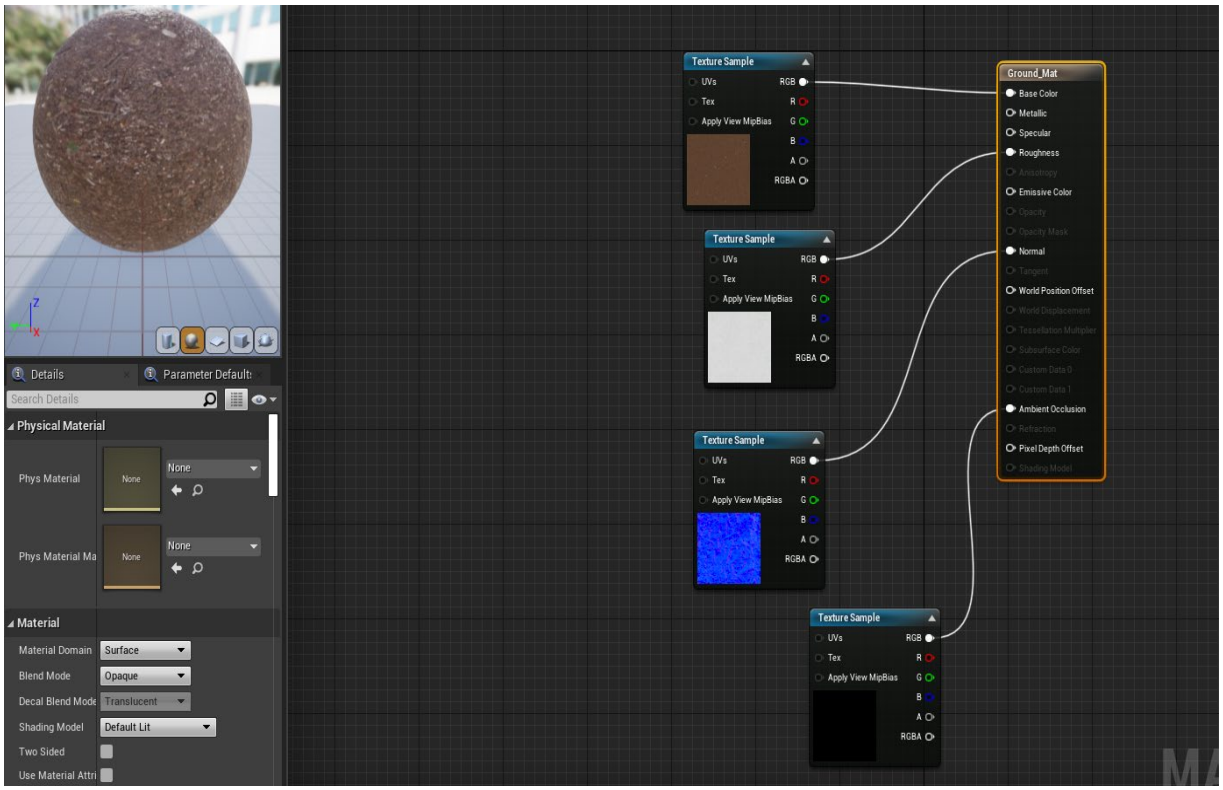


Рисунок 3.8 – Матеріал ландшафту

Для процедурної генерації рослинності на ландшафті було створено Procedural Foliage Spawner (PFS). Він відповідає за генерацію доданих у нього об’єктів на доданій ділянці в ігровому рівні (рис. 3.9).

Для того щоб додати об’єкти до PFS створюється Static Mesh Foliage (рис. 3.10. Це найкраще підходить для неруйнівної рослинності. Static Mesh Foliage автоматично групуються в пакети, які відображаються за допомогою апаратного створення екземплярів, де багато екземплярів можна відобразити лише за допомогою одного виклику малювання, у той час як Actor Foliage

коштує таку саму вартість візуалізації, що й розміщення звичайних акторів у сцені.

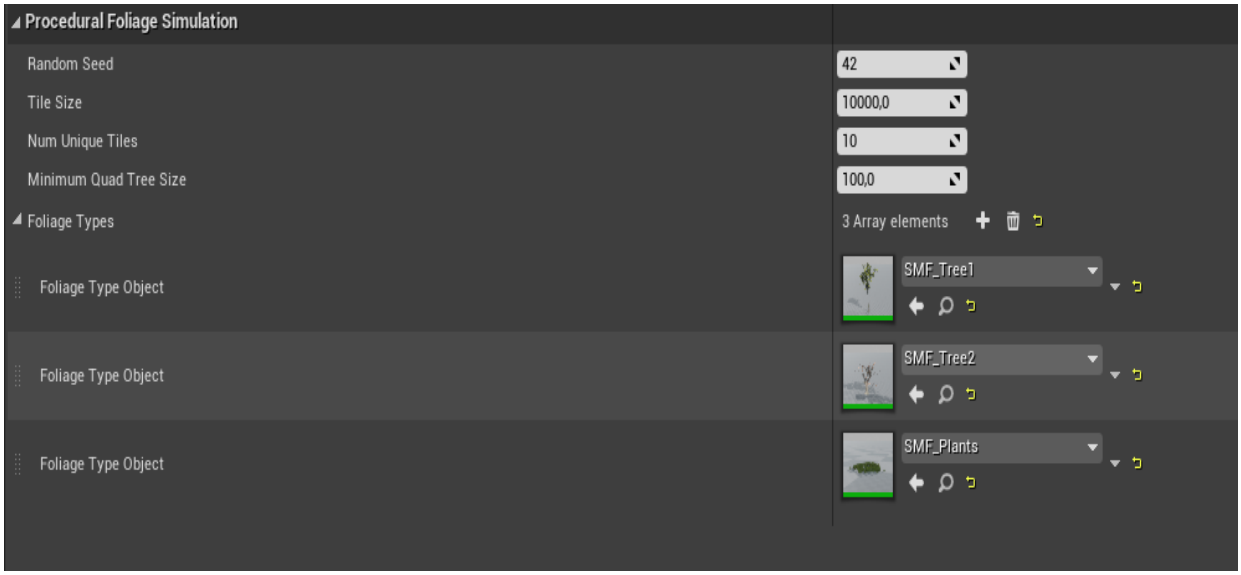


Рисунок 3.9 – Procedural Foliage Spawner

Після створення налаштовано параметри, а саме в поле Mesh додано 3D об’єкти дерев та трави. В параметрі Align Max Angel – 5, для того, щоб на низовинах чи височинах дерева правильно розташовувались в незалежності від кута ландшафту. Colission Radius та Shade Radius відповідають за радіус генерації об'єктів між собою. Якщо об’єкт попадає в радіус іншого, то він видаляється. Num Steps відповідає за кількість згенерованих об’єктів. Параметри Procedural Scale відповідають за розміри генерації об’єктів. Мінімальне значення відповідає за молоді дерева, а середні та максимальні за старі.

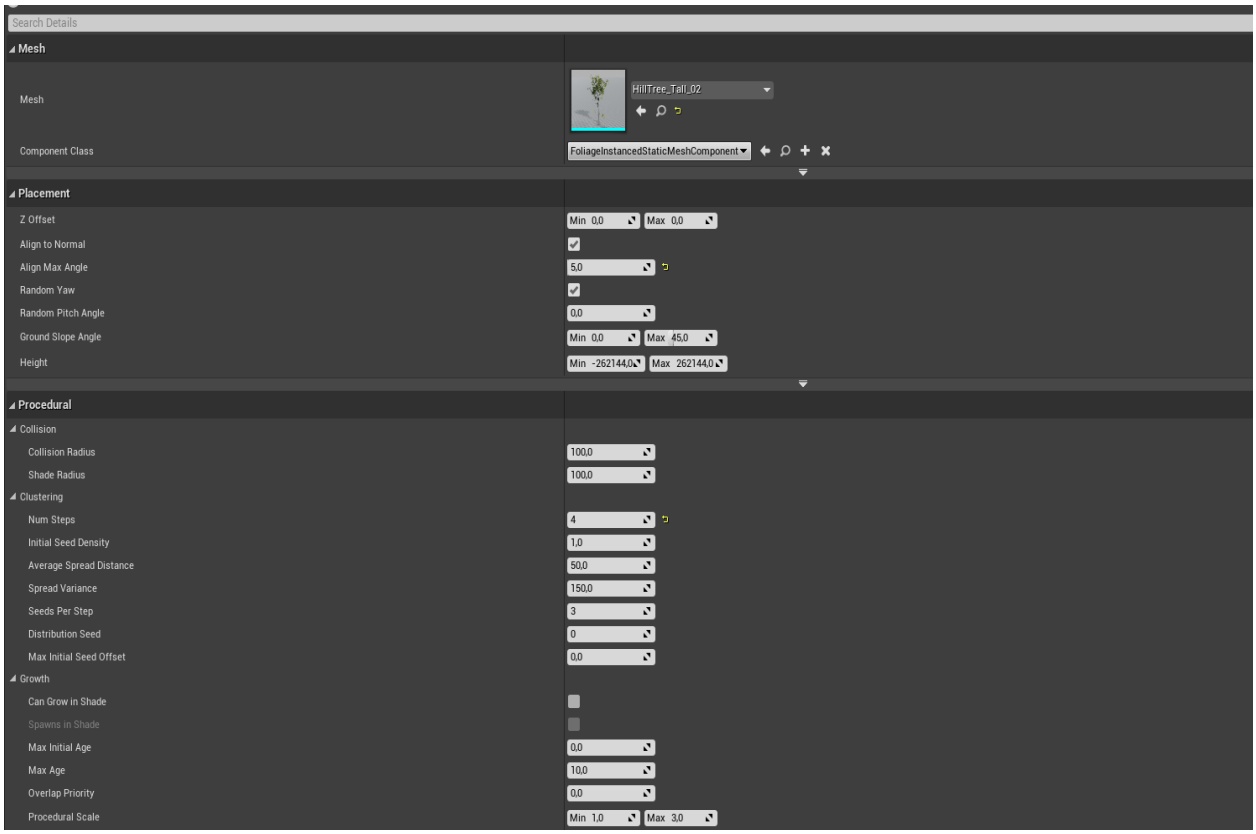


Рисунок 3.10 – Static Mesh Foliage

Після того, як створено за допомогою процедурної генерації, рослинність було відредаговано вручну за допомогою інструментів Foliage в Unreal Engine. Інструмент Foliage (рис. 3.11) дає змогу швидко додавати та редагувати рослинність на ігровому рівні [13]. Додані об’єкти сприймаються, як один, що дає змогу оптимізувати гру та не грузити кожен однаковий об’єкт на сцені.

Кожен доданий об’єкт має так само поля для налаштування (рис. 3.12). А саме, їх розміщення, масштабування чи кут обертання. Доступ до властивостей можна отримати, вибравши потрібний об’єкт і налаштувавши властивості.

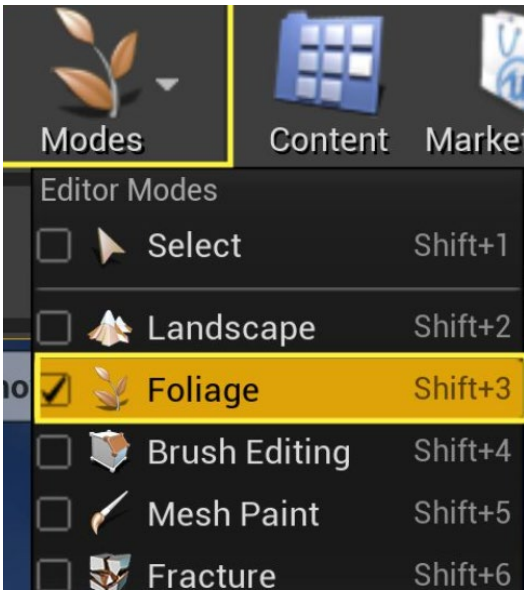


Рисунок 3.11 – Інструмент Foliage

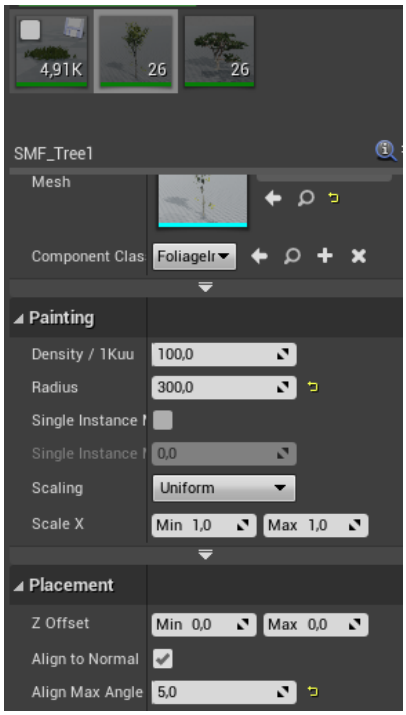


Рисунок 3.12 – Налаштування об’єктів

Після налаштування параметрів об’єктів у Foliage, використано декілька інструментів: Paint, Erase, Single та Select. Інструмент Paint є інструментом за замовчуванням у режимі редагування Foliage. Його використано для малювання об’єктів на рівні. Коли режим редагування активний, прозорий сферичний пензель малюється на рівні, який вказує, де працюватиме пензель із вибраним об’єктом. Інструмент Paint має такі налаштування: розмір кисті та

щільність, з якою додаються об’єкти під час використання лівої кнопки миші. Це значення від 0 до 1, де 1 малює об’єкти з максимальною щільністю, зазначеною у властивостях об’єкта. Якщо щільність об’єктів у пензлі вже більша за це, об’єкти не додаватимуться.

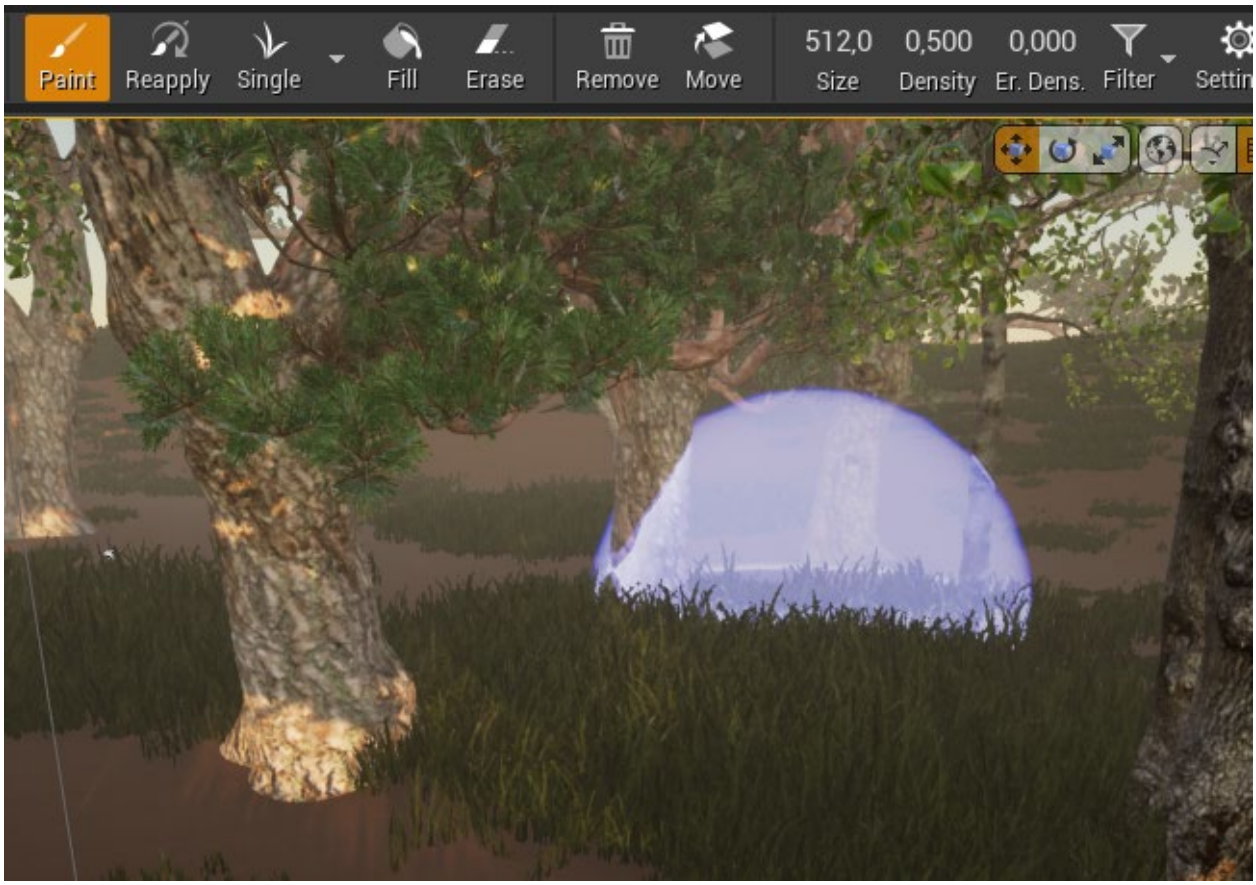


Рисунок 3.13 – Інструмент Paint

Коли інструмент Select активний, обираються об'єкти, клацнувши на них. Можна вибрати кілька об'єктів Foliage одночасно, утримуючи натиснутою кнопку Ctrl і клацаючи об’єкти Foliage на рівні. За допомогою різних кнопок вибору можна вибрати всі об'єкти Foliage, вибрати будь-які недійсні або очистити поточний вибір.

