ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ АВТОМАТИЗОВАНИХ СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Дослідження можливостей використання алгоритму Perlin Noise для процедурного генерування ландшафту при створенні ігр»

на здобуття освітнього ступеня магістра
зі спеціальності 126 Інформаційні системи та технології
(код, найменування спеціальності)
освітньо-професійної програми Інформаційні системи та технології
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Данило ГЕРАЙМОВИЧ
(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав:
здобувач вищої освіти
група ІСДМ-61
Данило ГЕРАЙМОВИЧ

Керівник:
науковий ступінь, вчене звання
Ольга ПОЛОНЕВИЧ
к.т.н., доцент

Рецензент:
науковий ступінь, вчене звання
_____
Ім'я, ПРІЗВИЩЕ

Київ 2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення автоматизованих систем

Ступінь вищої освіти Магістр

Спеціальність Інформаційні системи та технології

Освітньо-професійна програма Інформаційні системи та технології

ЗАТВЕРДЖУЮ
Завідувач кафедрою ІПЗАС

_____ Каміла СТОРЧАК
«_____» _____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Гераймовичу Данилу Сергійовичу
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Дослідження можливостей використання алгоритму Perlin Noise для процедурного генерування ландшафту при створенні ігр.

керівник кваліфікаційної роботи Ольга ПОЛОНЕВИЧ к.т.н, доцент
(Ім'я, ПРІЗВИЩЕ науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «19» 10.2023р. №145

2. Строк подання кваліфікаційної роботи «29» грудня 2023р.

3. Вихідні дані до кваліфікаційної роботи: алгоритми генерування ландшафтів, Perlin Noise, Unreal Engine 5, Науково-технічна література.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Аналіз наявних алгоритмів генерування ландшафтів
Вимоги до розробки застусунків на Unreal Engine 5

5. Перелік графічного матеріалу: *презентація*
6. Дата видачі завдання«19» жовтня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Вибір теми кваліфікаційної роботи	19.10-05.11.23	виконав
2	Підбір науково-технічної літератури	06.11-12.11.23	виконав
3	Аналіз ринку систем контролю та обліку персоналу	13.11-19.11.23	виконав
5	Дослідження алгоритмів генерування ландашфту	19.11-03.12.23	виконав
6	Проектування застусуноку генерації ландшафта на основі алгоритму perlin noise	04.12-10.12.23	виконав
7	Реалізація застусуноку генерації ландшафта на рушії unreal engine 5	11.12-28.12.23	виконав
8	Подання роботи в деканат	29.12.23	виконав

Здобувач вищої освіти _____ Данило ГЕРАЙМОВИЧ
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник
кваліфікаційної роботи _____ Ольга ПОЛОНЕВИЧ
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 58 стор., 58 рис., 18 джерел.

Мета роботи – виявити можливості та обмеження використання алгоритму Perlin Noise для генерації ігрового ландшафту.

Об'єкт дослідження – алгоритми генерування ландшафту.

Предмет дослідження – характеристики та параметри алгоритмів генерування ландшафту.

Короткий зміст роботи:

Проаналізовано найбільш популярні алгоритми генерування ландшафтів, їх характеристики та параметри. На основі результатів виконаних досліджень, спроектовано та розроблено застусунок з використанням ігрового рушія Unreal engine 5.

КЛЮЧОВІ СЛОВА

PERNLIN NOISE, SIMPLEX NOISE, VISUAL STUDIO, RIDER, UNREAL UNGINE 5, UNREAL LINK, RIDER LINK, C++, BLUEPRINT, 2D, 3D.

ABSTRACT

Text part of the master's qualification work:
58 pages, 58 pictures ,18 sources.

The purpose of the work reveal the possibilities and limitations of using the Perlin Noise algorithm to create a game landscape.

Object of research – landscape generation algorithms.

Subject of research – characteristics and parameters of landscape generation algorithms.

Summary of the work:

The most popular landscape generation algorithms, their characteristics and parameters are analyzed. Based on the results of the research, the app was designed and developed using the Unreal engine 5 game engine.

KEYWORD

PERNLIN NOISE, SIMPLEX NOISE, VISUAL STUDIO, RIDER, UNREAL UNGINE 5, UNREAL LINK, RIDER LINK, C++, BLUEPRINT, 2D, 3D.

ЗМІСТ

ВСТУП

Актуальність. У сучасному світі відомих технологій та високорозвинених галузей розвитку інформаційних технологій, важливе місце посідає галузь розробки комп'ютерних ігор. Завдяки швидкому розвитку обчислювальної техніки та графічних технологій, виникає постійна потреба вдосконалення процесів розробки та оптимізації ігрових середовищ. Одним із ключових аспектів, який впливає на реалістичність та унікальність ігрового світу, є процедурне генерування ландшафту. Зі збільшенням ігрових світів у сучасних ігрових проектів. Процедруне генерування ландшафту може полегчити та зробити більш дешевшим створення ігрових світів.

Мета роботи - виявити можливості та обмеження використання алгоритму Perlin Noise для генерації ігрового ландшафту, розробити рекомендації щодо оптимального використання алгоритму у контексті розробки ігор та підняти питання можливого подальшого вдосконалення та розширення його застосувань.

В процесі роботи вирішувалися такі основні завдання:

- Дослідження алгоритмів які використовуються для створення ігрових ландшафтів;
- Проектування та розробка застусунку генерування ландшафту за допомогою рушія Unreal Engine 5.

Об'єкт дослідження – алгоритми генерування ландшафту.

Предмет дослідження – характеристики та параметри алгоритмів генерування ландшафту.

Методи дослідження. Під час виконання завдань магістерської кваліфікаційної роботи були використані методи теоретичного дослідження, елементів системного аналізу, методи дослідження теорії з використанням штучного інтелекту.

Наукова новизна одержаних результатів. Наукова новизна магістерської кваліфікаційної роботи, полягає у розробці застусунку на найновішому ігровому рушії та створення інструментарію, завдяки якому вдається спростити створення ландшафту для ігрових світів.

Практична значущість одержаих результатів. Практична значимість дослідження полягає у можливості застосування запропонованого рішення на практиці при створенні ігр.

Апробація результатів магістерської роботи. Основні положення і результати магістерської роботи доповідались і обговорювались науково-практичній конференції.

1 ДОСЛІДЖЕННЯ АЛГОРИТМІВ ГЕНЕРУВАННЯ ЛАНДАШФТУ

1.1 Види алгортмів генерування

Генерація одновимірних схилів - простий алгоритм полягає в тому, щоб створити групу випадкових синусодальних функцій, щоб контролювати висоту

пагорба. Алгоритм, який чудово підходить для генерації такого типу місцевості, називається зміщенням середньої точки. Він працює, починаючи з випадкових значень у двох точках, обчислюючи їх середнє значення, додаває випадковий шум, а потім рекурсивно повторює для середніх точок між початковими точками та серединою.

Клітинний автомат генерації печер - підхід клітинних автоматів базується на ідеї починати з випадкової сітки з 0 і 1, а потім багаторазово застосовуючи простий набір правил до кожної комірки, щоб отримати нову сітку, повторюючи

обробити кілька разів. Ці правила часто виглядають приблизно так: «Якщо клітинка зараз є проходом, а якщо в принаймні 6 її сусідів є стінами, тоді клітина стане стіною».

Voronoi Noise Biome Generation - шум Вороного (він же Шум Уорлі) заснований на випадковому розміщенні групи «seed» (насінні) в сітці, кожне з

який має випадково вибраний біом. Потім біом кожної комірки в сітці базується на найближчому насінні.

Алгоритм ромбовидного квадрата створює рельєф (тобто пагорби) за допомогою фрактального підходу. Це починається з деяких випадкових значень, розташованих один від одного, потім обчислює значення шуму між ними, беручи середні значення та додаючи трохи шуму. Процес повторюється багато разів, поки кожна клітинка не матиме значення шуму.

Perlin Noise — це складний алгоритм для генерації випадкової місцевості, яка все ще є гладкою/безперервною. Його досить важко змусити працювати, але результати приголомшливі. Найкраще те, що цей алгоритм працює добре з

процедурною генерацією, тобто вона створює завантажувальну частину світу, а потім, коли ви рухаєтеся до краю, ви можете створити більше рельєфу, і він буде легко з’єднуватися.

Simplex Noise — це варіант шуму Перліна, який використовує трикутники замість квадратної сітки для градієнта векторні місця розташування. Його краще зберегти для додаткових функцій, оскільки його надзвичайно складно реалізувати.

1.2 Алгоритм генерації одновимірних схилів

Одновимірна місцевість – це група висот, визначених у кожній точці вздовж осі X. Приклад цього можна побачити в таких іграх, як Flappy Bird, Galaxy та багатьох інших. Таку місцевість також можна простежити як силуети гірських хребтів (Рисунок 1.1).