Інструмент Single (рис. 3.14) використано, щоб розмістити один об’єкт на рівні. Це використано, щоб заповнити прогалини та створити особливі особливості рельєфу. Якщо вибрано більше ніж один об'єкт, інструмент розмістить об'єкти один над одним в одному місці.

Рисунок 3.14 – Інструмент Single

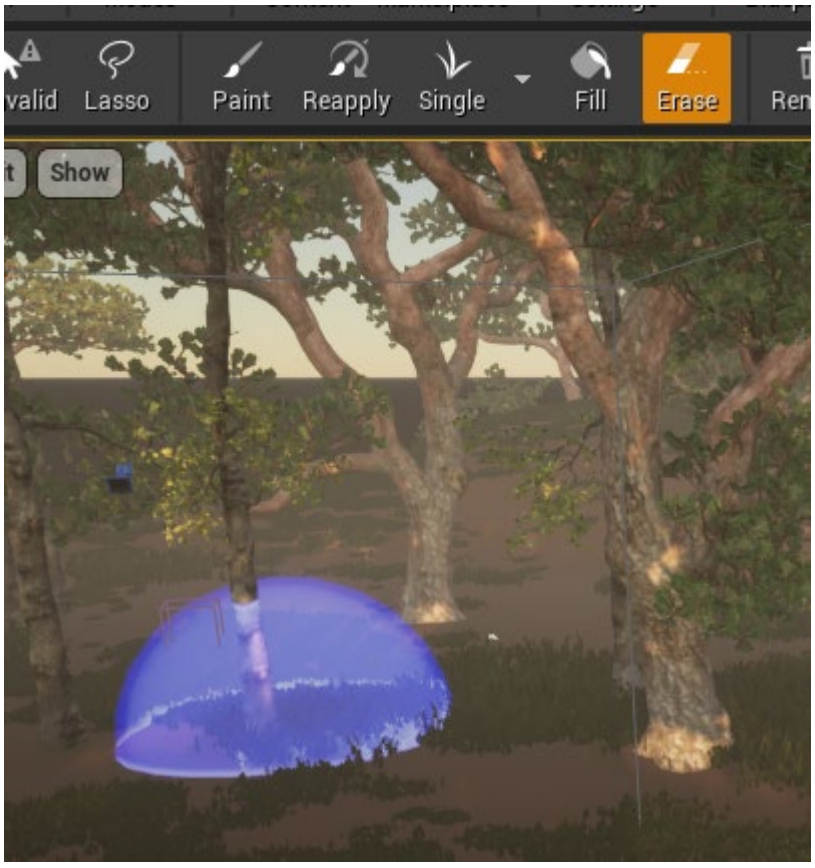


Рисунок 3.15 – Інструмент Erase

Інструмент Erase дозволяє видалити лише вибрані об’єкти з ділянки Foliage. Для його використання було обрано статичні сітку об'єкти, і проведено пензлем напівпрозорою сферою над областю для видалення. За допомогою цього інструменту було видалено лише вибрані статичні сітки (рис. 3.15).

3.3 Налаштування освітлення ігрового рівня

Основним світлом на ігровому рівні в UE4 є Directional Light. Directional імітує світло, що випромінюється від нескінченно далекого джерела [14]. Це означає, що всі тіні, які відкидає це світло, будуть паралельними, що робить правильну імітацію сонячного світла. Directional Light можна встановити в один із трьох параметрів мобільності :

– Статичний – означає, що світло не можна змінювати в грі. Це найшвидший спосіб візуалізації та дозволяє запекти освітлення;

– Стаціонарний – означає, що світло матиме лише тіньове та відбивне освітлення від статичної геометрії, випеченої Lightmass , все інше освітлення буде динамічним. Цей параметр також дозволяє світлу змінювати колір та інтенсивність в грі, але воно не рухається і дозволяє частково запекти освітлення;

– Динамічний – означає, що світло є абсолютно динамічним і дозволяє здійснювати динамічне затінення. Це найповільніший з точки зору візуалізації, але забезпечує найбільшу гнучкість під час гри.

Для Directional Light встановлено динамічний параметр мобільності. Це означає, що світло динамічне та дозволяє динамічно затіняти на ігровому рівні, що забезпечує найбільшу гнучкість під час гри.

Рисунок 3.16 – Сцена після додавання Directional Light

Після Directional Light додано Sky Atmosphere. Sky Atmosphere в UE4 – це фізична техніка відтворення неба й атмосфери [15]. Це досить гнучкий інструмент, щоб створити атмосферу, схожу на Землю, із часом доби, що включає схід і захід сонця, або створити позаземні атмосфери екзотичної природи. Також забезпечує перспективу з повітря, за допомогою якої можна імітувати переходи від землі до неба та космічного простору з правильною планетарною кривизною. Sky Atmosphere дає наближення до розсіювання світла через середовище планетарної атмосфери, надаючи зовнішнім рівням більш реалістичний або екзотичний вигляд, включаючи наступне: можна отримати зображення сонячного диска в атмосфері з кольором неба, який залежить від сонячного світла та властивостей атмосфери; колір неба змінюватиметься залежно від висоти сонця або наскільки вектор домінуючого спрямованого світла наближається до паралелі землі; контроль розсіювання та

нечітких налаштувань, що дозволяє повністю контролювати вашу атмосферну щільність. Повітряна перспектива, яка імітує кривизну світу під час переходу від землі до неба до космосу.

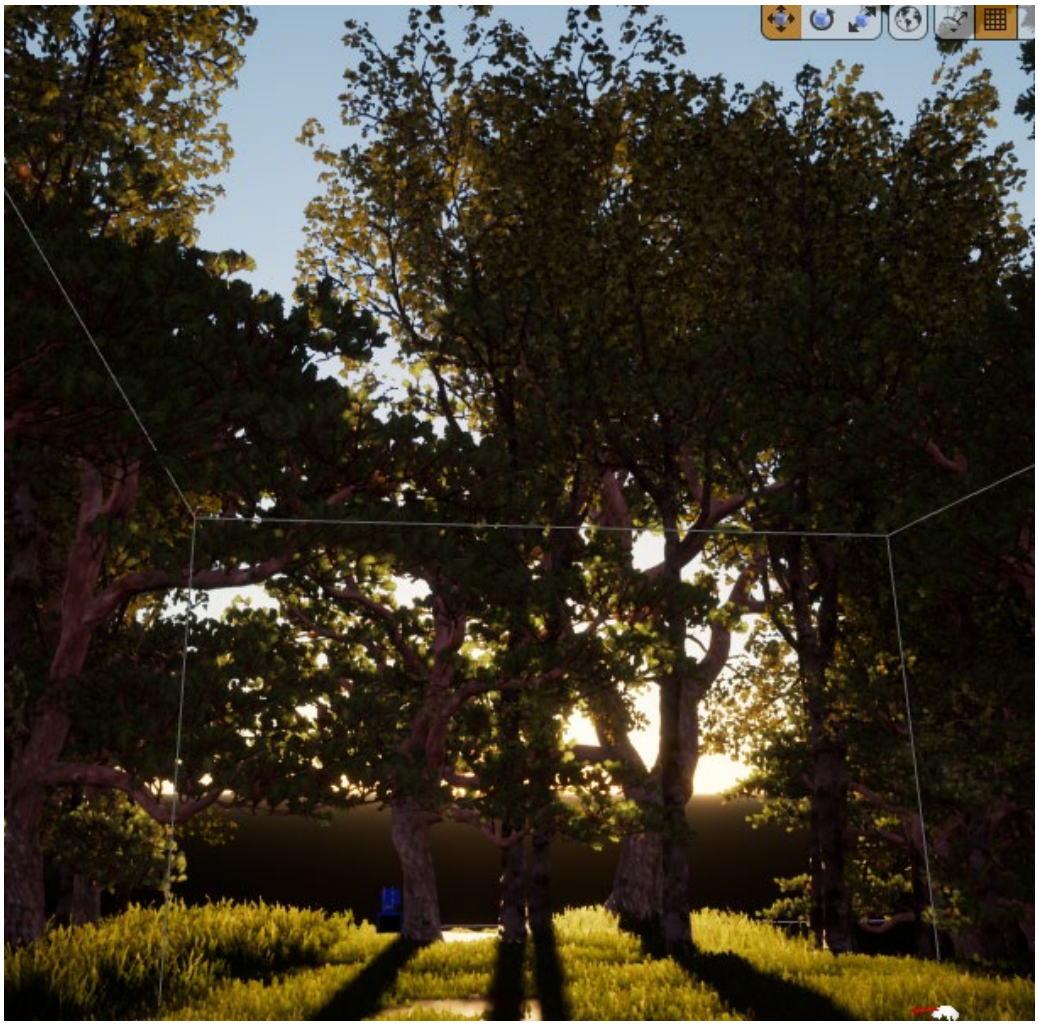


Рисунок 3.17 – Sky Atmosphere

Після цього додано Sky Light. Sky Light захоплює віддалені частини рівня та застосовує їх до сцени як світло. Це означає, що зовнішній вигляд неба та його освітлення/відблиски будуть збігатися, навіть якщо небо надходить з атмосфери, або шаруватих хмар на вершині скайбокса, або далеких гір [16].

Для того, щоб трава та гілки дерев рухались на сцену додано Wind Directional Source. Wind Directional Source імітує вітер на ігровому рівні. За допомогою цього інструменту можна налаштувати максимальну кількість пориву, встановити максимальне відхилення для поривів вітру, мінімальну

кількість пориву, налаштовувати радіус, швидкість генерованого вітру, силу та тип вітру.

Рисунок 3.17 – Sky Light

Також до сцени додано туман за допомогою Exponential Height Fog (рис. 3.18). Exponential Height Fog створює більшу щільність у низьких місцях карти та меншу щільність у високих місцях [17]. Перехід плавний, тому ніколи не отримається жорсткий зріз, при збільшенні висоти (віддалення камери).

Рисунок 3.18 - Exponential Height Fog

Після того, як було налаштовано освітлення на сцену додано Post Process Volume. Post Process у Unreal Engine 4 дозволяє створювати різноманітні художні ефекти та змінювати загальний вигляд усієї гри. Ефекти Post Process можуть використовуватися окремо для впливу лише на певну область або всю сцену, також має можливість перекривати кілька томів після обробки та відтворювати їхні ефекти на основі їх пріоритету. Об’єми після обробки можна використовувати для додавання або зміни простих ефектів, таких як Блум, відблиски об’єктива, адаптація очей, глибина різкості тощо, а також їх можна використовувати для отримання розширених ефектів за допомогою матеріалів. Ще однією чудовою функцією Post Process Volume є таблиця пошуку (LUT), яка використовується для зберігання перетворень кольорів із програмного забезпечення для редагування зображень. Замість використання трьох одновимірних (1D) таблиць пошуку використовується одна тривимірна (3D)

таблиця пошуку. Це пропонує більш складну трансформацію кольору, яку можна використовувати для чогось на кшталт зменшення насиченості [18].

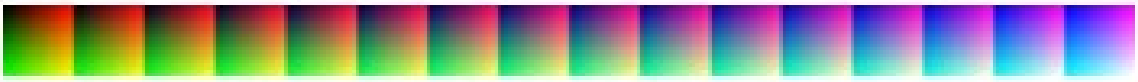


Рисунок 3.19 – LUT за замовченням

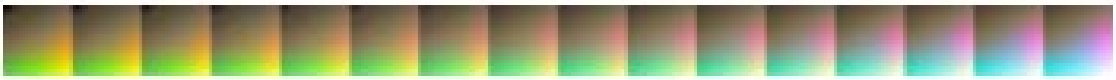


Рисунок 3.20 – LUT після корекції кольору

Використання системи LUT дає змогу оптимізувати гру, тому що замінює стандартні налаштування Post Process Volume, які дуже навантажують процесор. Таблиця LUT яка має набір кольорів за замовченням додається в PS і в подальшому за рахунок корекції кольору змінюється та додається в поле в Post Process Volume.

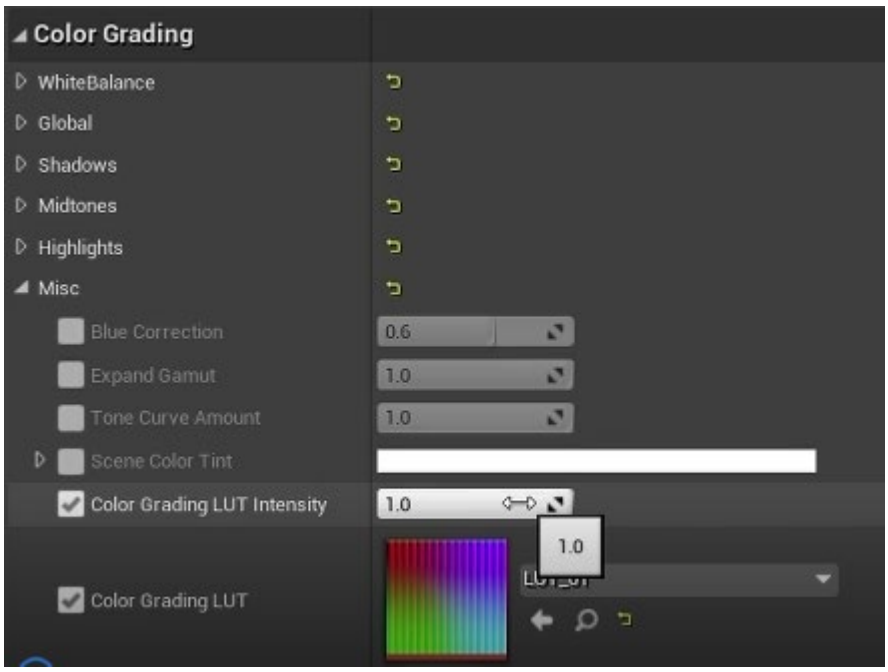


Рисунок 3.21 – Додавання LUT в UE4

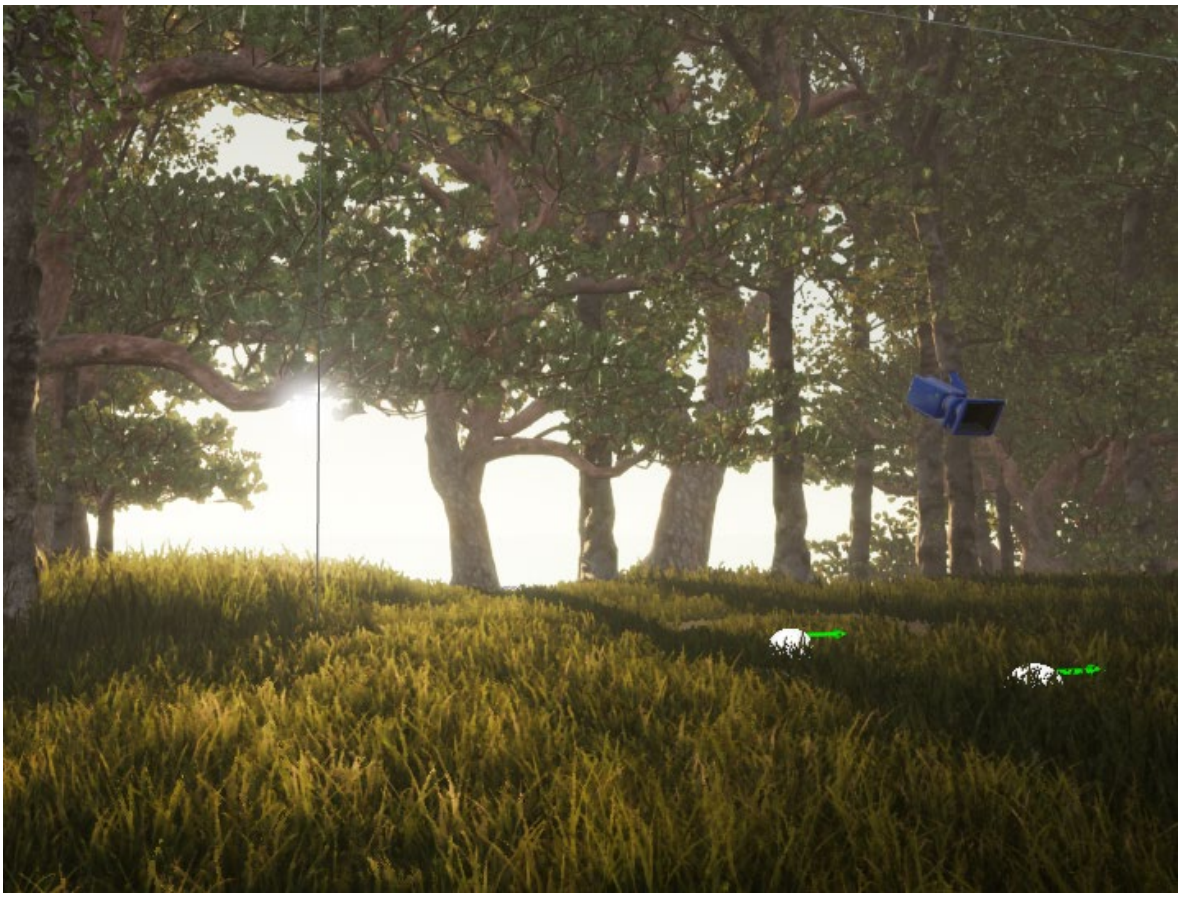


Рисунок 3.22 – Результат в UE4

На рисунку 3.22 результат створення ігрового рівня в Unreal Engine 4 з налаштованим освітленням та пост-обробкою.

Висновки до розділу 3

Розроблено 3D об’єкти рослинності для процедурної генерації. Створено high poly та low poly об’єкти, UV розгортку та атласи в Maya 3D. Зроблено текстури в Substance Painter. В Unreal Engine 4 створено ландшафт для сцени та матеріал. За допомогою інструменту Procedural Foliage Spawner для створення процедурної генерації заповнено ігровий рівень рослинністю. За рахунок інструментів Foliage додано об’єктів та заповнено прогалини після процедурної генерації.

До ігрового рівня додано освітлення: Directional Light, Sky Atmosphere, Sky Light. Додано туман за допомогою інструменту Exponential Height Fog. Пост обробку сцени зроблено за рахунок Post Process Volume та оптимізовано за рахунок системи LUT.

4 Реалізація комп'ютерної 3D гри з використанням системи Blueprint в Unreal Engine 4

4.1 Реалізація покрокової системи

Blueprint в Unreal Engine - це система сценаріїв ігрового процесу, заснована на концепції використання інтерфейсу на основі вузлів для створення елементів ігрового процесу з редактора Unreal Editor. Як і більшість розповсюджених мов сценаріїв, вона використовується для визначення об'єктно-орієнтованих (ОО) класів або об'єктів в рушії.

Основна логіка полягає в тому, що персонажі атакують по черзі і їх хід залежить від їх швидкості. Система працює на тому, що персонаж який не ходив потрапляє в чергу, коли хід закінчився, перевіряється ця черга, а потім ініціюється хід цього персонажа.

Спочатку створено ігровий режим та контролер гравця, а також компонент, який оброблятиме кожного персонажа та буде містити в собі основні команди.

В blueprint ігрового режиму оброблюється чий хід наступний. В ньому міститься подія – TurnRequest (рис. 4.1), яка викликається персонажем гравця або ворожим персонажем про те, що їх черга ходити. Створено масив, в якому містяться клас для персонажів та клас для ворогів. Addunique відповідає за те, щоб в масиві не було дублікатів, а саме кожен персонаж з’являвся лише один раз. І потім викликається подія початку ходу – StartTurn (рис. 4.2). Для того, щоб викликався для ходу лише один персонаж використовується вузол DoOnce. Далі йде перевірка індексу масиву з персонажами, якщо він не порожній, то викликається функція початку ходу (рис. 4.3), якщо порожній, то ходить інший персонаж. Якщо індекс був не порожнім, то викликається функція початку ходу та видаляється індекс для того, щоб хід переходив до іншого персонажа в черзі.

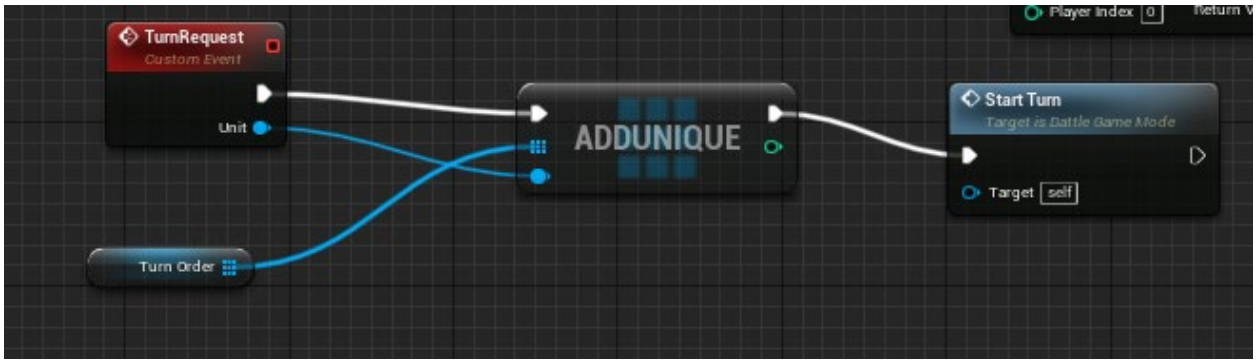


Рисунок 4.1 – Подія TurnRequest

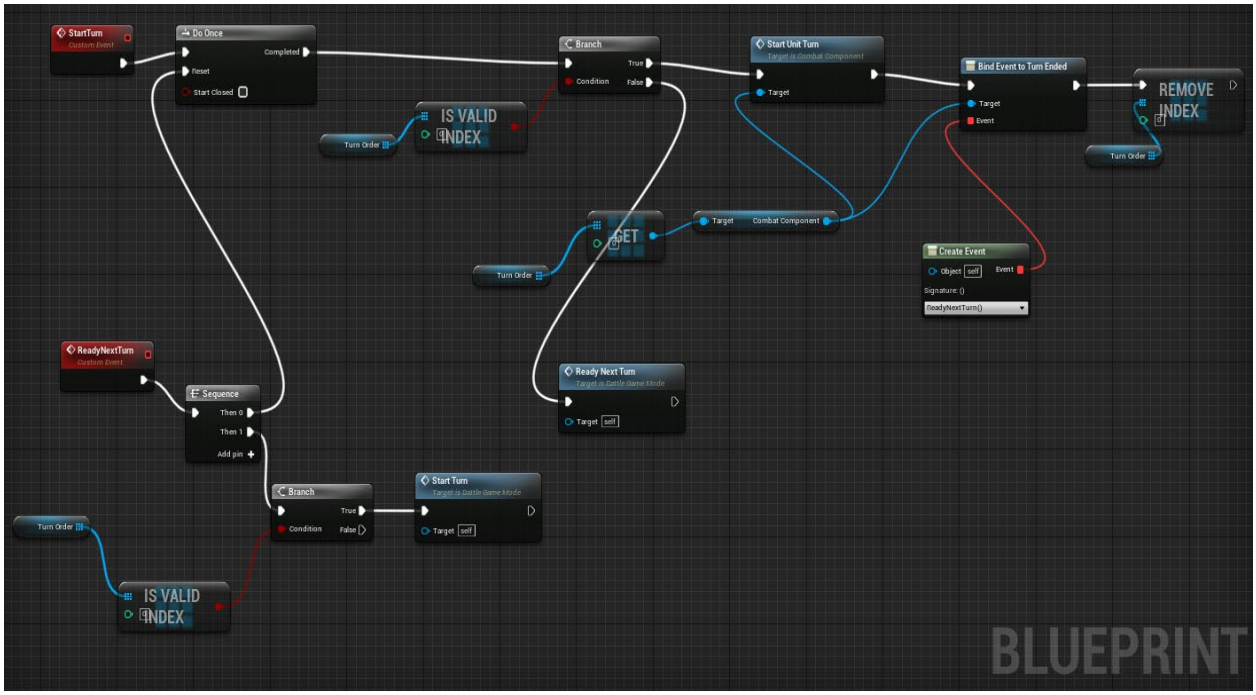


Рисунок 4.2 – Blueprint для початку ходу

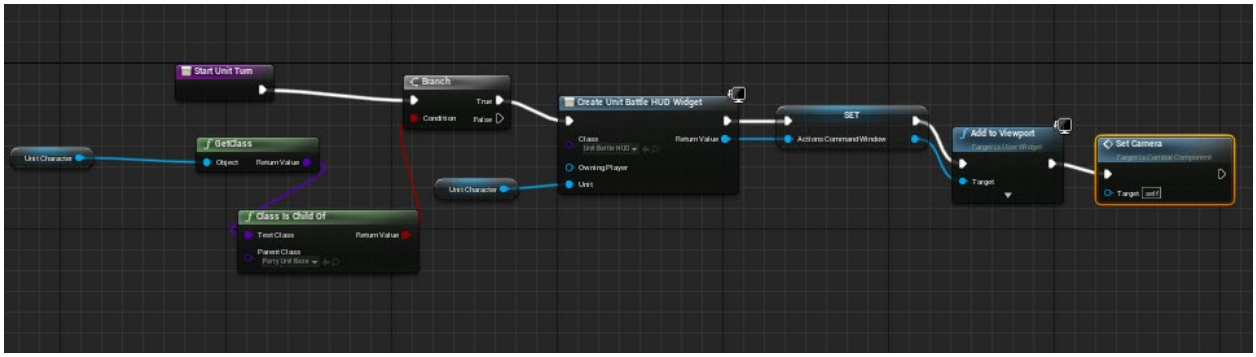


Рисунок 4.3 – Функція початку ходу

Функція початку ходу містить в собі вузол в якому перевіряється до якого класу належить персонаж, далі викликається вікно інтерфейсу для персонажа чий хід та прив’язується камера до нього.

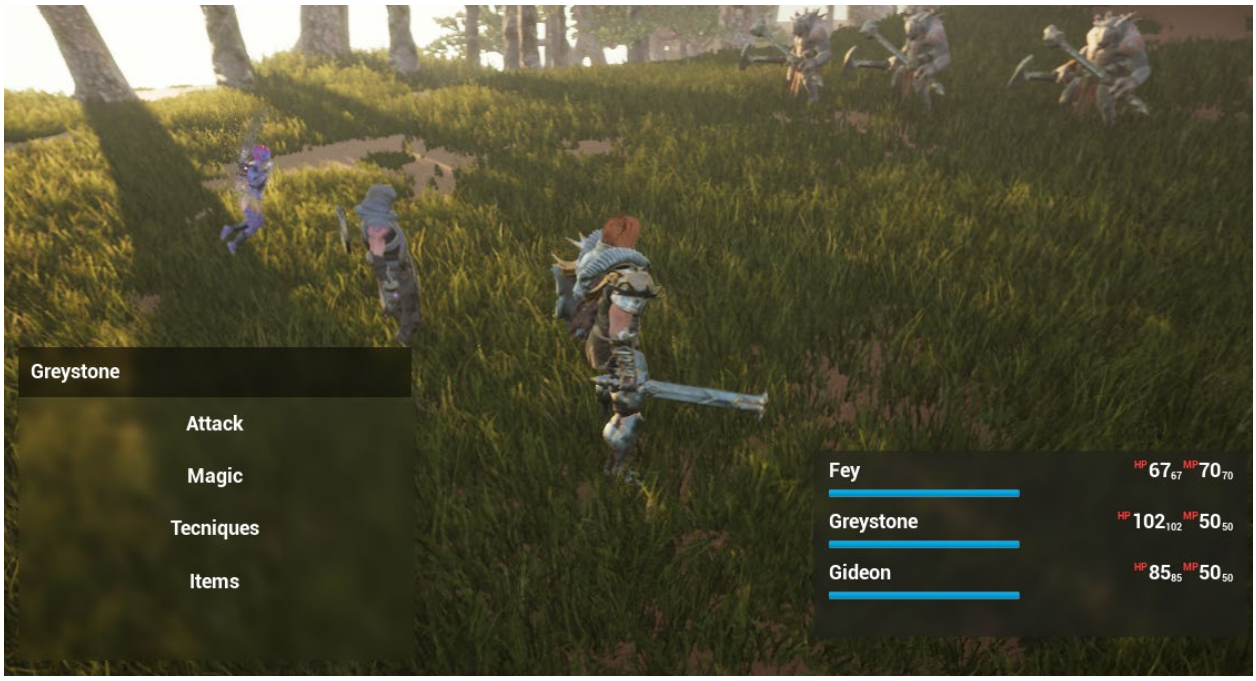


Рисунок 4.4 – Початок ходу

Інтерфейс гравця містить у собі вкладку з здоров’ям кожного персонажа та його маною справа та зліва ActionMenu. Гравець обирає дію під час ходу персонажа. Після того, як гравець обирає один з видів атак на екрані з’являється список з цілями для нападу, в якому гравець обирає під час ходу персонажа на кого напасти. Для того, щоб можна було атакувати лише ворожих персонажів створено функцію PopulateTargetList (рис. 4.5). У цій функції якщо натиснуто кнопку атаки, то в списку для атаки з’являються ворожі персонажі.

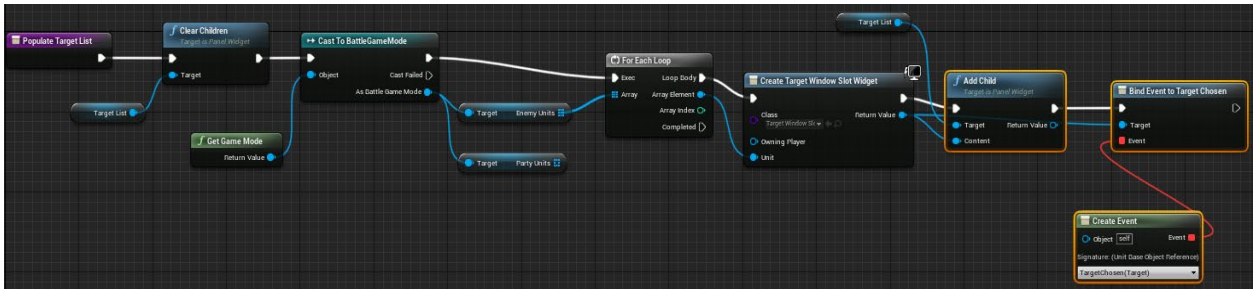


Рисунок 4.5 – Функція перевірки персонажа

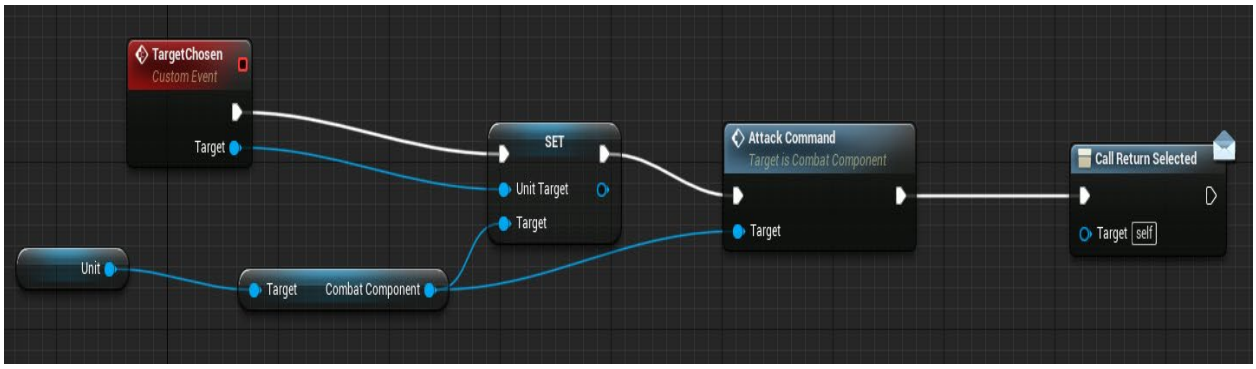


Рисунок 4.6 – Подія для вибору персонажа

Після того, як обрано ціль для атаки викликається blueprint в якому обробляється інформація про нанесення пошкодження ворогу.