Рисунок 1.1 – одновимірний ландшафт

Простим рішенням для створення випадкової місцевості є використання випадкової функції для кожної точки вздовж осі X. Випадкова функція видає випадкові значення в інтервалі [0, 1].

На рисунку 1.2 показано рельєф, створений за допомогою цієї процедури, має багато стрибків і різких змін висоти, і він явно не імітує рельєф у реальному світі. Рельєф реального світу, хоча й випадковий, не має великої кількості гострих шипів, натомість зміни висоти є дуже поступовими та забезпечують певну плавність.

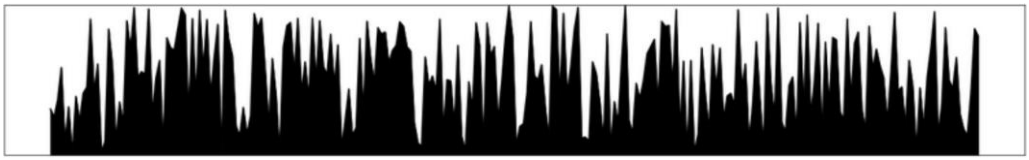


Рисунок 1.2 – ландшафт функції з випадковими значеннями

Для того, щоб забезпечити плавність терраїну, згенерованого наивним рельєфом, ми розглянемо відомий метод оцінювання під назвою інтерполяція, який оцінює проміжні значення, враховуючи групу відомих точок.

1.3 Клітинний автомат генерації печер

Клітинний автомат — дискретна обчислювальна модель, вперше відкрита в 1940-х роках. Фахівці з математики, фізики та біології всебічно її вивчали, і хоча вона породила гори складної математики, основна концепція дійсно проста.

За своєю суттю клітинний автомат складається з n-вимірної сітки з кількістю комірок у кінцевому стані та набором правил переходу. Кожна комірка сітки може перебувати в одному з кількох станів; у найпростішому випадку комірки можуть бути увімкненими або вимкненими.

Початковий розподіл станів клітинок становить зародок автомата в початковому стані. (Рисунок 1.3)

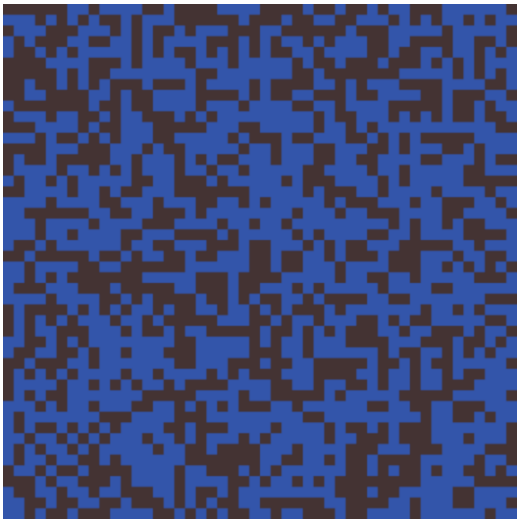


Рисунок 1.3 – Початковий розподіл станів клітин

Нове покоління створюється (збільшення t на 1) шляхом застосування правил переходу до всіх комірок одночасно, таким чином переводячи кожну комірку в новий стан з точки зору її поточного стану та поточних станів усіх комірок у її сусідстві. Околиці визначають, які клітини навколо даної клітини впливають на її майбутній стан. (Рисунок 1.4)

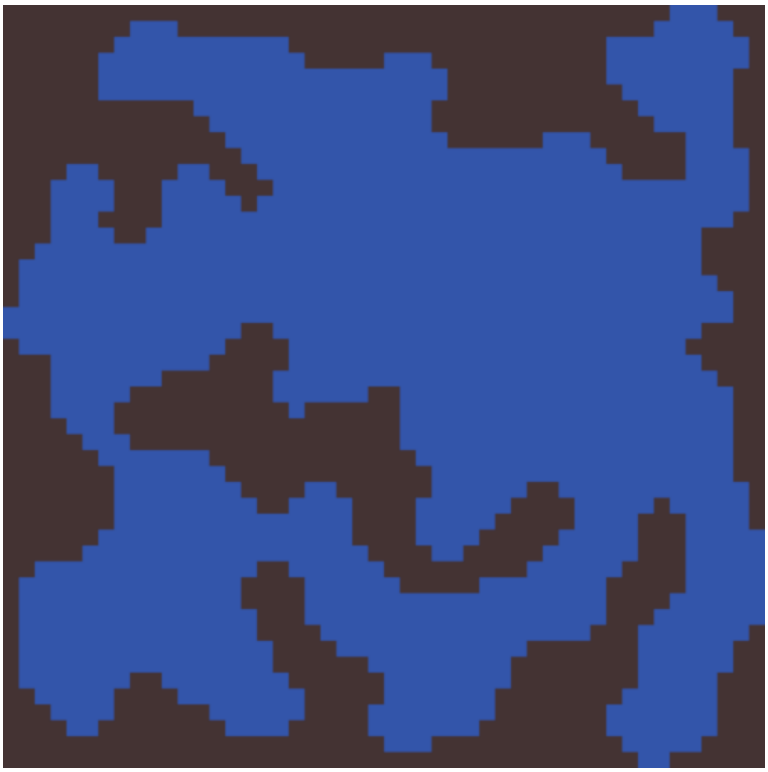


Рисунок 1.4 – Стан клітин після обробки алгоритмом

Приклад клітинних автоматів, добре відомий багатьом розробникам ігор — гра життя Конвея, яка являє собою двовимірну сітку клітинок, кожна з яких перебуває в стані мертвих або живих.

Перевагами алгоритму є:

- Процедурногенерація, тому немає двох (імовірно) однакових рівнів.
- Відносно проста концепція та реалізація.
- Природна карта, схожа на печеру, щоб додати різноманітності, унікальності або альтернативи кімнатним підземеллям.

До недоліків можна віднести:

- Створення ізольованих печерних секцій, які, можливо, блокують просування гравця.
- Алгоритм не підходить для створення міст, тому, ймовірно, потрібно буде запровадити інші генератори карт.
- Дуже великі карти, як правило, виглядають не так природно, або можуть вимагати дуже тонкого налаштування.

1.4 Шум Вороного

Шум Вороного є функцією шуму, представленою Стівеном Ворлі в 1996 році. Шум Вороного є розширенням діаграми Вороного, яка виводить дійсне значення за заданою координатою, що відповідає відстані n-го найближчого зерна , як правило, найближче насіння, і насіння рівномірно розподіляється по регіону.

У математиці діаграма Вороного — це розбиття площини на області, близькі до кожного з даного набору об’єктів. Його також можна класифікувати як тесселяцію. У найпростішому випадку ці об’єкти являють собою лише скінченну кількість точок на площині (так звані насіння, сайти або генератори). Для кожного зерна існує відповідна область, яка називається коміркою Вороного, що складається з усіх точок площини, ближчих до цього зерна, ніж до будь-якої іншої. Діаграма Вороного набору точок двоїста до тріангуляції Делоне цього набору.

Переваги алгоритму у тому що, його можна комбінувати з іншими алгоритмами, для створення ландшафту який поділений на біоми.

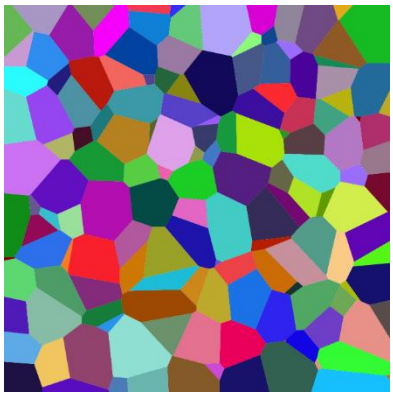


Рисунок 1.5 - Зріз діаграми Вороного випадкового набору точок у 3D

1.5 Алгоритм ромбовидного квадрата

Алгоритм ромбовидного квадрата починається з двовимірного квадратного масиву шириною та висотою 2n + 1. Для чотирьох кутових точок масиву спочатку потрібно встановити початкові значення. Потім кроки ромба та квадрата виконуються по черзі, доки не буде встановлено всі значення масиву.

Ромбовий крок: для кожного квадрата в масиві встановіть середину цього квадрата як середнє значення чотирьох кутових точок плюс випадкове значення.

Квадратний крок: для кожного ромба в масиві встановіть середину цього ромба як середнє значення чотирьох кутових точок плюс випадкове значення.

Кожне випадкове значення множиться на постійну масштабу, яка зменшується з кожною ітерацією на коефіцієнт 2-h, де h є значенням від 0,0 до 1,0 (нижчі значення створюють більш нерівний рельєф).