4.2 Реалізація механік атаки

Спочатку було створено таблицю даних в Excel у форматі .csv (рис. 4.7), яка містить у собі інформацію про:

– Рівень гравця;

– Здоров’я;

– Ману;

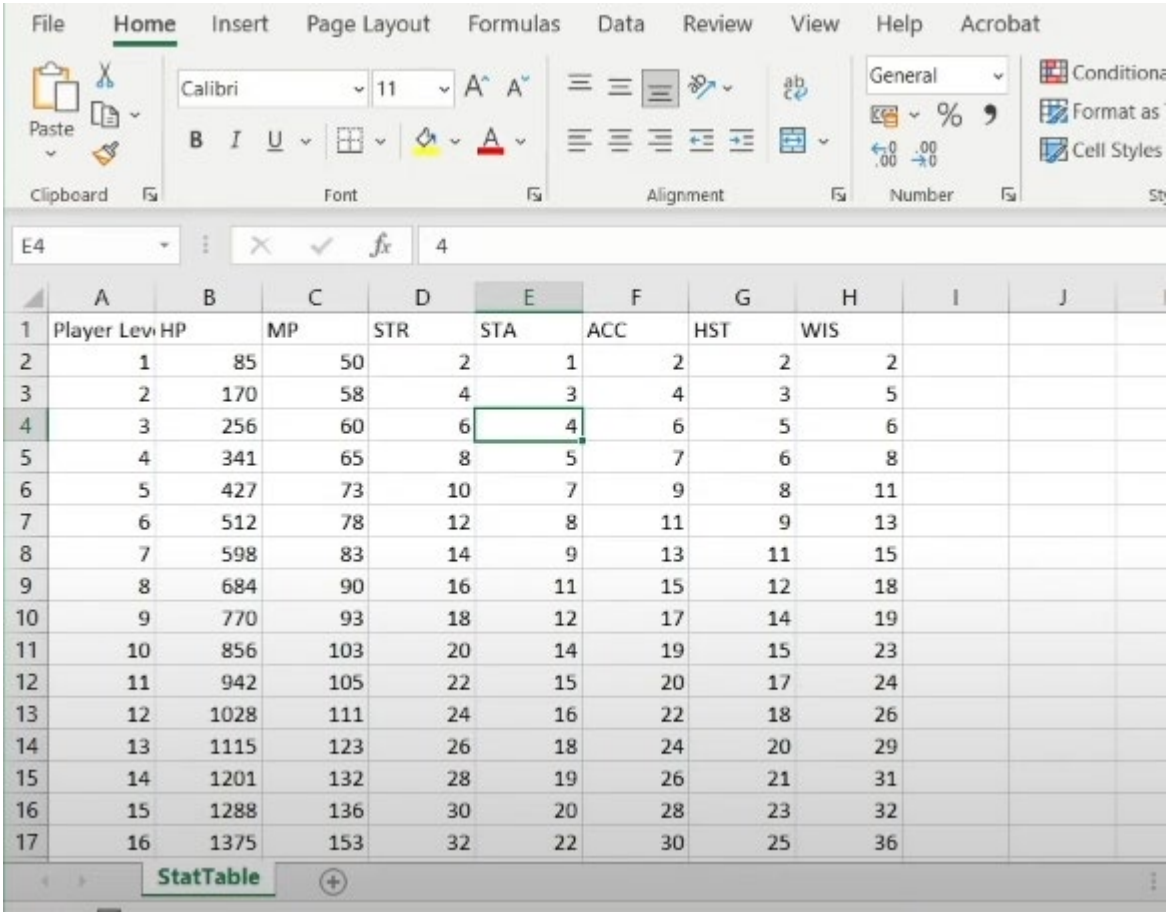
– Силу;

– Витривалість;

– Точність;

– Квапливість;

– Мудрість.



File Home Insert Page Layout Formulas Data Review View Help Acrobat

Paste Calibri 11 General Conditiona Format as Cell Styles

Clipboard Font Alignment Number

E4 4

	A	B	C	D	E	F	G	H	I	J
1	Player Lev	HP	MP	STR	STA	ACC	HST	WIS		
2	1	85	50	2	1	2	2	2		
3	2	170	58	4	3	4	3	5		
4	3	256	60	6	4	6	5	6		
5	4	341	65	8	5	7	6	8		
6	5	427	73	10	7	9	8	11		
7	6	512	78	12	8	11	9	13		
8	7	598	83	14	9	13	11	15		
9	8	684	90	16	11	15	12	18		
10	9	770	93	18	12	17	14	19		
11	10	856	103	20	14	19	15	23		
12	11	942	105	22	15	20	17	24		
13	12	1028	111	24	16	22	18	26		
14	13	1115	123	26	18	24	20	29		
15	14	1201	132	28	19	26	21	31		
16	15	1288	136	30	20	28	23	32		
17	16	1375	153	32	22	30	25	36		

StatTable

Рисунок 4.7 – Таблиця даних показників

Для обробки в Unreal Engine було створено структуру даних в яких поля відповідають даним в таблиці Excel.

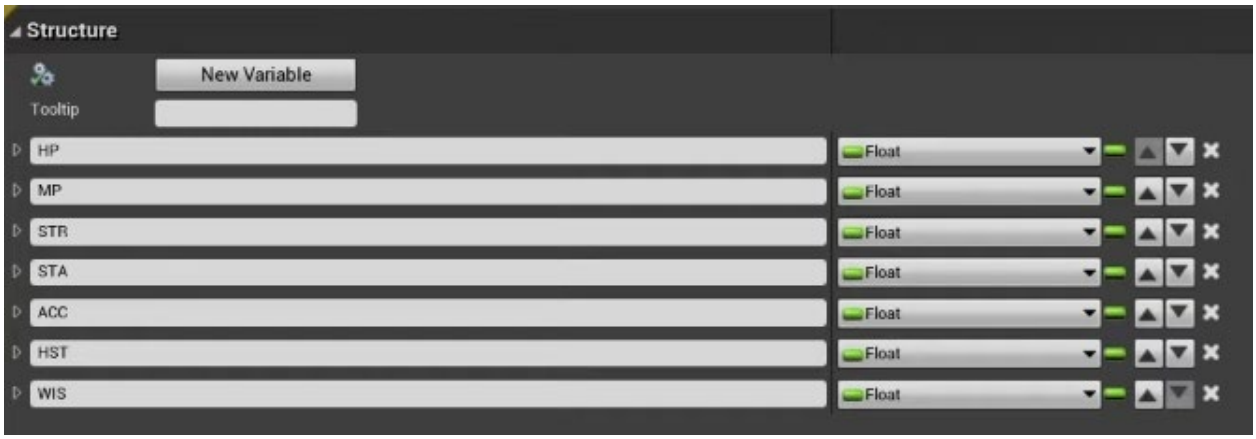


Рисунок 4.8 – Структура даних в UE4

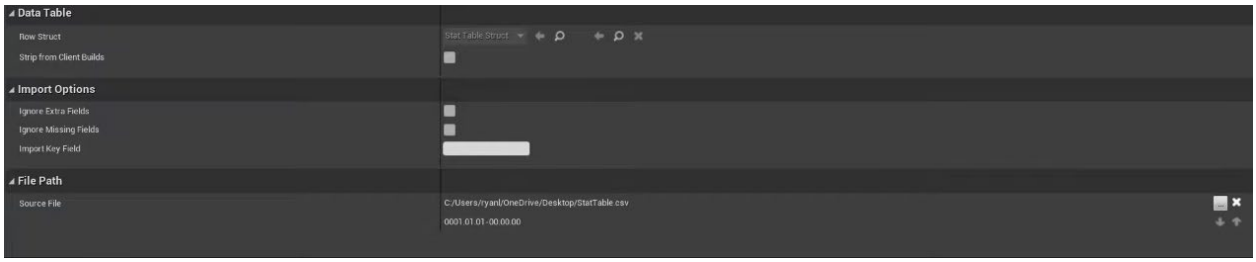
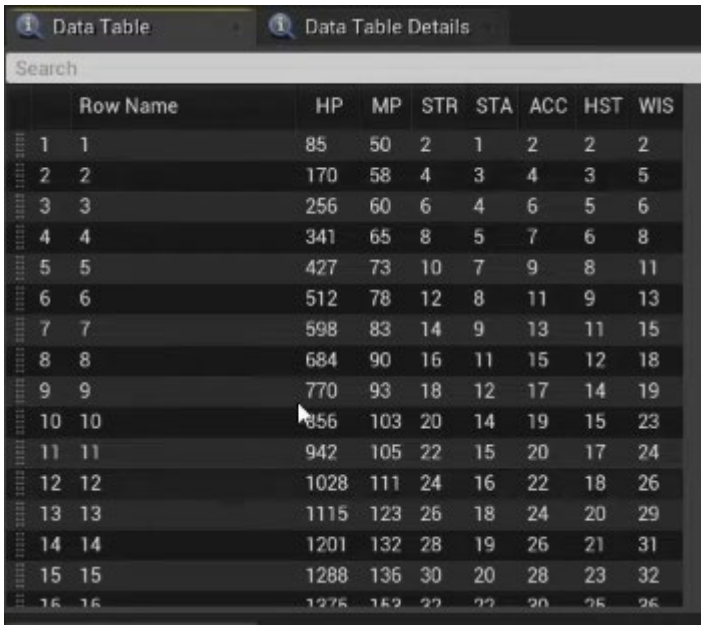


Рисунок 4.9 – Підключення таблиці даних Excel до UE4



	Row Name	HP	MP	STR	STA	ACC	HST	WIS
1	1	85	50	2	1	2	2	2
2	2	170	58	4	3	4	3	5
3	3	256	60	6	4	6	5	6
4	4	341	65	8	5	7	6	8
5	5	427	73	10	7	9	8	11
6	6	512	78	12	8	11	9	13
7	7	598	83	14	9	13	11	15
8	8	684	90	16	11	15	12	18
9	9	770	93	18	12	17	14	19
10	10	856	103	20	14	19	15	23
11	11	942	105	22	15	20	17	24
12	12	1028	111	24	16	22	18	26
13	13	1115	123	26	18	24	20	29
14	14	1201	132	28	19	26	21	31
15	15	1288	136	30	20	28	23	32

Рисунок 4.10 – Завантаження даних в UE4

Кожен персонаж має базові значення з таблиці даних, на додаток до цього, модифікатори для зміни значень кожного персонажа, в залежності від їх класу (рис. 4.11).

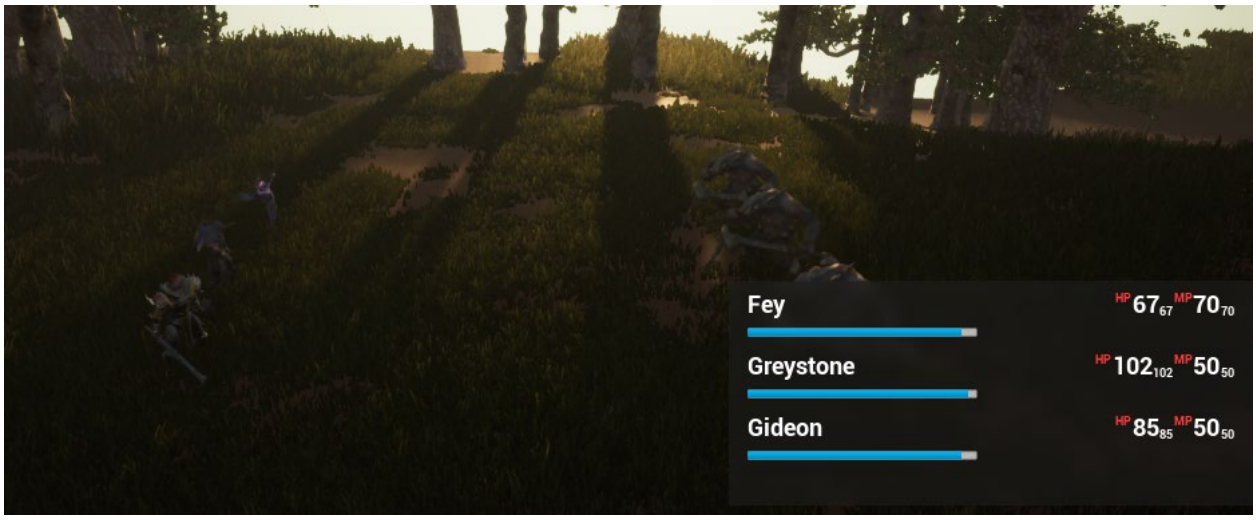


Рисунок 4.11 – Характеристики героїв під час ігрового процесу

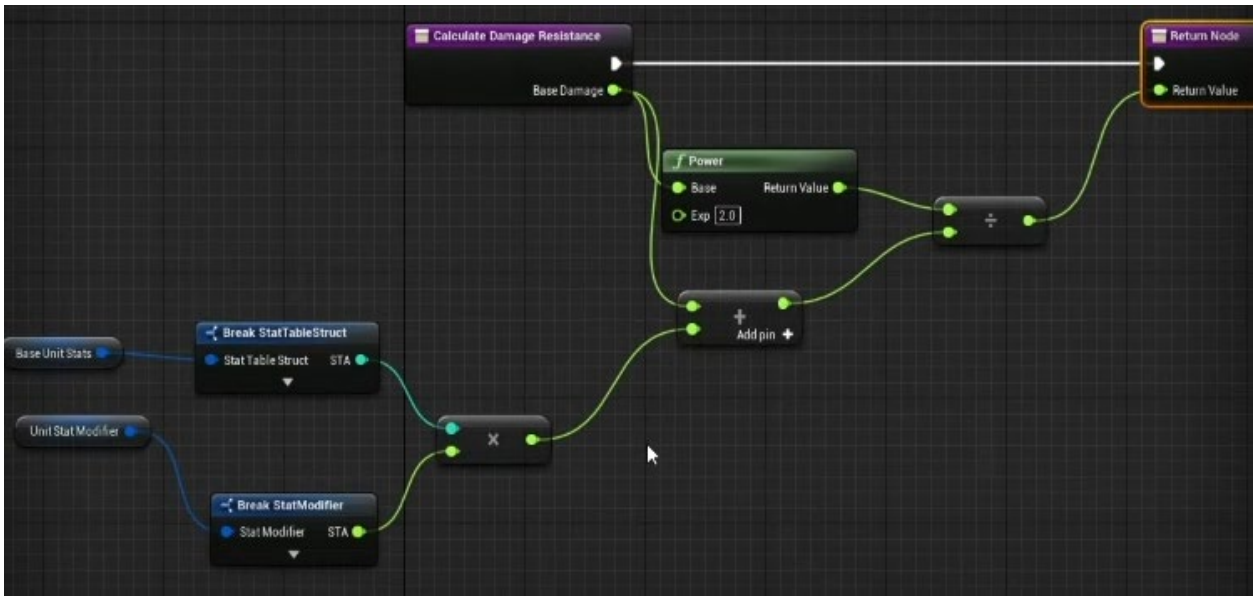


Рисунок 4.12 – Функція для обчислення нанесення пошкодження

В залежності від типу атаки, сили та витривалості персонажа вираховується нанесене пошкодження (рис. 4.12).

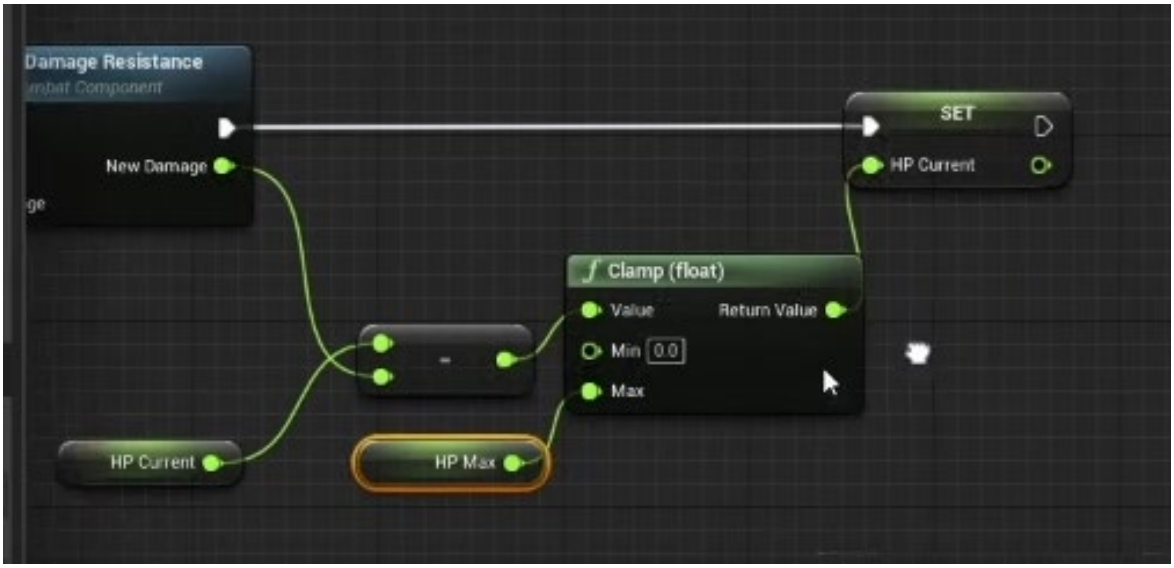


Рисунок 4.13 – Вузли для зміни здоров'я після пошкодження

Від змінної, яка містить поточне здоров’я в залежності від нанесеного значення пошкодження зменшується показник здоров’я персонажа.

Для кожного персонажа для атаки додано ефекти. Після того як відбувається анімація атаки вибрано момент в яких додається ефект. Коли герой атакує під час анімації [19] викликається спеціальний ефект (рис. 4.14).

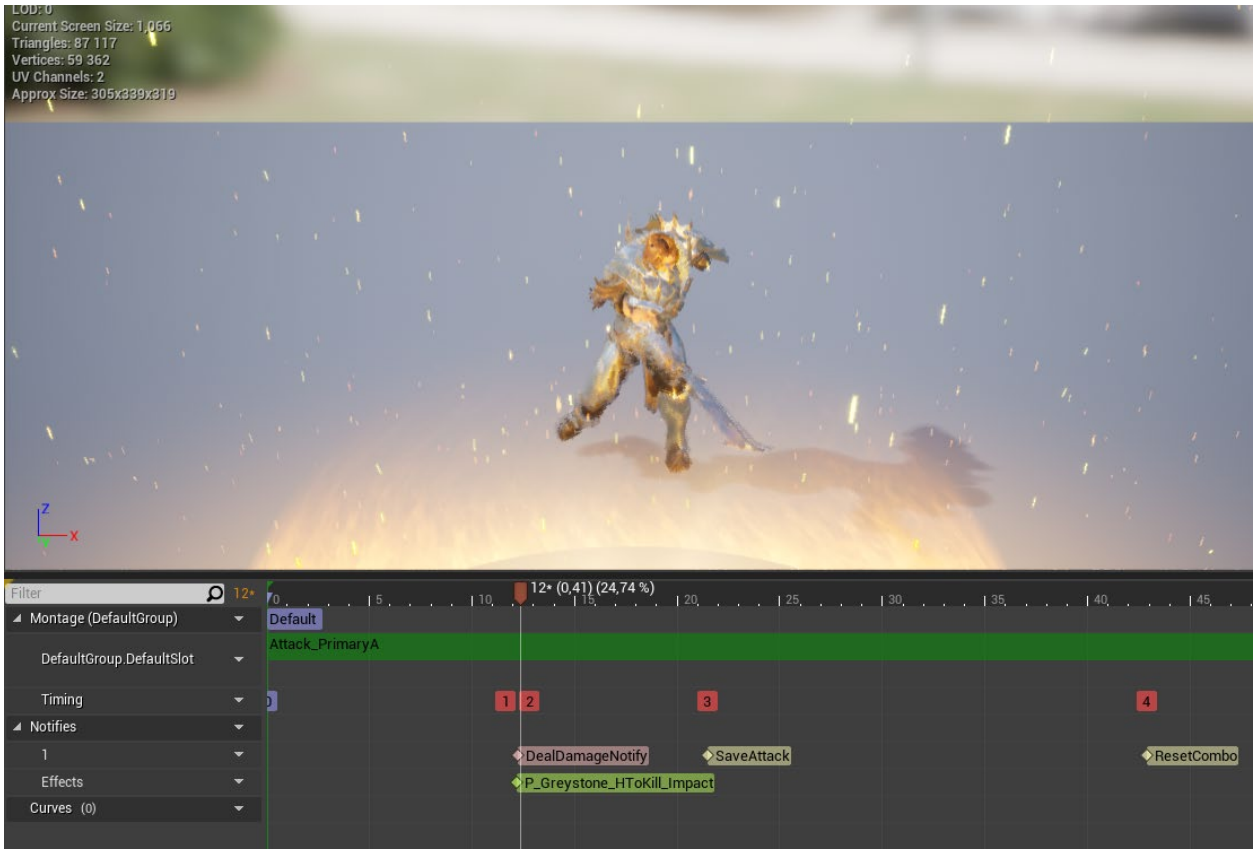


Рисунок 4.14 – Редагування ефекту під час анімації

Рисунок 4.15 – Ефект атаки під час гри

Окрім ефекту звичайної атаки додано ефект та анімацію для випускання вогняної кулі. Створено окремий об’єкт кулі на яку додано ефект вогню (рис.

4.16), а також ефект який додано на анімацію запуску кулі самого персонажа (рис. 4.17).

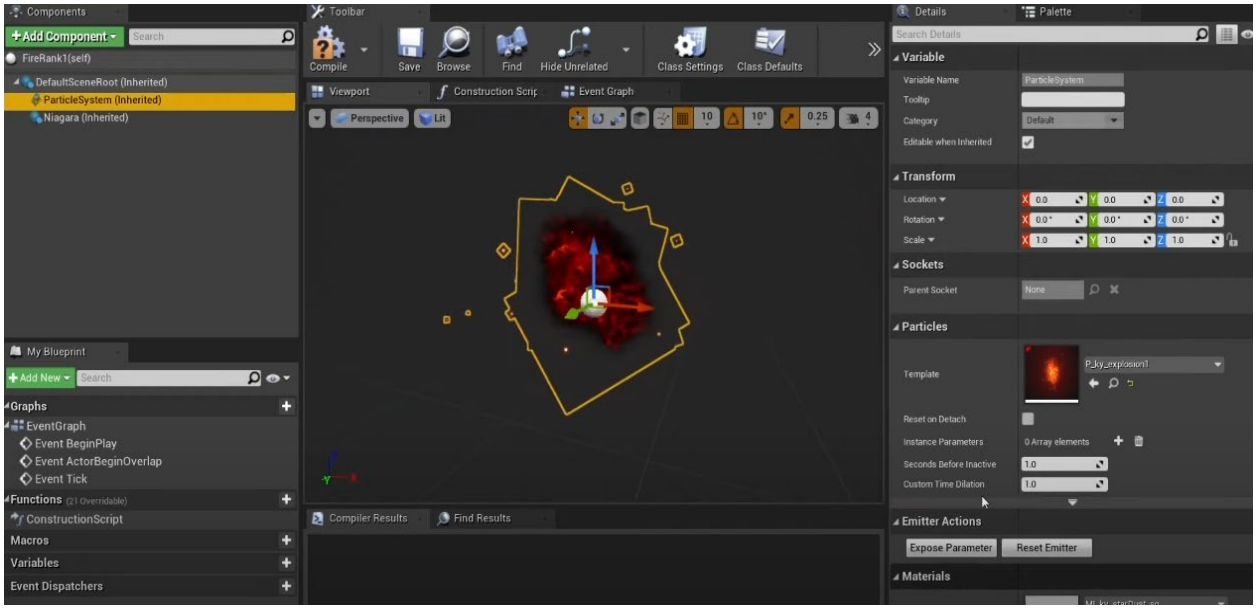


Рисунок 4.16 – Редагування ефекту вогняної кулі

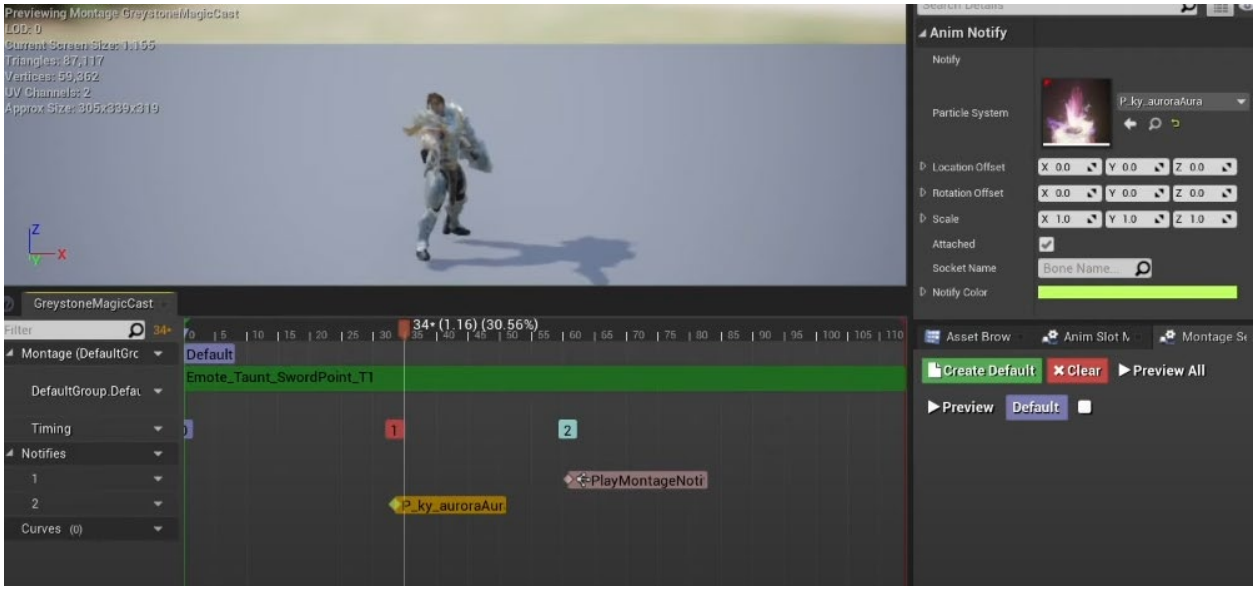


Рисунок 4.17 – Редагування ефекту виклику вогняної кулі

Для того, щоб під час виклику вогняного шару під час програвання анімації ефект не пропадав, було налаштовано повторення в системі Niagara в Unreal Engine. Niagara – це система створення та налаштування візуальних ефектів в рушії [20].

Рисунок 4.18 – Налаштування повторень ефекту

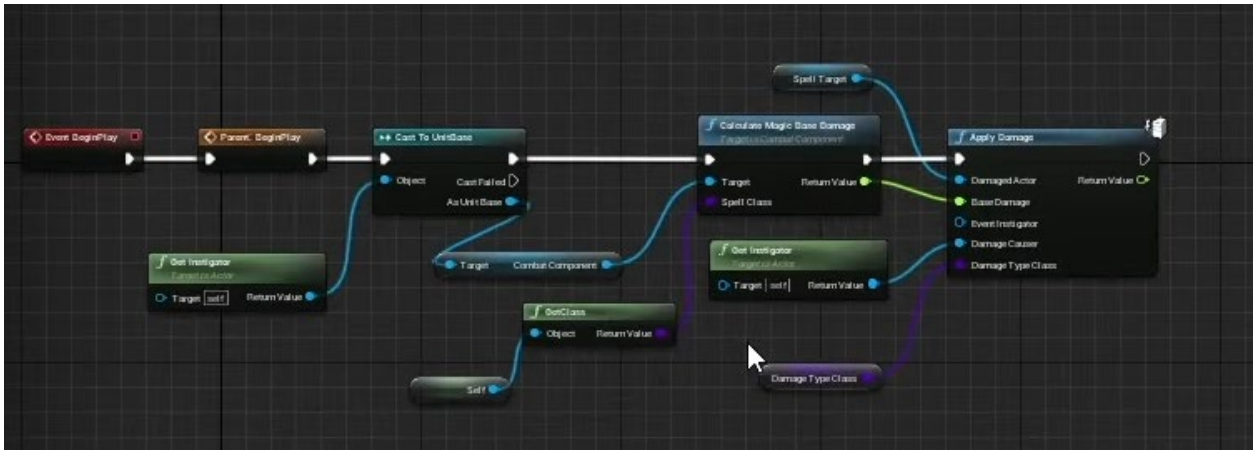


Рисунок 4.19 – Blueprint для магії

4.3 Реалізація збереження прогресу та системи отримання нагород

Для зберігання прогресу гри було використано SaveGame клас від Unreal Engine. Створивши клас SaveGame, з’являється можливість заповнювати його змінними для зберігання даних гри. Коли гра зберігається, інформація переноситься з поточного ігрового світу в об’єкт SaveGame, а під час завантаження гри це копіюється з об’єкта SaveGame в об’єкт гри, такі як персонажі, контролер гравця або ігровий режим.

Для того щоб знати, який файл зберігати створено змінну SaveFileName. Does Save Game Exist - для перевірки чи існує файл. Якщо існує, то тоді прогрес переноситься в об'єкт SaveGame і це буде зберігатись у змінній. Async

Save Game To Slot — це рекомендований спосіб збереження гри, оскільки він дозволяє уникнути збоїв навіть при збереженні великих обсягів даних. Проте, якщо дані збереження гри замалі або якщо зберігаються з меню чи екрана паузи, можна зберегти гру за допомогою вузла SaveGameToSlot.

У грі зберігаються отримані рівні та досвід, а також життя та мана, що є між битвами.