Під час квадратних кроків точки, розташовані на краях масиву, матимуть лише три суміжні значення, а не чотири. Існує кілька способів впоратися з цим ускладненням - найпростішим є взяти середнє лише трьох суміжних значень. Іншим варіантом є «обгортання», беручи четверте значення з іншого боку масиву. При використанні з узгодженими початковими кутовими значеннями цей метод також дозволяє з’єднати згенеровані фрактали без розривів.

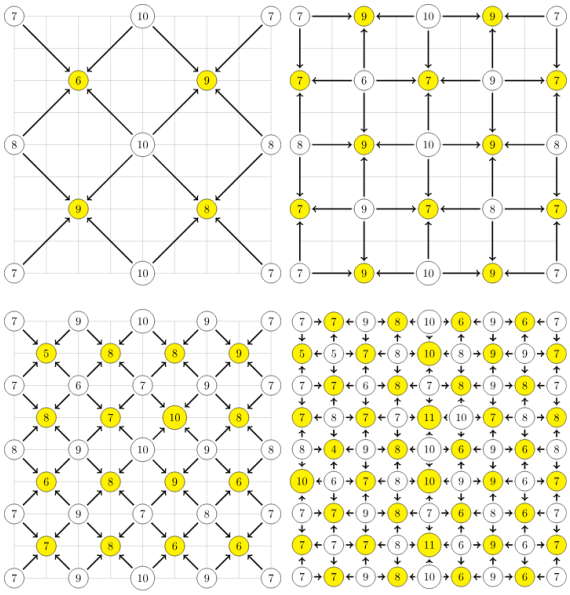


Рисунок 1.6 – Демонстація алгоритму ромбовидного квадрата

1.6 Алгоритм Perlin Noise

Шум Перліна — це примітив процедурної текстури, тип градієнтного шуму, розроблений Кеном Перліном у 1983 році, 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. Синтетичні текстури з використанням шуму Перліна часто використовуються в CGI, щоб зробити згенеровані комп'ютером візуальні елементи, такі як поверхні об'єктів, вогонь, дим або хмари, більш природними, імітуючи контрольовану випадкову появу текстур у природі.

Шум також часто використовується для генерації текстур, коли пам'ять надзвичайно обмежена, наприклад, у демонстраціях. Його наступники, такі як фрактальний шум і симплексний шум, стали майже повсюдними в графічних процесорах як для графіки в реальному часі, так і для процедурних текстур не в реальному часі у всіх видах комп'ютерної графіки.

Також, його часто використовується у відеоіграх, щоб зробити процедурно згенеровану місцевість, яка виглядає природно.

Реалізація зазвичай включає три етапи: визначення сітки випадкових градієнтних векторів, обчислення скалярного добутку між градієнтними векторами та їх зміщеннями та інтерполяцію між цими значеннями.

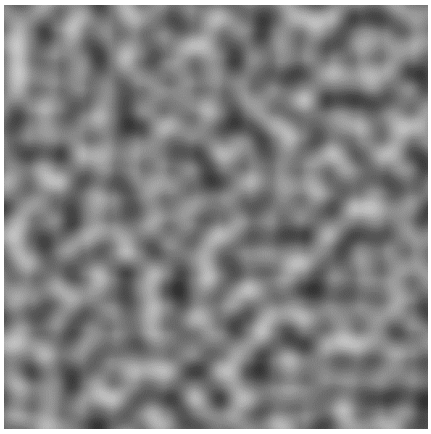


Рисунок 1.7 - Двовимірний зріз тривимірного шуму Перліна при z = 0

1.7 Алгоритм Simplex Noise

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. Кен Перлін розробив алгоритм у 2001 році, щоб усунути обмеження його класичної функції шуму, особливо у вищих вимірах.

Переваги симплексного шуму над шумом Перліна:

- Симплексний шум має меншу обчислювальну складність і потребує менше множень.
- Симплексний шум масштабується до вищих розмірів (4D, 5D) з набагато меншими обчислювальними витратами
- Симплексний шум не має помітних спрямованих артефактів (є візуально ізотропним), хоча шум, створений для різних вимірів, візуально відрізняється (наприклад, 2D-шум має інший вигляд, ніж 2D-фрагменти 3D-шуму, і він виглядає ще гірше для вищих вимірів).
- Симплексний шум має чітко визначений і безперервний градієнт (майже) всюди, який можна обчислити досить дешево.
- Симплексний шум легко реалізувати в апаратному забезпеченні.

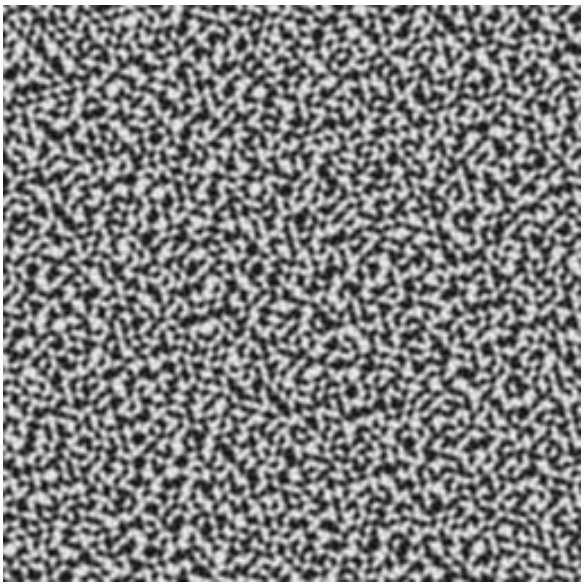


Рисунок 1.8 – Симплексний Шум

У той час як класичний шум інтерполює між градієнтами в навколишніх кінцевих точках гіперсітки, симплексний шум ділить простір на симплекси Це зменшує кількість точок даних.

Трикутники є рівносторонніми у 2D, але у вищих вимірах симплекси лише приблизно правильні. Наприклад, мозаїка в 3D-випадку функції є орієнтацією тетрагональної дисфеноїдної соти.

Симплексний шум корисний для додатків комп'ютерної графіки, де шум зазвичай обчислюється в 2, 3, 4 або, можливо, 5 вимірах. Для вищих розмірів n-сфери навколо n-симплексних кутів недостатньо щільно упаковані, що зменшує підтримку функції та робить її нульовою у великих частинах простору.

1.8 Приклади ігр з процедурної генерацією ландшафтів

Багато ігор процедурно генерують аспекти середовища або неігрових персонажів під час процесу розробки, щоб заощадити час на створенні активів. Наприклад, SpeedTree — це пакет проміжного програмного забезпечення, який процедурно генерує дерева, які можна використовувати для швидкого заповнення лісу. У той час як більшість ігор використовують цю техніку для створення статичного середовища для кінцевого продукту, деякі використовують процедурну генерацію як ігрову механіку, наприклад для

створення нових середовищ для дослідження гравцем. Рівні в Spelunky генеруються процедурно шляхом перегрупування готових плиток геометрії в рівень із входом, виходом, розв’язаним шляхом між ними та перешкодами на цьому шляху.Інші ігри процедурно генерують інші аспекти ігрового процесу, такі як зброя в Borderlands, яка має рандомізовані статистичні дані та конфігурації.

Starbound — це пригодницька відеогра, розроблена та видана Chucklefish. Події Starbound відбуваються у двовимірному, процедурно згенерованому всесвіті, який гравець може досліджувати, щоб отримати нову зброю, броню та предмети, а також відвідати міста та села, населені різноманітними розумними формами життя.

Рисунок 1.9 – Гра Starbound

Є чотири можливі розміри для кожного світу. Розмір планети впливає не тільки на її окружність, але й на відстань до ядра магми, а також на відстань до верхніх шарів атмосфери (космічного шару). Планети бувають 3 розмірів, а їхні супутники бувають 4, менших розмірів. Усі світи, крім Безпліддя, Місяця та Щупальця, мають однакові основні дані, коли справа доходить до визначення розміру планети.

Ось деякі факти, які стосуються всіх створенних світів:

- Космічний шар становить приблизно 1100 блоків над поверхнею планет. Розмір цього шару змінюється, але він дозволяє гравцям будувати серед астероїдів. Гравцям також доступний інший світ Asteroid Field, який навіть більший, ніж космічний шар на планетах.
- Усі планети породжують великих літаючих монстрів, створених процедурно, у шарі атмосфери, приблизно в 500 блоках над поверхнею планети.
- Кожен вторинний біом, створений на планеті, займає 50-80% площі, охопленої первинним біомом.
- Кожен суббіом, створений для первинного або вторинного біома, охоплює 25-40% площі, яку покривав би батьківський біом, якби він не створив суббіоми.
- Підповерхня (найнижча точка, з якої можна телепортуватися) закінчується на 100 блоків нижче дна поверхні планети.

У Minecraft гравці досліджують блоковий, піксельний, процедурно згенерований тривимірний світ із практично нескінченною місцевістю. Гравці можуть знаходити та добувати сировину, створювати інструменти та предмети, будувати споруди, земляні роботи та машини.

Рисунок 1.10 – Гра Minecraft

Генерація світу Minecraft не зовсім випадкова, тому що кожен згенерований світ починається з початкового номера. При створенні світу, є можливість ввести зерно або дозволити Minecraft вибрати зерно випадково. Ці числа поміщаються в «генератор псевдовипадкових чисел». Це алгоритм, який генерує списки чисел, наближених до випадкових чисел.

Ці псевдовипадкові числа є основою створення світу Minecraft. Вони використовуються для обчислення чисел для процесу, який називається «процедурна генерація». Цей процес алгоритмічно створює дані для створення текстур і великомасштабної 3D-комп'ютерної графіки. Це стосується багатьох комп'ютерних ігор, включаючи Minecraft.

Підсумовуючи:

- Вводиться початковий номер.
- Це початкове число вводиться в генератор псевдовипадкових чисел.
- Математичні дані створюються з зерна, які використовуються як координати та місцезнаходження.
- Оскільки всі дані надходять із початкового номера, ідентичне насіння створить ідентичний світ.

Valheim — це гра на виживання з відкритим світом, у яку грають від третьої особи. Гравці можуть створити світ, який процедурно генерується з вихідного коду карти. Кожен світ розділений на кілька біомів, таких як луки, Шварцвальд, болота, гори, рівнини, океани, Туманні землі, Глибока Північ і Попелищі. Кожен біом має своїх унікальних ворогів, предметів і босів, які змінюють те, наскільки складно там вижити. Valheim працює за денним і нічним циклом у грі.

Рисунок 1.11 – Гра Valheim

Насіння світу — це напіввипадково згенерований текст під час створення світу, який використовується для процедурної генерації світу у Valheim. Можна вручну ввести початкове значення світу або дозволити грі вибрати його для вас під час створення світу.

Також можете генерувати світи онлайн, використовуючи веб-сайт https://valheim-map.world/ , створений wd40bomber7. Можна легко переглядати всі місця розташування босів, гробниці та інші важливі наземні знаки. Цей генератор знаходиться на дуже ранній стадії розробки.

Enter the Gungeon — гра розроблена Dodge Roll і видана Devolver Digital. Геймплей, який розгортається в Gungeon на тему вогнепальної зброї, розповідає про чотирьох персонажів гравців на ім’я Gungeoneers, які перетинають процедурно згенеровані кімнати, щоб знайти зброю, яка може «вбити минуле». Gungeoneers борються з ворогами у формі куль, з якими борються як звичайними, так і екзотичними видами зброї.

Рисунок 1.12 – Гра Enter the Gungeon

Гравець починає зі входу, пробігає через скрині та крамаря, щоб зібрати предмети, а потім, перемагає боса. Для швидкого повернення в кімнати можна користуватися телепортами. Очевидно, що окремі кімнати створені вручну і населені різними ворогами, яких треба розстрілювати. При просуванні до наступних рівнів гри гравець бачить той самий патерн знову і знову, тільки етапи стають більшими.

Особливість генератора стає помітною, коли ви починаєте грати. Рівні здаються трохи спланованішими, ніж очікується від випадковості. Кімната з босом завжди знаходиться на розумній відстані від початку. Кімнати з ворогами завжди розумно перемежовуються спокійними кімнатами, лавками та перехрестями. І найважливіше — багато скриньок розташовані за петлями з односторонньою прохідністю.

Terraria — це пригодницька гра-пісочниця 2011 року, розроблена Re-Logic. Гра спочатку була випущена для Windows, а потім була перенесена на інші платформи ПК і консолей. У грі представлено дослідження, створення, будівництво, малювання та бої з різноманітними істотами в процедурно згенерованому 2D-світі. Terraria отримала загалом позитивні відгуки та продала

понад 44 мільйони копій до 2022 року, що зробило її однією з найбільш продаваних відеоігор.

Рисунок 1.13 – Terraria

Гра починається в процедурно згенерованому світі, коли гравці починають із базовими інструментами та керівництвом для неігрових персонажів, щоб розпочати й привернути їхню увагу до аспектів гри та прогресу. Світ гри складається з кількох шарів плиток, з якими гравці можуть взаємодіяти та змінювати. Багато ресурсів, таких як металеві руди, можна знайти під час дослідження печер. Гравці починають із низьким рівнем здоров'я та мани, які можна збільшити, знаходячи та створюючи певні предмети. Деякі ресурси можна знайти лише в певних областях карти, зберігати в звичайних і рідкісних контейнерах або викинути певними ворогами. Гравці використовують ресурси для створення нових предметів і обладнання. Різні рецепти вимагають різних ресурсів і крафтових станцій. Кілька предметів у Terraria створюють складні дерева ремесла, які включають велику кількість предметів для створення одного потужного спорядження.

2 ПРОЕКТУВАННЯ ЗАСТУСУНОКУ ГЕНЕРАЦІЇ ЛАНДШАФТА НА ОСНОВІ АЛГОРИТМУ PERLIN NOISE

2.1 Вибір платформи для розробки застусунку генерації 3D ландшафту

На сьогодніший день інструментарій для роботи з динамічним 3D простором є дуже різноманітний. Найкращі представники программного забезпечення які можуть надати функціонал для створення та редагування тривимірного простору є ігрові рушії та ПО для створення візуальних ефектів. При створенні ігр частіше вього використовують ігрові рушії, найбільш популярними і безкоштовними з яких є Unity3D та Unreal Engine.

Unity3D - міжплатформне середовище розробки комп'ютерних ігор. Unity3D дозволяє створювати програми, що працюють під більш ніж 20 різними операційними системами, що включають персональні комп'ютери, ігрові консолі, мобільні пристрої, інтернет-програми та інші. Випуск Unity3D відбувся у 2005 році і з того часу триває постійний розвиток.

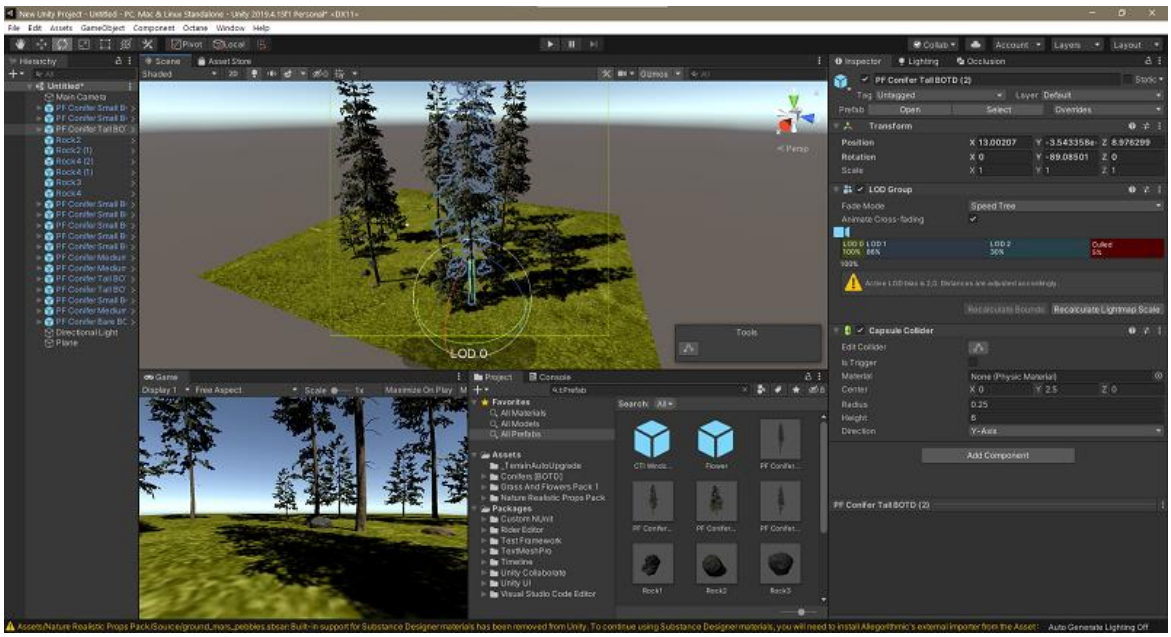


Рисунок 2.1 – Ігровий рушій Unity3D

Основними перевагами Unity3D є наявність візуального середовища

розробки, міжплатформної підтримки та модульної системи компонентів. До недоліків відносять появу складнощів при роботі з багатокомпонентними схемами та утруднення при підключенні зовнішніх бібліотек.