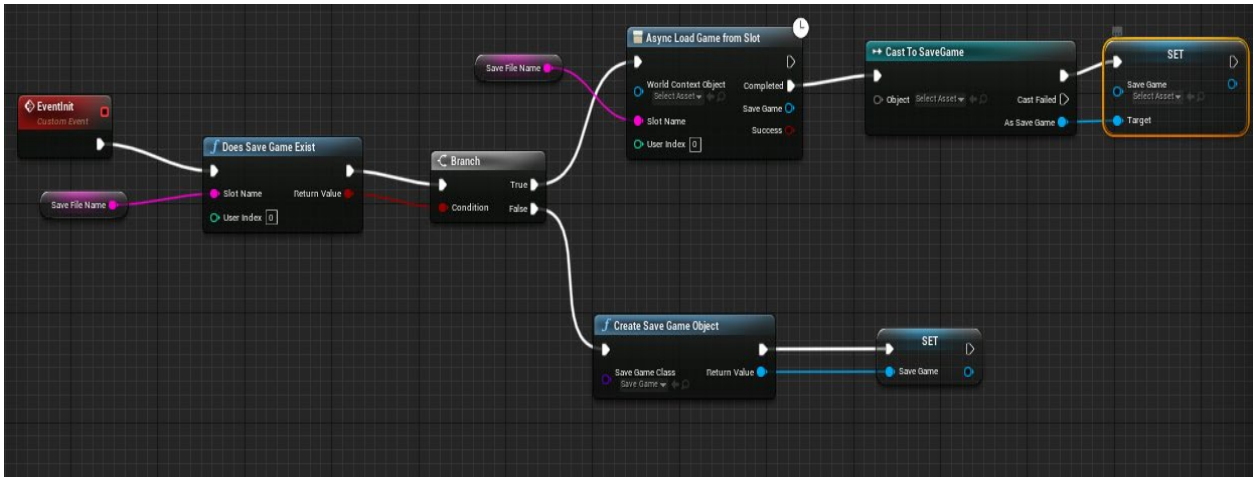


Рисунок 4.20 – Blueprint збереження прогресу

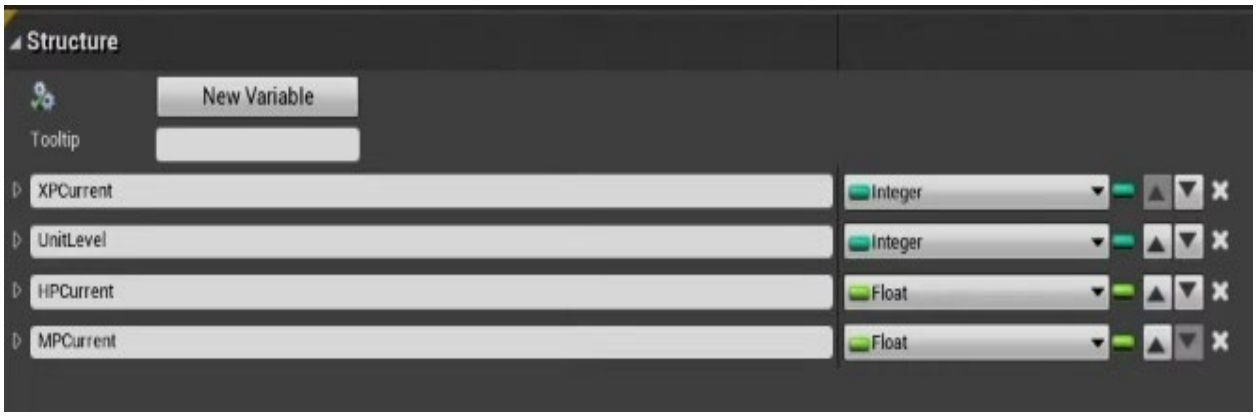


Рисунок 4.21 – Структура даних для збереження прогресу у грі

Після закінчення бою перемогою, на екран виводиться вікно з нагородами: золотом, досвідом та предметами (рис. 4.22).

Рисунок 4.22 – Вікно перемоги

На рисунку 4.23 функція отримання досвіду. Якщо досвіду достатньо, то відбувається перехід на новий рівень. Якщо не достатньо, то нічого не відбувається і досвід просто починає накопичуватись до тих пір, доки умова для переходу на наступний рівень не буде виконуватись.

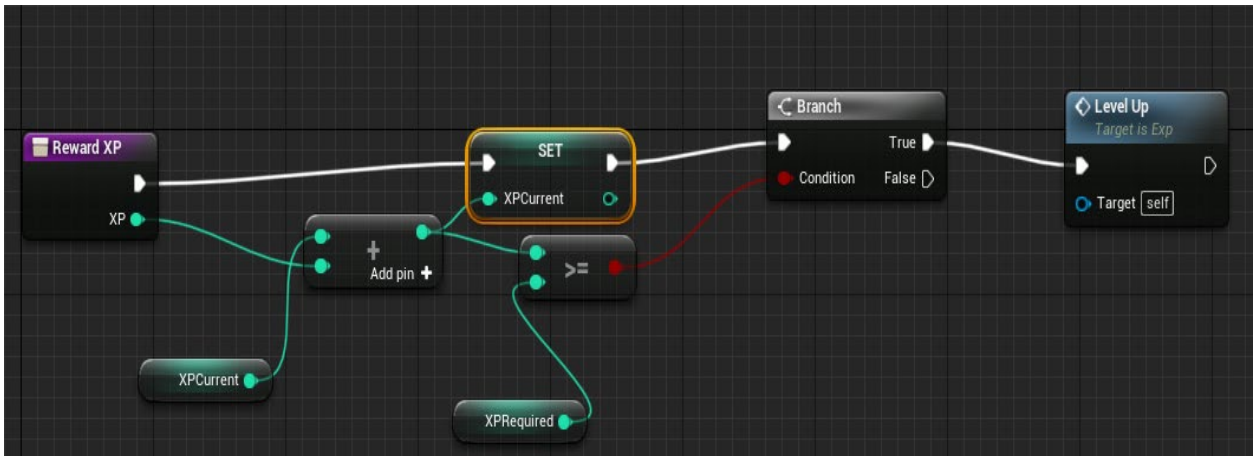


Рисунок 4.23 – Функція отримання досвіду

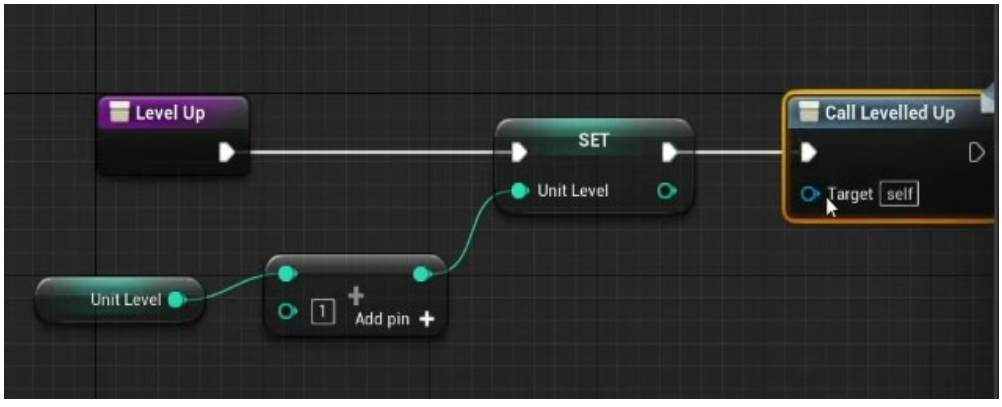


Рисунок 4.24 – Функція збільшення рівня

Компонент Widget є тривимірним екземпляром Widget Blueprint, з яким можна взаємодіяти у ігровому світі. У більшості ігор є необхідність виводити деяку інформацію своїм гравцям через інтерфейс користувача гри. Це може варіюватися від таких речей, як головне меню, меню паузи в грі, елементи HUD, предмети інвентарю або текст довідки, який вказує гравцям, що робити в певній ситуації. За допомогою Unreal Motion Graphics (UMG) можна створювати план віджетів для керування відображенням елементів інтерфейсу користувача у проекті. Widget Blueprint використовує Blueprint Visual Scripting для розробки макета, а також функціональних можливостей сценарію для елементів інтерфейсу користувача, наприклад, що відбувається, коли натискається кнопка або змінюється значення.

Для виводу рівня на екран було створено віджет, який буде виводитись на вікно перемоги (рис. 4.25).

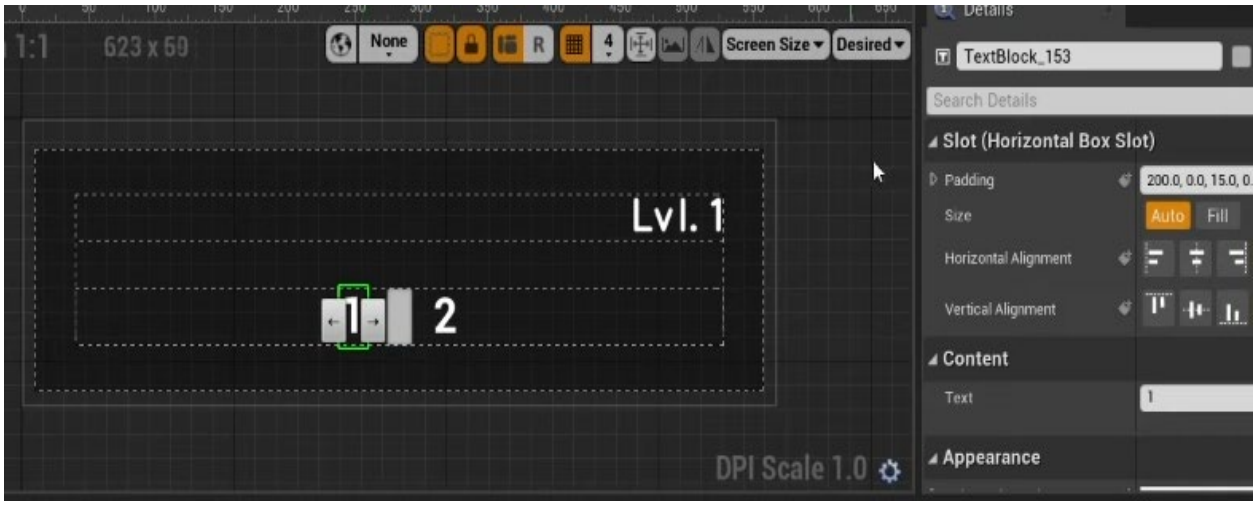


Рисунок 4.25 – Віджет системи досвіду

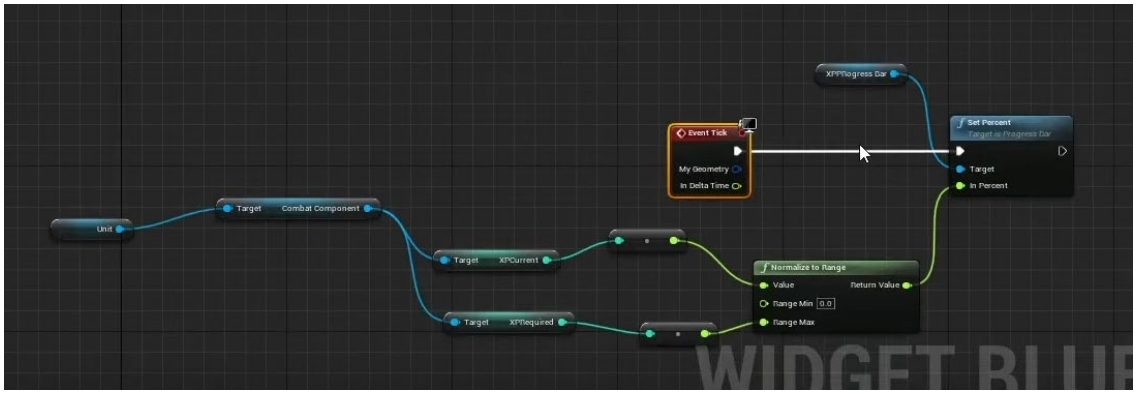


Рисунок 4.26 – Blueprint індикатору досвіду

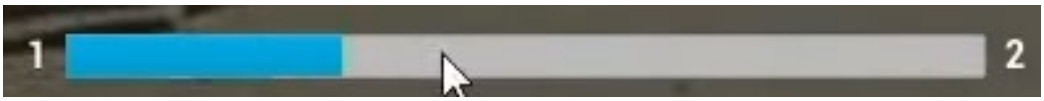


Рисунок 4.27 – Індикатор досвіду

Для індикатора досвіду у персонажа, коли він отримує нагороди створено подію. На рисунку 4.26 з масиву персонажів береться поточний досвід та необхідний досвід і ця інформація додається у Progress Bar з віджету.

4.4 Тестування продуктивності за допомогою режимів в Unreal Engine 4

Для тестування продуктивності сцени використано режим Unreal Engine 4 - Shader Complexity (рис. 4.28). Він використовується для візуалізації кількості інструкцій шейдера, які використовуються для обчислення кожного пікселя сцени [21]. Shader Complexity також використовується для оптимізації ефектів частинок, які, як правило, спричиняють стрибки продуктивності з великою кількістю перерисовки протягом короткого періоду часу. Режим використовує кольоровий спектр, щоб вказати, наскільки дорога сцена [22]. Зелений і червоний представляють лінійну залежність від «дуже недорогих» до «дорогих», а рожевий і білий вказують на великий стрибок до «дуже дорогих» пікселів. Якщо б більшість екрану була яскраво-червоною або білою, продуктивність була б поганою. Сцена в грі є достатньо оптимізована, що показує зелений колір.

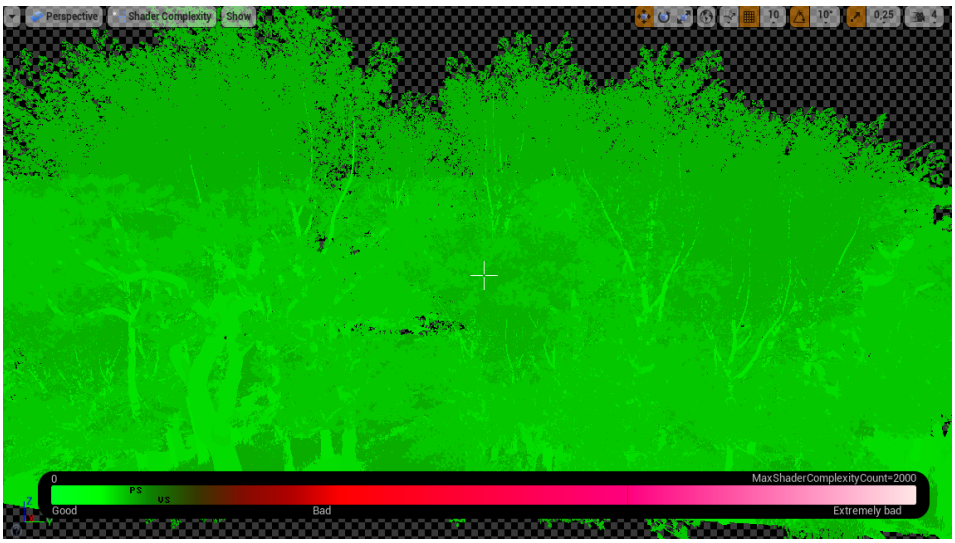


Рисунок 4.28 – Режим Shader Complexity

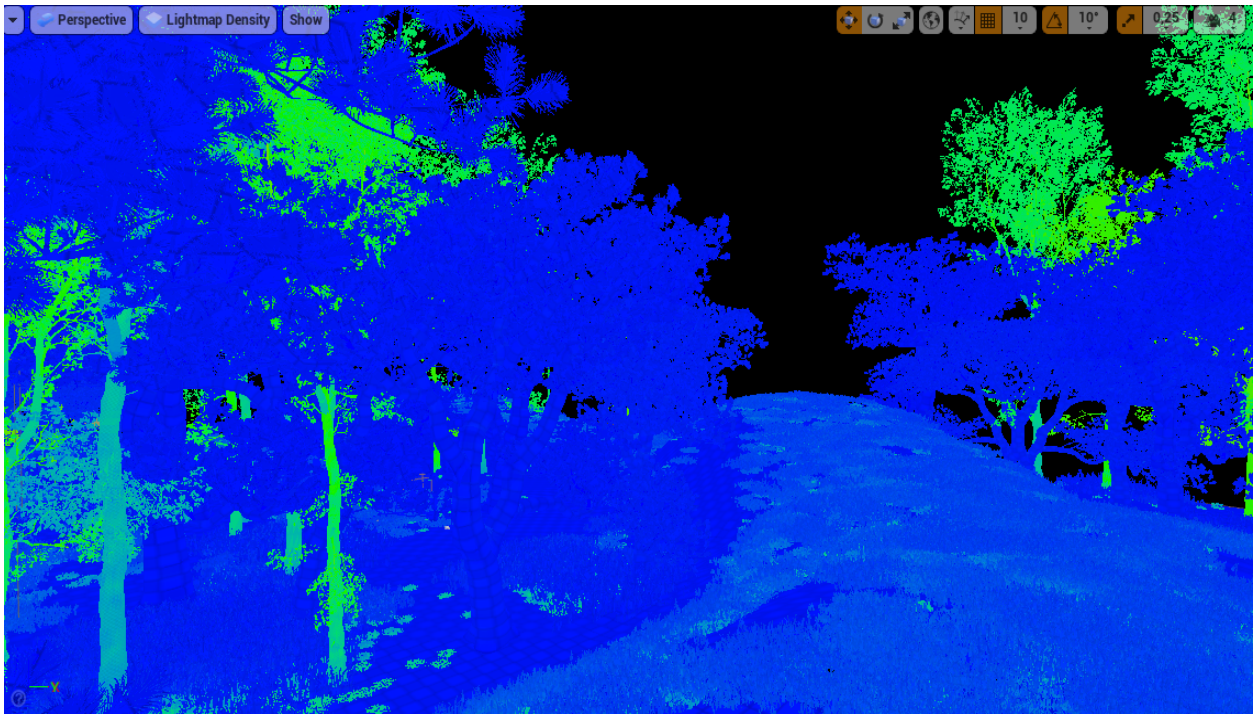


Рисунок 4.29 - Режим Lightmap Density

На рисунку 4.29 ігрова сцена в режимі Lightmap Density. Цей режим відображає щільність карти освітлення об’єктів, на які нанесено текстуру, кодуючи їх кольором відповідно до параметрів ідеальної/максимальної щільності та відображаючи сітку, яка відповідає фактичним текселям карти освітлення [23]. Важливо мати рівномірну щільність текселів по всій сцені, щоб отримати рівномірне освітлення карти освітлення.