Ігровий двигун надає безліч функціональних можливостей, що дозволяють їх задіяти в різних іграх, до яких входять моделювання фізичних середовищ, карти нормалей, динамічні тіні та багато іншого. На відміну від багатьох ігрових движків, у Unity3D є дві основні переваги: наявність візуального середовища розробки та міжплатформова підтримка. Перший фактор включає не тільки інструментарій візуального моделювання, а й інтегроване середовище, ланцюжок складання, що спрямоване на підвищення продуктивності розробників, зокрема етапів створення прототипів та тестування. Під міжплатформною підтримкою надається не тільки місця розгортання (установка на персональному комп'ютері, на мобільному пристрої, консолі тощо), але й наявність інструментарію розробки (інтегроване середовище може використовуватись під Windows та Mac OS). Третьою перевагою називається модульна система компонентів Unity, за допомогою якої відбувається конструювання ігрових об'єктів, коли останні є комбінованими пакетами функціональних елементів. На відміну від механізмів успадкування, об'єкти в Unity створюються за допомогою об'єднання функціональних блоків, а не приміщення у вузли дерева успадкування. Такий підхід полегшує створення прототипів, що є актуальним при розробці ігор.

Рисунок 2.2 – Список платформ доступних з Unity3D

Як недоліки наводяться обмеження візуального редактора під час роботи з багатокомпонентними схемами. Другим недоліком називається відсутність підтримки Unity посилань на зовнішні бібліотеки, роботу з якими програмістам доводиться налаштовувати самостійно, і це також ускладнює командну роботу. Ще один недолік пов'язаний із використанням шаблонів екземплярів (англ. prefabs). З одного боку, ця концепція Unity пропонує гнучкий підхід візуального редагування об'єктів, але з іншого боку, редагування таких шаблонів є складним. Також, WebGL-версія движка, в силу специфіки своєї архітектури (трансляція коду з C# в C++ і далі JavaScript), має ряд невирішених проблем з продуктивністю, споживанням пам'яті і працездатністю на мобільних пристроях.

Unreal Engine 5 (UE5) — остання версія Unreal Engine, одного з найпотужніших і найпопулярніших ігрових движків. Unreal містить такі функції: Nanite, Lumen, World Partition system, MetaSounds та анімації.

Unreal Engine 4 вже відомий тим, що дозволяє створювати відкриті світи. Але UE5 виводить це на наступний рівень, прискорюючи створення великомасштабного світу та спрощуючи співпрацю.

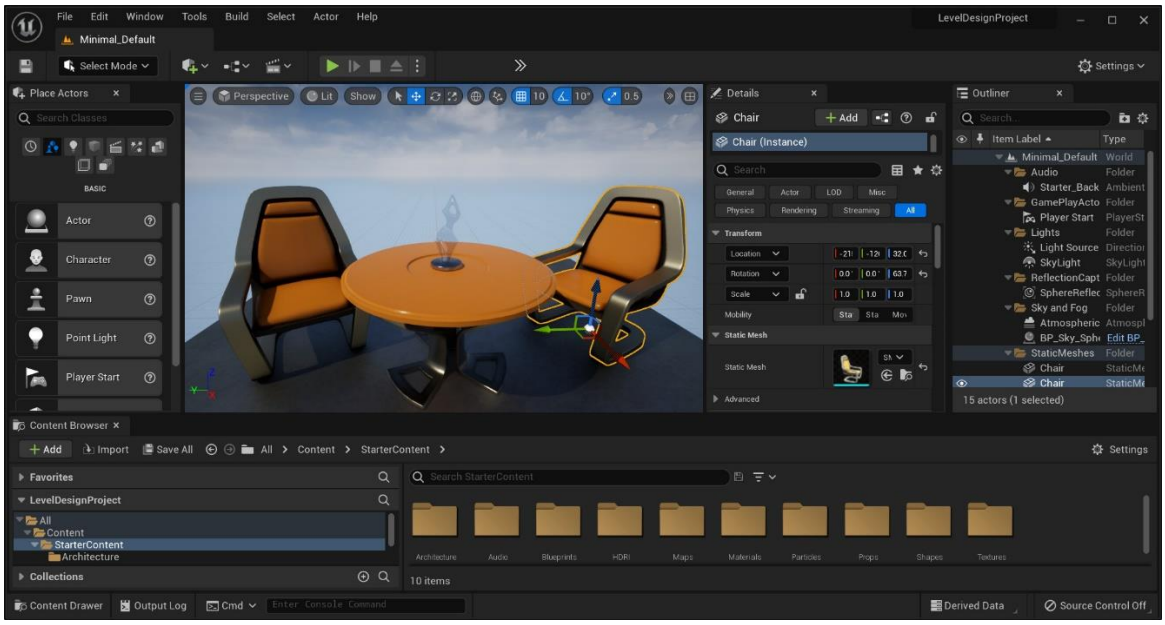


Рисунок 2.3 – Ігровий рушій Unreal Engine 5

Система World Partition використовує сітку для відображення підрівнів цілого всесвіту. Ви можете керувати складними рівнями, які завантажуються та розвантажуються, коли гравець йде по ландшафту. Крім того, система One File Per Actor допомагає командам працювати паралельно. Це зменшує перекриття, зберігаючи дані як зовнішні файли для кожного учасника.

Рисунок 2.4 – приклад World Partition у грі Fortnite

Технологія Unreal Engine 5 має вирішальне значення для консолей наступного покоління. Він також став критично важливим у створенні простих, потужних і масштабованих тривимірних світів для різноманітних галузей, включаючи архітектуру, трансляцію, події в прямому ефірі, автомобільну промисловість і транспорт, кіно та телебачення.

Рушій дозволяє збільшувати розміри файлів — і використовувати текстури 4K, 8K і 12K. Скажімо, гравець на ігровий об’єкт. Він відтворюється в 4K. Але коли гравець збільшує зображення, роздільна здатність динамічно змінюється. І гравець бачить об'єкт у 12K.

Ця функція неймовірна. І UE5 робить це можливим, обробляючи більші розміри файлів. Оскільки команди працюють над створенням ігор для консолей наступного покоління, таких як PlayStation 5 і Xbox Series S, їм доведеться мати можливість обробляти великі файли.

Також у UE5 впроваджений концепт «програмування без коду» - система Blueprints, що дозволяє швидко розробити прототип застусунку, без написання коду. Система візуальних сценаріїв Blueprint у Unreal Engine — це повна система сценаріїв ігрового процесу, заснована на концепції використання інтерфейсу на основі вузлів для створення елементів ігрового процесу в Unreal Editor. Як і в багатьох поширених мовах сценаріїв, він використовується для визначення об’єктно-орієнтованих (ОО) класів або об’єктів у механізмі.

Ця система є надзвичайно гнучкою та потужною, оскільки надає дизайнерам можливість використовувати практично повний спектр концепцій та інструментів, які зазвичай доступні лише програмістам. Крім того, спеціальна розмітка Blueprint, доступна в C++ реалізації Unreal Engine, дозволяє програмістам створювати базові системи, які можуть бути розширені дизайнерами.

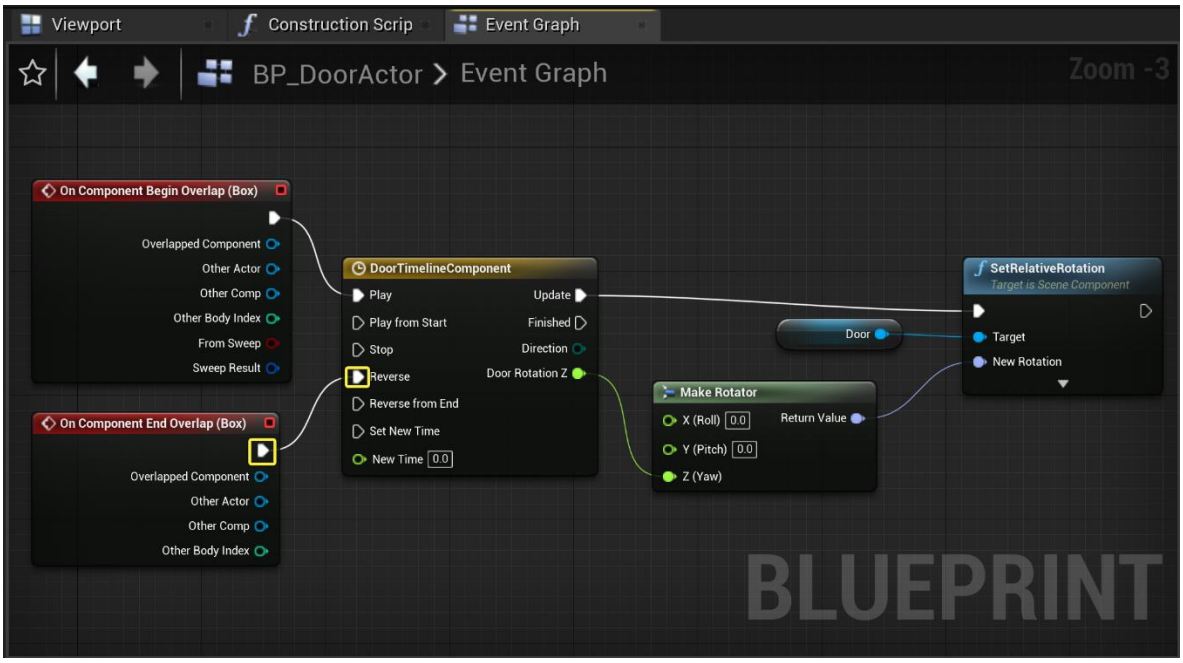


Рисунок 2.5 – приклад Blueprints

Однією із переваг Unreal Engine над Unity3D є відкритість вихідного коду. З Unreal Engine 4 вихідний код є повністю відкритий для налаштування та модифікування основного ігрового рушія. Це допомагає розробникам більш гнучкіше налаштовувати ігровий рушій під контретний проект або групи проектів. Вихідний код кожної актуальної версії рушія можна знайти у посиланні на GitHub репозиторій, якщо ви є зареєстрованим користовачем Epic Games. У Unity3D розробники обмежені архітекрутую рушія. Але модифікування рушія можливо тільки за допомогою плагінів.