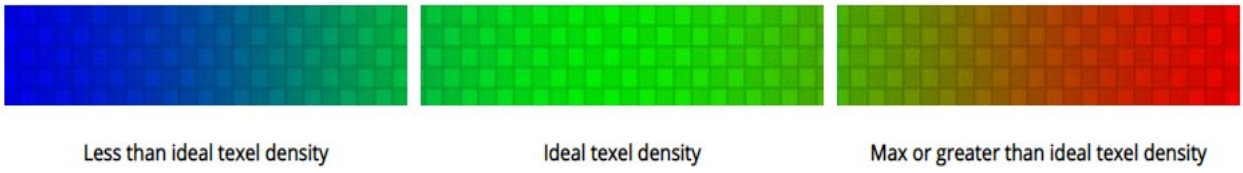


Рисунок 4.30 – Показники Lightmap Density

На рисунку 4.31 режим Light Complexity. Light Complexity показує кількість нестатичних джерел світла, які впливають на геометрію. Це корисно для відстеження вартості освітлення - чим більше світла впливає на поверхню, тим дорожче буде затіняти її [24].

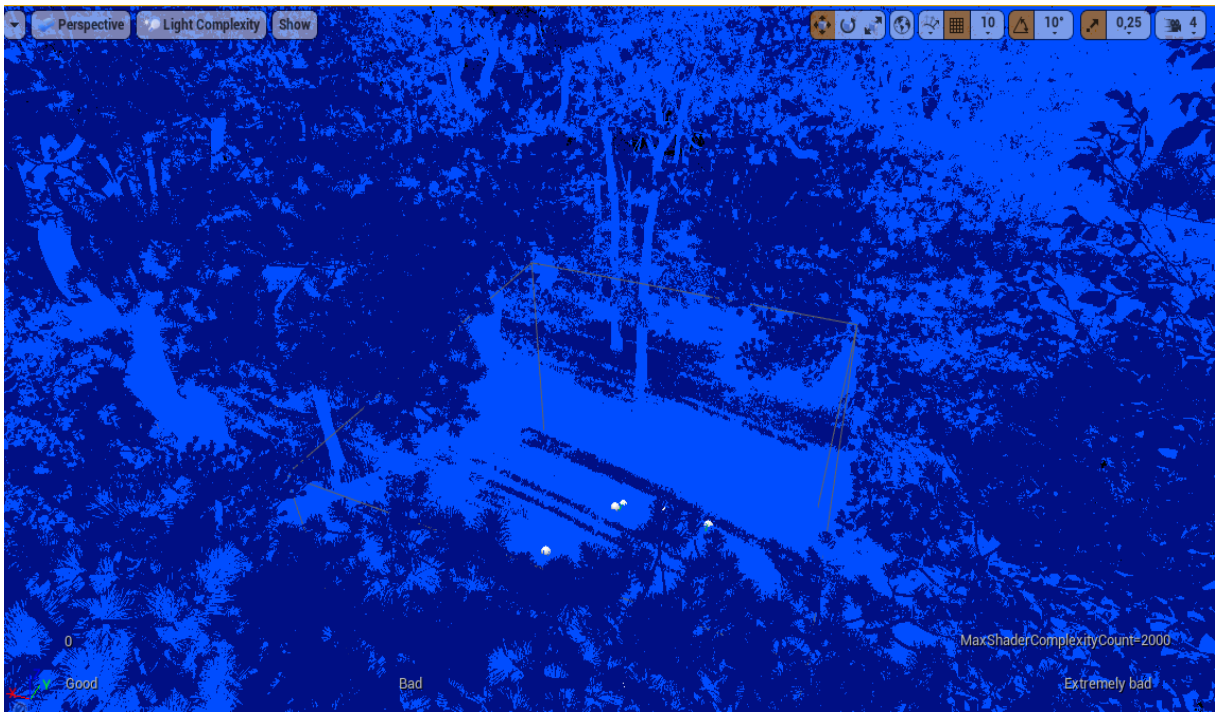


Рисунок 4.31 – Light Complexity

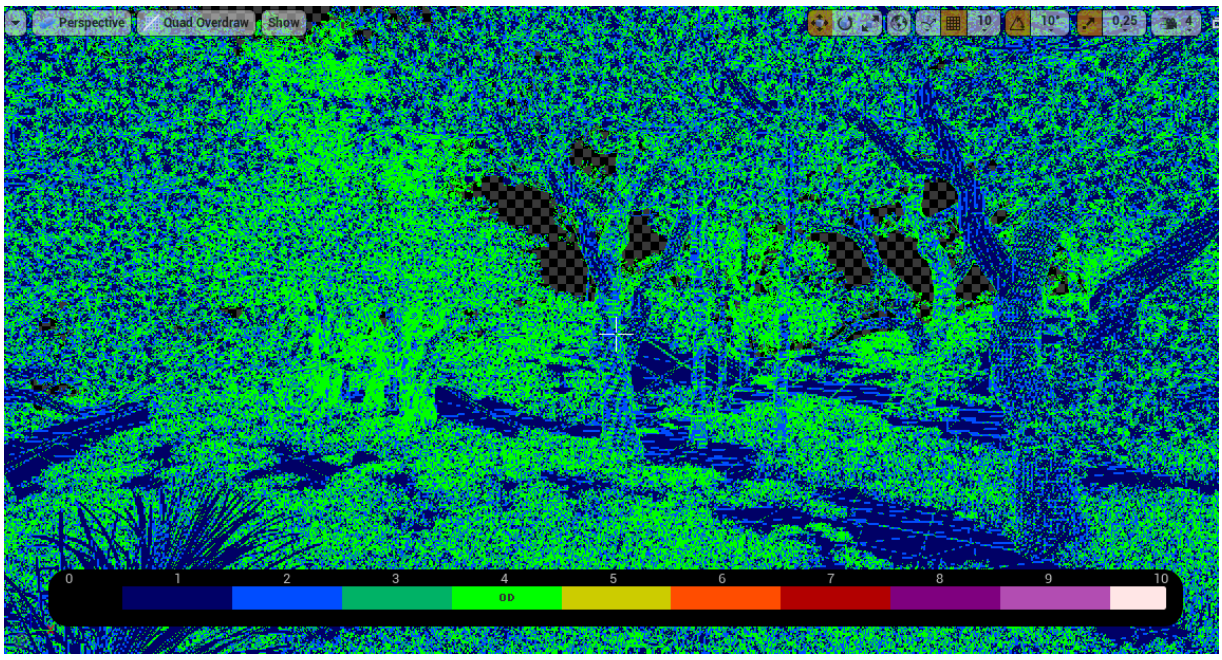


Рисунок 4.32 - Режим Quad Overdraw

На рисунку 4.32 режим Quad Overdraw. Коли декілька полігонів перекриваються і відображаються в одному квадраті на графічному блоці обробки, продуктивність сцени падає, оскільки вони обчислюються, а потім використовуються [25]. Сцена достатньо оптимізована і тому в цьому режимі діапазон кольорів правильний.

4.5 Аналіз отриманих результатів

Реалізовано покрокову стратегію на мові Blueprint в Unreal Engine 4, що містить процедурну генерацію ігрового середовища. В таблиці 4.1 показано аналіз за критеріями реалізованої гри з аналогами Heroes of Might and Magic та The Battle for Wesnoth.

Таблиця 4.1 – Порівняння гри з аналогами

Критерій	Реалізована гра	Heroes of Might and Magic	The Battle for Wesnoth
Швидке завантаження ігрового сеансу	+	-	-
Процедурна генерація	+	-	-
Сучасна графіка	+	-	-
Система покращення характеристик героя	+	+	+
Використання PBR текстур	+	-	-

В порівнянні з аналогами кваліфікаційна робота має реалізовані переваги, а саме за рахунок процедурної генерації має швидке завантаження ігрового сеансу та достатньо різноманітну рослинність. Текстури, що використано для процедурної генерації мають фізично корректний рендеринг за рахунок карт висот, нормалей та жорсткості, що дає переваги у графіці в порівнянні з аналогами. Гра має достатню оптимізацію графіки, що дає перевагу у комфортній грі.

Крім цього гра містить якісні анімації та ефекти, що робить її ще більш сучаснішою. Ігри Heroes of Might and Magic та The Battle for Wesnoth мають системи покращення, але в кваліфікаційній грі ці показники реалізовано більш глибоко, додано набагато більше характеристик (швидкість, захист, розум,

витривалість) для кожного персонажа, які покращуються за рахунок збільшення рівня та спеціальних предметів, яких не мають аналоги. Кожен персонаж має декілька видів атак, а також систему збільшення рівня за рахунок отримання досвіду.

Для кожного персонажа виведено показники здоров’я та мани на основний екран гри, що робить користувацький інтерфейс гри більш зручнішим та красивішим в порівнянні з Heroes of Might and Magic та The Battle for Wesnoth.

Висновки до розділу 4

Розроблено 3D гру на мові Blueprint у середовищі Unreal Engine 4. Створено ігровий режим та контролер гравця покрокової стратегії. Реалізовано механіки атаки та функції розрахунку пошкоджень. Створено та відредаговано спеціальні графічні ефекти для різних ігрових механік.

Реалізовано систему збереження прогресу за допомогою SaveGame, а також систему отримання винагород. Створено функції збільшення рівня, індикатори отриманого досвіду, перехід та збереження досвіду. Створено віджети інтерфейсу гравця.

Проведено тестування продуктивності за допомогою режимів в Unreal Engine 4: Shader Complexity, Light Complexity, Lightmap Density, Quad Overdraw. Ігрова сцена оптимізована і не навантажує систему, має рівномірну щільність текселів та рівномірне освітлення карти освітлення. Реалізована гра відповідає представленій специфікації вимог.

ВИСНОВКИ

Кваліфікаційна робота пов’язана з використанням технології розробки комп’ютерної 3D гри в жанрі покрокових стратегій на мові візуального програмування Blueprint на Unreal Engine 4 з процедурною генерацією ігрового середовища. На даний момент процедурна генерація дозволяє зменшити витрати, на кожному етапі розробки, тому що дозволяє зменшити кількість персоналу та збільшити різноманіття локацій, а також економить час. За рахунок цього, процедурна генерація є актуальною на сьогоднішній день.

Проведено аналіз предметної галузі та досліджено аналоги гри. Визначено, що основними недоліками аналогів є недостатньо розвинена система покращення характеристик юнітів, застаріла графіка та набридливий ракурс камери. Перевагою розроблюваної гри є система покращення характеристик героїв за рахунок системи досвіду гравця. Виявлено, що для жанру покрокових стратегій ніколи не використовувалась процедурна генерація.

Досліджено процедурну генерацію в іграх. Визначено, що процедурна генерація займає 80% сцени, а інші 20% дороблюються художником. Для створення процедурної генерації розглянуто декілька інструментів для створення ландшафту та карти висот, які створюються за рахунок змішування певних шумів.

Змодельовано та спроєктовано гру, представлено діаграми: варіантів використання, класів, станів, діяльності та послідовності. На основі цих діаграм представлено механіки ігрового процесу: систему покрокових ходів, послідовність виклику методів для атаки героїв, атаки ворога, атаки магією, меню апгрейдів, отримання досвіду та предметів. Розроблено мокапи інтерфейсу гри.

Розроблено 3D об’єкти рослинності для процедурної генерації. Створено high poly та low poly об’єкти, UV розгортку та атласи в Maya 3D. Процес фарбування текстур зроблено в Substance Painter. Ландшафт та

матеріал для нього зроблено в Unreal Engine 4. За допомогою інструменту Procedural Foliage Spawner для створення процедурної генерації заповнено ігровий рівень рослинністю. Процедурну генерацію імплементовано за допомогою інструментарію Foliage. Освітлення на сцені зроблено за рахунок: Directional Light, Sky Atmosphere, Sky Light. Додано туман за допомогою інструменту Exponential Height Fog. Пост обробку сцени зроблено за рахунок Post Process Volume та оптимізовано за рахунок системи LUT.

Для створення гри використано мову візуального програмування blueprint. Створено ігровий режим та контролер гравця покрокової стратегії. Реалізовано механіки атаки та функції розрахунку пошкоджень. Відредаговано та створено спеціальні графічні ефекти, для візуалізації під час використання механік за допомогою інструменту Niagara. Створено анімації для дій персонажів та їх взаємодії. Реалізовано систему збереження прогресу, а також систему отримання винагород. Створено функції збільшення рівня, індикатори отриманого досвіду та збереження досвіду. Створено віджети інтерфейсу гравця.

Проведено тестування продуктивності за допомогою режимів в Unreal Engine 4: Shader Complexity, Light Complexity, Lightmap Density, Quad Overdraw. Тестування продуктивності показало, що сцена достатньо оптимізована для комфортної гри.

Отже мета, що полягала у покращенні ігрового процесу за рахунок процедурної генерації ігрового середовища шляхом використанням системи Blueprint була досягнута.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Переваги Unreal Engine 4. URL: https://gamesacademy.com.ua/our-courses/unreal-engine/ (дата звернення: 10.09.2022).

2. Переваги Unreal Engine 4. URL: https://blog.artcraft.net.ua/page11435532.html (дата звернення: 11.09.2022).

3. Heroes of Might and Magic. URL: https://www.filfre.net/2021/12/heroes-of-might-and-magic/ (дата звернення: 13.09.2022).

4. Heroes of Might and Magic. URL: https://gamicus.fandom.com/wiki/The_Battle_for_Wesnoth (дата звернення: 13.09.2022).

5. Яшников Д. В., Кандиба І. О. ПРОЦЕДУРНА ГЕНЕРАЦІЯ ПРИ СТВОРЕННІ КОМП'ЮТЕРНИХ ІГОР.Інформаційні технології та інженерія : Всеукраїнська науковопрактична конференція молодих вчених, аспірантів і студентів : тези доп., 7–10 лютого 2023 р. / ЧНУ імені Петра Могили. Миколаїв, 2023. С.112-114.

6. Яшников Д. В., Кандиба І. О. Інструментарій процедурної генерації рушія UNREAL ENGINE 4. Могилянські читання – 2022, : Досвід та тенденції розвитку суспільства в Україні : глобальний, національний та регіональний аспекти : XXV Всеукр. наук.-практ. конф. : тези доповідей : Комп'ютерні науки. Технічні науки, Миколаїв, 7–11 листоп. 2022 р. С. 143-144.

7. Tessendorf J. Simulating ocean water. Simulating nature: realistic and interactive techniques. SIGGRAPH. 2001. VOL. 1, № 2. P. 1-26.

8. Simplex noise. URL: https://thebookofshaders.com/11/ (дата звернення: 28.09.2022).

9. Worley noise. URL: https://www.sidefx.com/docs/houdini/nodes/vop/worleynoise.html (дата звернення: 28.09.2022).

10. Gaea. URL: https://quadspinner.com/ (дата звернення: 02.10.2022).

11. World Creator features. URL: https://www.world-creator.com/features.html (дата звернення: 02.10.2022).

12. World Machine. URL: https://www.world-machine.com/ (дата звернення: 02.10.2022).

13. Foliage Unreal Engine 4. URL: https://docs.unrealengine.com/4.27/en-US/BuildingWorlds/Foliage/ (дата звернення: 02.11.2022).

14. Directional light. URL: https://docs.unrealengine.com/4.27/en-US/BuildingWorlds/LightingAndShadows/LightTypes/Directional/ (дата звернення: 10.11.2022).

15. Sky atmosphere. URL: https://docs.unrealengine.com/5.1/en-US/sky-atmosphere-component-in-unreal-engine/ (дата звернення: 10.11.2022).