Досліджуючи актуальні ігрові рушії, обираємо той який предаставляє новіші технології для актуального покоління консолей та ПК. Unreal Engine 5 найбільш підходить для задачі роботи.

2.2 Вибір середовища розробки

Ігровий рушій UE5 підтримує мову програмування C++. Тому вибір інтегрованно середовища розробки включає тільки ті, що підтримують C++ 11-ї і вище версії. Найпопулярніші IDE з можливостями написання коду на C++ є Microsoft Visual Studio, Ride. Також для редагування коду використовують

текстовий редактор Visual Studio Code.

Інтегроване середовище розробки Visual Studio є творчим стартовим майданчиком, який можна використовувати для редагування, налагодження та складання коду, а також для публікації програми. На додаток до стандартного редактора та відладчика, що надаються більшістю інтегрованих середовищ розробки, Visual Studio включає компілятори, засоби завершення коду, графічні конструктори та багато інших функцій для покращення процесу розробки програмного забезпечення.

Редактор коду у Visual Studio також підтримує встановлення закладок у коді для швидкої навігації. Інші навігаційні засоби включають згортання блоків коду та інкрементний пошук, на додаток до звичайного текстового пошуку та пошуку за регулярними виразами. Редактор коду також містить багатоелементний буфер обміну та список завдань. Редактор коду підтримує фрагменти коду, які є збереженими шаблонами для повторюваного коду, які можна вставляти в код і налаштовувати для проекту, над яким працюєте. Також вбудовано інструмент керування для фрагментів коду. Ці інструменти відображаються у вигляді плаваючих вікон, які можна налаштувати на автоматичне приховування, коли вони не використовуються, або прикріплені збоку від екрана. Редактор коду в Visual Studio також підтримує рефакторинг коду, включаючи зміну порядку параметрів, перейменування змінних і методів, вилучення інтерфейсу та інкапсуляцію членів класу всередині властивостей, серед іншого.

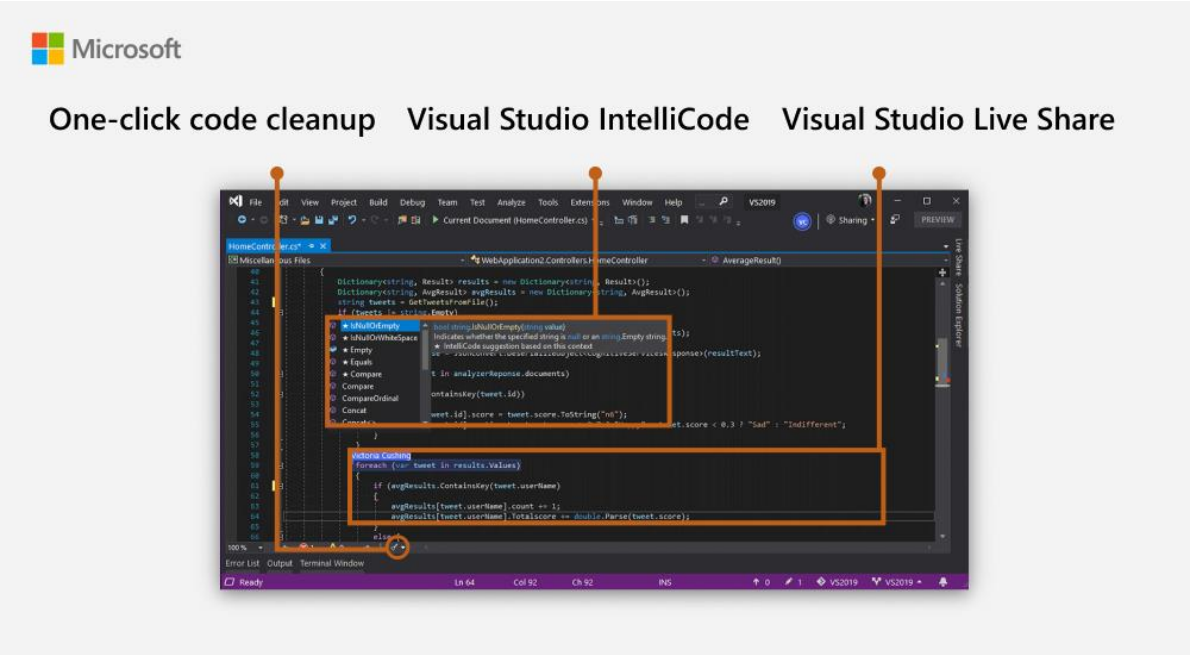


Рисунок 2.6 – Visual Studio IDE

Visual Studio містить налагоджувач, який працює як налагоджувач на рівні джерела, так і на рівні машини. Він працює як з керованим кодом, так і з рідним кодом і може використовуватися для налагодження програм, написаних будь-якою мовою, що підтримується Visual Studio. Крім того, він також може приєднуватися до запущених процесів, контролювати та налагоджувати ці процеси. Якщо вихідний код для запущеного процесу доступний, він відображає код під час виконання. Якщо вихідний код недоступний, він може показати розбирання. Налагоджувач Visual Studio також може створювати дампи пам'яті, а також завантажувати їх пізніше для налагодження. Також підтримуються багатопотокові програми. Налагоджувач можна налаштувати на запуск, коли програма, яка працює поза середовищем Visual Studio, аварійно завершує роботу.

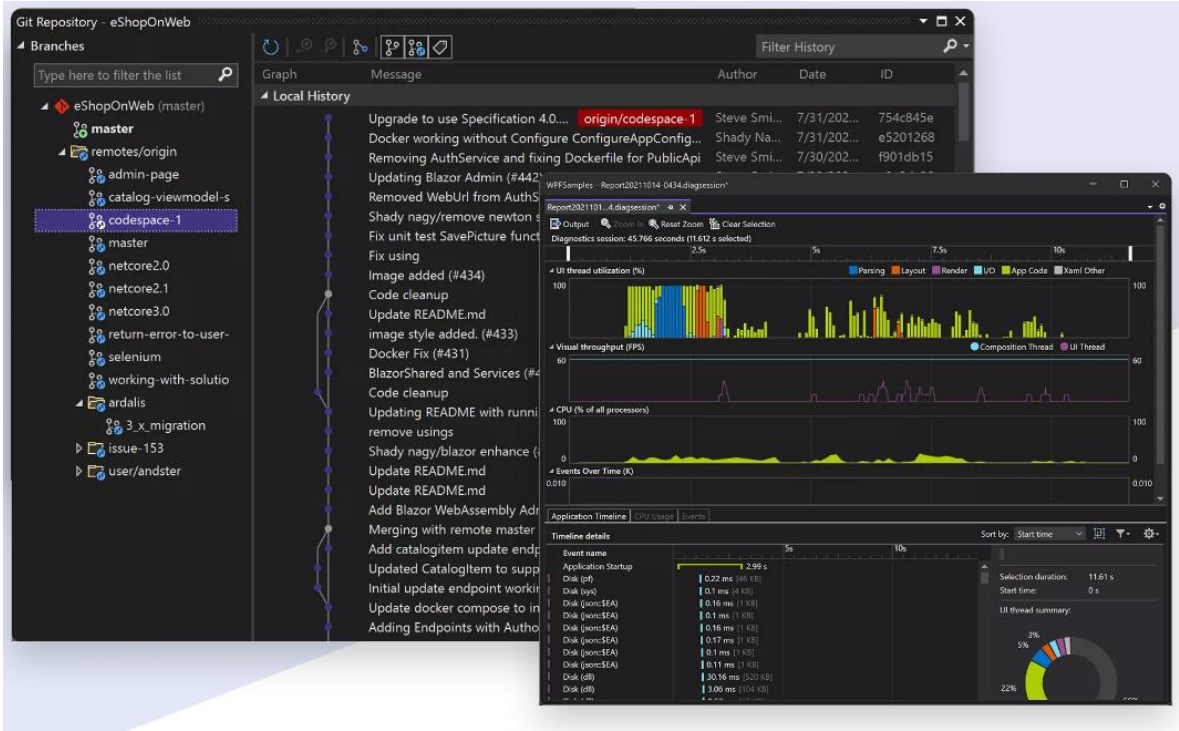


Рисунок 2.7 - Включенні у IDE іструменти розробки

Rider — це крос-платформена IDE для .NET-розробників, заснована на платформі IntelliJ і ReSharper. Rider надає більше 2200 інспекцій коду та автоматизованих виправлень для усунення виявлених проблем як в індивідуальному, так і в масовому порядку. Механізм аналізу помилок по всьому вирішенню шукатиме помилки в кодовій базі та повідомлятиме про них, навіть якщо проблемний файл не відкритий у редакторі.

Rider для Unreal Engine працює на Windows, MacOS і Linux. IDE заснована на ReSharper C++, отже забезпечує просунуту підтримку сучасного C++. У розпорядженні користувачів є більше 1300 інспекцій коду, 290 швидких виправлень, рефакторинг по всьому рішенню та можливості кодогенерації. Крім того, Rider автоматично додає відсутні директиви #include, і все це без шкоди для швидкості роботи та часу відгуку IDE. Rider спеціально налаштована для роботи з кодом Unreal Engine. Нативні файли .uproject можна відкривати у Rider, не створюючи проект Visual Studio, Makefile чи Xcode. IDE читає файли Blueprints з вашого проекту, показує використання таких файлів у коді C++ і відображає значення перевизначених властивостей.

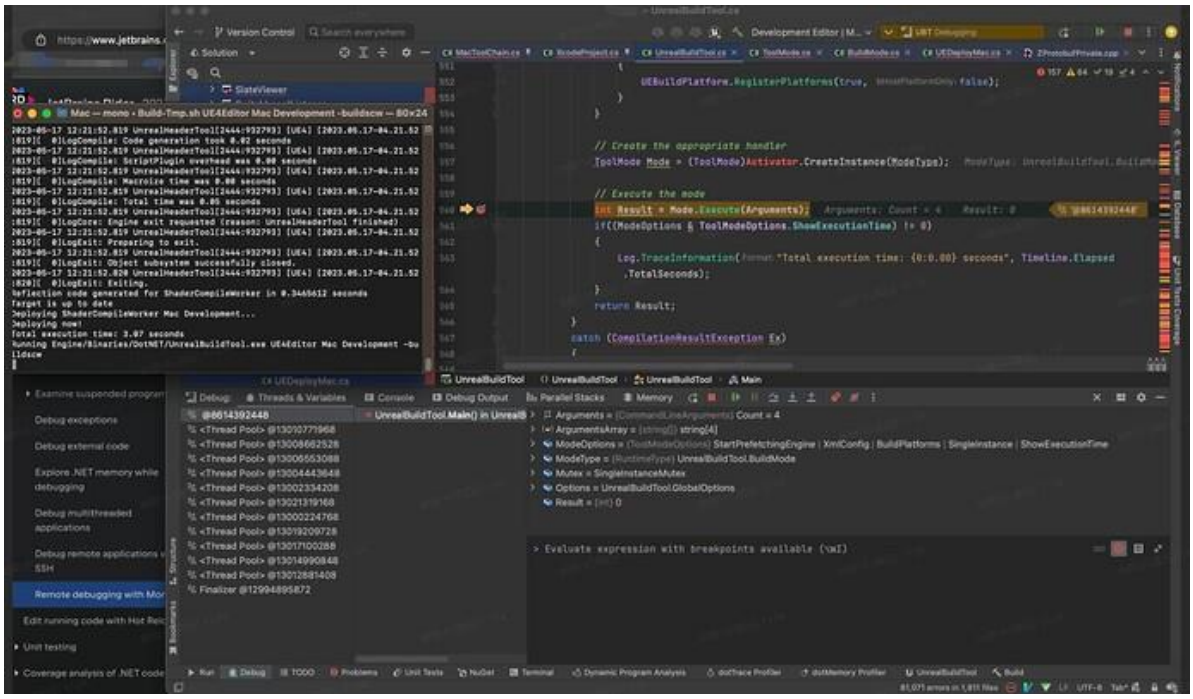


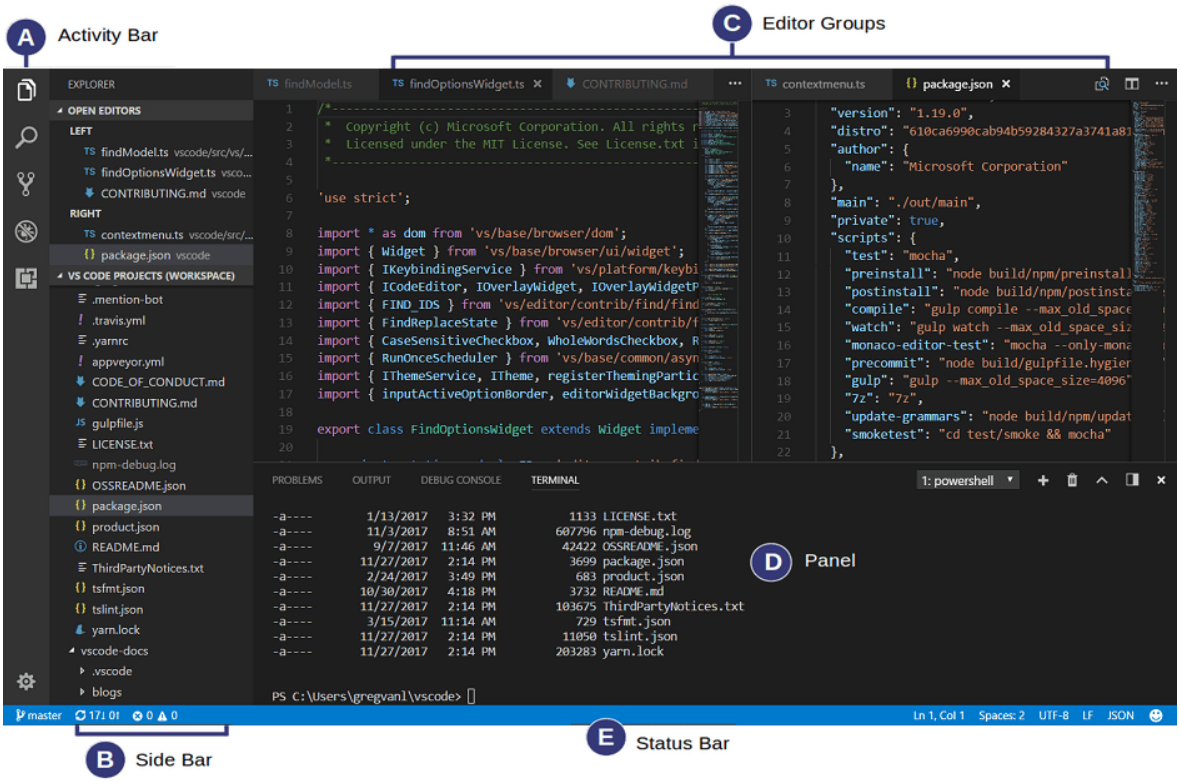
Рисунок 2.8 – Rider IDE

Керувати грою також можна безпосередньо в IDE, використовуючи розширену версію журналу Unreal Editor.

Rider також має плагіни для роботи з Unreal Engine такий як UnrealLink. Плагін UnrealLink забезпечує розширену інтеграцію між JetBrains Rider і Unreal Editor від Epic Games. Плагін передає інформацію Blueprints у редактор, додає налаштування для керування запуском гри та надає більш зручну версію журналу Unreal Editor.

Рисунок 2.9 – Плагін Unreal Link у Rider

Visual Studio Code, який також зазвичай називають VS Code — це редактор вихідного коду, розроблений Microsoft для Windows, Linux і macOS. Функції включають підтримку налагодження, підсвічування синтаксису, інтелектуальне завершення коду, фрагменти, рефакторинг коду та вбудований Git. Користувачі можуть змінювати тему, комбінації клавіш, параметри та встановлювати розширення, які додають функціональність.



Рисунко 2.10 – Редактор вихідного коду Visual Studio Code

Редактор вихідного коду можна використовувати з різними мовами програмування, включаючи C, C#, C++, Fortran, Go, Java, JavaScript, Node.js, Python, Rust і Julia. Він базується на структурі Electron, яка використовується для розробки веб-додатків Node.js, які працюють на механізмі компонування Blink. Visual Studio Code використовує той самий компонент редактора, який використовується в Azure DevOps.