16. Sky light. URL: https://docs.unrealengine.com/4.27/en-US/BuildingWorlds/LightingAndShadows/LightTypes/SkyLight/ (дата звернення: 10.11.2022).

17. Height Fog. URL: https://docs.unrealengine.com/4.27/en-US/BuildingWorlds/FogEffects/HeightFog/ (дата звернення: 10.11.2022).

18. Post Process Effects. URL: https://docs.unrealengine.com/4.27/en-US/RenderingAndGraphics/PostProcessEffects/UsingLUTs/ (дата звернення: 20.11.2022).

19. Sanders A. An introduction to Unreal engine 4. AK Peters/CRC Press, 2016. 270 p.

20. Creating visual effects in Niagara. URL: https://docs.unrealengine.com/5.1/en-US/creating-visual-effects-in-niagara-for-unreal-engine/ (дата звернення: 10.12.2022).

21. View modes. URL: https://docs.unrealengine.com/4.27/en-US/BuildingWorlds/LevelEditor/Viewports/ViewModes/ (дата звернення: 16.12.2022).

22. The cost of materials. URL: https://subscription.packtpub.com/book/game-development/9781789538540/1/ch01lvl1sec16/checking-the-cost-of-our-materials (дата звернення: 16.12.2022).

23. Computing Lightmap UV Coordinates. URL: https://docs.speedtree.com/doku.php?id=making_lightmap_uvs_for_ue4 (дата звернення: 17.12.2022).

24. Shannon T. Unreal Engine 4 for design visualization: Developing stunning interactive visualizations, animations, and renderings. Addison-Wesley Professional, 2017. 384 p.

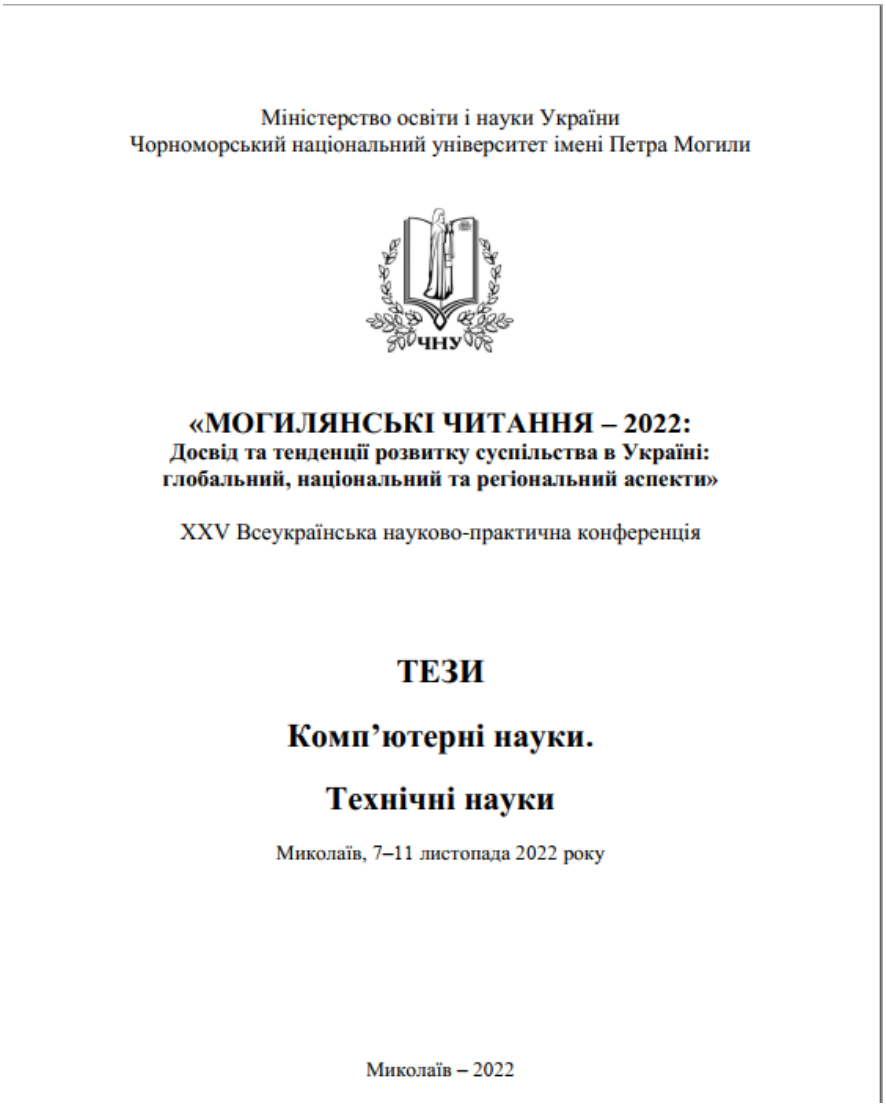
25. Map optimization. URL: https://squad.fandom.com/wiki/Map_Optimisation (дата звернення: 26.01.2022).

ДОДАТОК А

Апробація кваліфікаційної роботи

Результати досліджень були представлені на двох конференціях:

– Могилянські читання – 2022, : Досвід та тенденції розвитку суспільства в Україні : глобальний, національний та регіональний аспекти : XXV Всеукр. наук.-практ. конф. : тези доповідей : Комп’ютерні науки. Технічні науки, Миколаїв, 7–11 листоп. 2022 р. / ЧНУ ім. Петра Могили. – Миколаїв : Вид-во ЧНУ ім. Петра Могили, 2022.



Міністерство освіти і науки України
Чорноморський національний університет імені Петра Могили

ЧНУ

«МОГИЛЯНСЬКІ ЧИТАННЯ – 2022:
Досвід та тенденції розвитку суспільства в Україні:
глобальний, національний та регіональний аспекти»

XXV Всеукраїнська науково-практична конференція

ТЕЗИ

Комп’ютерні науки.

Технічні науки

Миколаїв, 7–11 листопада 2022 року

Миколаїв – 2022

Рисунок А.1 – Обкладинка збірника тез конференції Могилянські читання – 2022

може значно спростити процес виявлення шкідливого ПЗ та запобігти нанесенню ним шкоди.

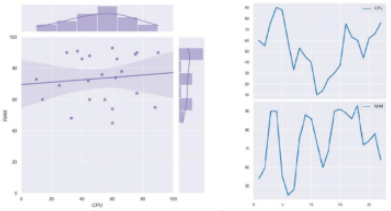


Рисунок 1 – Діаграми представлені за допомогою SEABORN

УДК 004.4

Яшников Д. В.,
магістрант спеціальності інженерії програмного забезпечення,
Кандиба І. О.,
PhD, старший викладач кафедри інженерії програмного забезпечення
ЧНУ ім. Петра Могили, м. Миколаїв, Україна

ІНСТРУМЕНТАРІЙ ПРОЦЕДУРНОЇ ГЕНЕРАЦІЇ РУШІЯ UNREAL ENGINE 4

Сучасні ігрові світи в іграх - це величезні простори з реалістичними лісами, полями, селищами, містами, стежками та багатополосними шосе. У нас час відкриті світи в іграх стали на стільки великими, що пророблять їх вручну дуже довго та дорого. Для цього все частіше використовуються спеціальні алгоритми. При створенні ігрового простору використовується принцип Парето – гравець активно досліджує лише 20 відсотків локації, а інші 80 служать фоном, Процедурна генерація допомагає заповнювати ці 80 відсотків, полегшуючи процес створення декорацій оточення художникам та геймдезайнери, щоб перевіряти механіки на начерку світу і на ходу додавати нові.

Логіка генерації реалізовується різними інструментами командою художників.Після чого додаються елементи та редагуються вже згенеровані.

Розробники The Witcher 3 збирались розробити світ, який в 35 разів більший ніж у другій частині, для цього було розроблено систему завдяки якій можна було легко розміщувати траву та інші елементи рослинності, для цього було створено алгоритм для розпізнавання природних та штучних поверхонь і створювати плавні переходи між ними при цьому процедурно згенерована трава не перекриває деталей вихідної текстури, а навпаки підкреслює їх. Для цього створили алгоритм для аналізу місць, де щось може рости, а саме трава могла рости лише там, де є земля. Щоб колір трави й текстури збігався алгоритм використовував градієнтну сітку в якій колір трави ближче до коріння мав колір землі, а зверху зберігався вхідний колір.

Для побудови міста в грі The Thinking City студія Frogwares створила інструмент процедурної генерації на Unreal Engine 4. Художники спочатку вручну створювали сітку (макет) міста і вказували які типи будинків могли знаходитись в тому чи іншому районі і коли сітка була завершена, то використовувалась процедурна генерація.

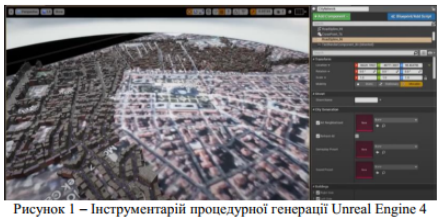


Рисунок 1 – Інструментарій процедурної генерації Unreal Engine 4

Процедурна генерація полегшує роботу не тільки художників по оточенню. Багато сучасних алгоритмів використовують ітеративний підхід, тому їх можна застосовувати на ранніх етапах розробки. Левел-дизайнери можуть використовувати ітеративну генерацію для створення наочний блокаутів, геймдизайнери – щоб перевіряти механіки, додаючи на ходу нові, оскільки відпадає необхідність висаджувати кожне дерево, замість цього створюється все автоматично.

Рисунок А.2 – Тези конференції Могилянські читання – 2022 – Інформаційні технології та інженерія: Всеукраїнська науковопрактична конференція молодих вчених, аспірантів і студентів : тези доп., 7–10 лютого 2023 р. / ЧНУ імені Петра Могили. Миколаїв, 2023.

Міністерство освіти і науки України
Чорноморський національний університет
імені Петра Могили

«Інформаційні технології та інженерія»

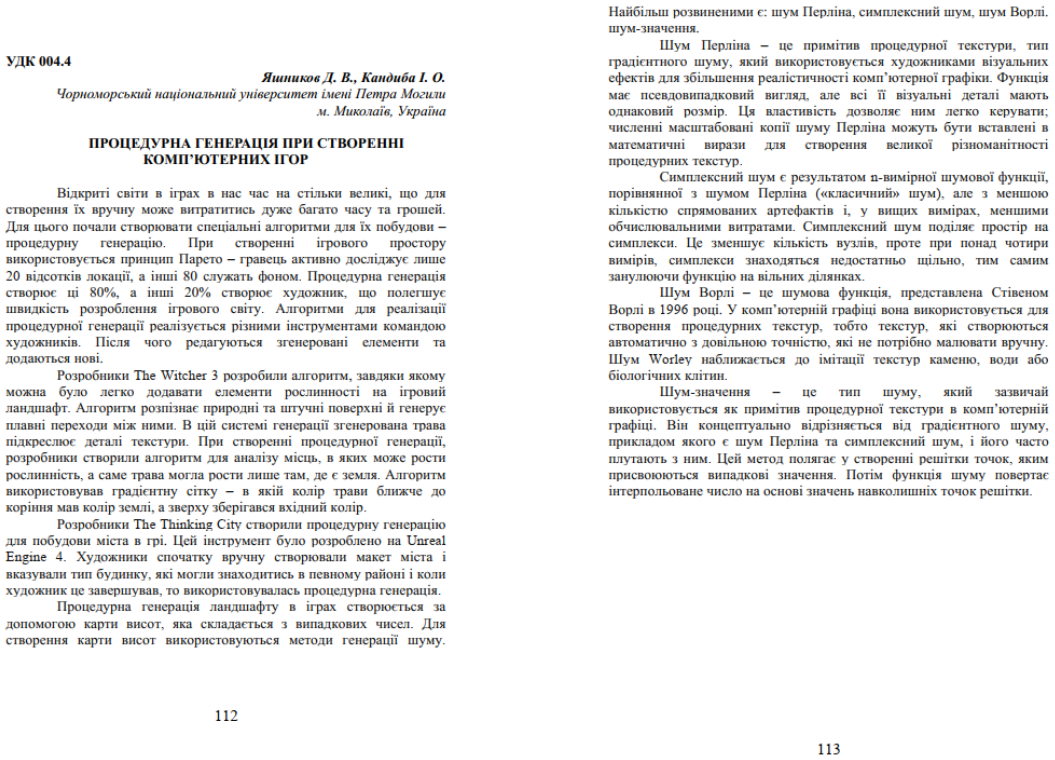
Всеукраїнська науково-практична конференція молодих вчених, аспірантів і студентів

ТЕЗИ

7–10 лютого 2023 року

Миколаїв – 2023

Рисунок А.3 – Обкладинка збірника тез конференції Інформаційні технології та інженерія – 2023



УДК 004.4

Яшников Д. В., Кандиба І. О.
Чорноморський національний університет імені Петра Могили
м. Миколаїв, Україна

ПРОЦЕДУРНА ГЕНЕРАЦІЯ ПРИ СТВОРЕННІ КОМП'ЮТЕРНИХ ІГОР

Відкриті світи в іграх в нас час на стільки великі, що для створення їх вручну може витратитись дуже багато часу та грошей. Для цього почали створювати спеціальні алгоритми для їх побудови – процедурну генерацію. При створенні ігрового простору використовується принцип Парето – гравець активно досліджує лише 20 відсотків локації, а інші 80 служать фоном. Процедурна генерація створює ці 80%, а інші 20% створює художник, що полегшує швидкість розроблення ігрового світу. Алгоритми для реалізації процедурної генерації реалізується різними інструментами командою художників. Після чого редагуються згенеровані елементи та додаються нові.

Розробники The Witcher 3 розробили алгоритм, завдяки якому можна було легко додавати елементи рослинності на ігровий ландшафт. Алгоритм розпізнає природні та штучні поверхні й генерує плавні переходи між ними. В цій системі генерації згенерована трава підкреслює деталі текстури. При створенні процедурної генерації, розробники створили алгоритм для аналізу місць, в яких може рости рослинність, а саме трава могла рости лише там, де є земля. Алгоритм використовував градієнтну сітку – в якій колір трави ближче до коріння мав колір землі, а зверху зберігався вхідний колір.

Розробники The Thinking City створили процедуру генерацію для побудови міста в грі. Цей інструмент було розроблено на Unreal Engine 4. Художники спочатку вручну створювали макет міста і вказували тип будинку, які могли знаходитись в певному районі і коли художник це завершував, то використовувалась процедурна генерація.

Процедурна генерація ландшафту в іграх створюється за допомогою карти висот, яка складається з випадкових чисел. Для створення карти висот використовуються методи генерації шуму.

Найбільш розвиненими є: шум Перліна, симплексний шум, шум Ворлі, шум-значення.

Шум Перліна – це примітив процедурної текстури, тип градієнтного шуму, який використовується художниками візуальних ефектів для збільшення реалістичності комп'ютерної графіки. Функція має псевдовипадковий вигляд, але всі її візуальні деталі мають однаковий розмір. Ця властивість дозволяє ним легко керувати; численні масштабовані копії шуму Перліна можуть бути вставлені в математичні вирази для створення великої різноманітності процедурних текстур.

Симплексний шум є результатом n-вимірної шумової функції, порівнянної з шумом Перліна («класичний» шум), але з меншою кількістю спрямованих артефактів і, у вищих вимірах, меншими обчислювальними витратами. Симплексний шум поділяє простір на симплекси. Це зменшує кількість вузлів, проте при понад чотири вимірів, симплекси знаходяться недостатньо щільно, тим самим занулюючи функцію на вільних ділянках.

Шум Ворлі – це шумова функція, представлена Стівеном Ворлі в 1996 році. У комп'ютерній графіці вона використовується для створення процедурних текстур, тобто текстур, які створюються автоматично з довільною точністю, які не потрібно малювати вручну. Шум Worley наближається до імітації текстур каменю, води або біологічних клітин.

Шум-значення – це тип шуму, який зазвичай використовується як примітив процедурної текстури в комп'ютерній графіці. Він концептуально відрізняється від градієнтного шуму, прикладом якого є шум Перліна та симплексний шум, і його часто плутають з ним. Цей метод полягає у створенні решітки точок, яким присвоюються випадкові значення. Потім функція шуму повертає інтерпольоване число на основі значень навколишніх точок решітки.

Рисунок А.2 – Тези конференції Інформаційні технології та інженерія – 2023