Visual Studio Code містить базову підтримку для більшості поширених мов програмування. Ця базова підтримка включає підсвічування синтаксису, зіставлення дужок, згортання коду та настроювані фрагменти. Visual Studio Code

також постачається з IntelliSense для JavaScript, TypeScript, JSON, CSS і HTML, а також підтримує налагодження Node.js. Підтримка додаткових мов може бути забезпечена безкоштовними розширеннями на VS Code Marketplace.

Код Visual Studio можна розширити за допомогою розширень, доступних через центральне сховище. Це включає доповнення до редактора та підтримку мови. Примітною функцією є можливість створювати розширення, які додають підтримку нових мов, тем, налагоджувачів, налагоджувачів подорожей у часі, виконують статичний аналіз коду та додають лінтери коду за допомогою протоколу Language Server.

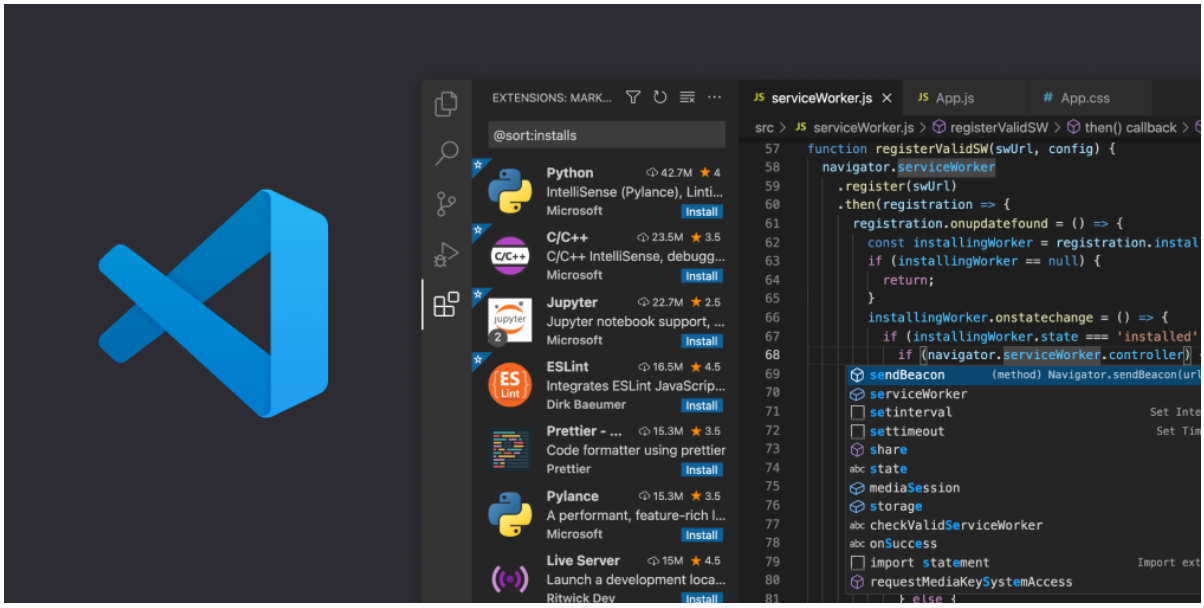


Рисунок 2.11 – Розширення для Visual Studio Code

Керування джерелом є вбудованою функцією Visual Studio Code. Він має спеціальну вкладку всередині панелі меню, де користувачі можуть отримати доступ до налаштувань керування версіями та переглянути зміни, внесені до поточного проекту. Щоб використовувати цю функцію, Visual Studio Code має бути зв’язано з будь-якою підтримуваною системою керування версіями (Git, Apache Subversion, Perforce тощо). Це дозволяє користувачам створювати репозиторії, а також робити запити push і pull безпосередньо з програми Visual Studio Code.

Visual Studio Code можна використовувати для роботи з Unreal Engine, але

в порівнянні з Rider чи Visual Studio, які є інтегрованими середовищами розробки, не може бути конкуретним – фукціонал набато менший і без компілятора Visual Studio, його не можна використовувати.

Порівняємо Visual Studio та Rider і чому Rider буде кращим вибором для нашої роботи.

Rider на відміну від Visual Studio, не зациклився на 32-бітних процесах. Навіть якщо у Rider є процеси, доступні тільки для back-end, наприклад, SWEA (Solution-Wide Analysis), створення коду буде проходити гладко, без жодних пауз або збоїв. І як зазначає більшість користувачів працювати з Visual Studio і Rider, останній працює набагато стабільніше і швидше ніж VS.

JetBrains Rider є кросплатформним, він може працювати на платформах Windows, Mac або Linux з однаковою функціональністю і стабільністю. Visual Studio працює переважно на платформі Windows. І якщо є необхідність перейти на Linux або Mac, то необхідно буде купувати додаткові рішення: Visual Studio Code (для Linux) і Visual Studio для Mac. Головним недоліком є те, що версії Visual Studio для Mac і Linux мають різний функціонал і зовнішній вигляд до якого доведеться звикати. Rider, як зовні так і за своїми функціями, однаковий на всіх платформах, тому, якщо користувач вирішить перейти з Windows на Mac або Linux, він отримає вже звичне середовище розробки і не буде витрачати дорогоцінний час на навчання.

Середовище Rider включає більшість функцій, популярного розширення Visual Studio для розробників .NET – ReSharper. У складі Rider є значний набір для рефакторінгу, перевірки коду і контекстних дій для всіх підтримуваних мов і технологій. У Visual Studio також є набори рефакторингов і перевірки помилок коду, але набагато більш обмежений, ніж ті, що надані у Rider і ReSharper.

Варто також згадати про те з якими інструментами від JetBrains раніше працювали користувачі. Ті з них, хто не з чуток знайомий з IntelliJ IDEA, WebStorm, DataGrip або іншими середовищами набагато швидше перейдуть на Rider ніж користувачі, які працювали тільки з Visual Studio.

У Rider є величезна кількість функцій, успадкованих від платформи IntelliJ:

- Підтримка систем контролю версій: крім Git і Mercurial, Rider працює з CVS та Subversion. Інтеграція VSTS доступна через спеціальний плагін, підтримуваний компанією Microsoft.
- Rider (з допомогою DataGrip) підтримує підключення до баз даних і SQL. Користувачам Visual Studio в більшості випадків потрібно буде використовувати ODBC.
- Підтримка можливостей для front-end розробки з використанням JavaScript, TypeScript, CSS, HTML, LESS, Sass і т. д. Доступна в Rider завдяки тому, що продукт містить у своєму складі, функції спеціалізованого ПЗ для веб-розробки – JetBrains WebStorm.
- У середовищі розробки Rider також присутня можливість інтеграції з багатьма трекерами проблем, такими як Team Foundation Server і Visual Studio Team Services. Також він підтримує JIRA Software, YouTrack та інші рішення, а також велику кількість високоякісних спеціалізованих плагінів, розроблених для IntelliJ і ReSharper, більшість з яких безкоштовні. Visual Studio теж підтримує різні плагіни, але безкоштовними з них є одиниці.

Рішення і проекти, з якими працює JetBrains Rider, повністю сумісні з Visual Studio, і не використовують власні формати.

При виборі IDE надаємо перевагу Rider, тому що вона надає більш специфічний функціал для работи з Unreal Engine. Але Rider не працює без компілятора та бібліотек, які є частиною Visual Studio, тому бажано одразу мати два середовища розробки для комфортної роботи з проектами на UE.

2.3 Опис функціоналу застусунку генерації 3D ландшафту

В основі застусунку є алгоритм Perlin Noise, який генерує градієнтний псевдовипадкой шум на основі якого створюється ландшафт. Розберемо більш детальніше.

Алгоритм приймає як вхід певну кількість параметрів з плаваючою комою

і повертає значення в певному діапазоні. Скажімо, розрядність є двовимірним, тому він приймає 2 параметри: x і y. Тепер x і y можуть бути чим завгодно, але зазвичай вони є позицією. Щоб створити текстуру, x і y будуть координатами пікселів у текстурі. Отже, для генерації текстури ми б прокрутили кожен піксель у текстурі, викликавши функцію шуму Перліна для кожного з них і вирішивши, на основі поверненого значення, якого кольору буде цей піксель.

Вхідні дані розглядаються як цілочисельна сітка (Рисунок 2.1). Кожен вхід із плаваючою комою лежить у межах квадрата цієї сітки. Для кожного з 4 кутів цього квадрата ми генеруємо значення. Потім ми інтерполюємо між цими 4 значеннями й отримуємо остаточний результат. Різниця між шумом Перліна та шумом значення полягає в тому, як отримані ці 4 значення. Там, де шум значення використовує генератор псевдовипадкових чисел, шум Перліна виконує скалярний добуток між 2 векторами.

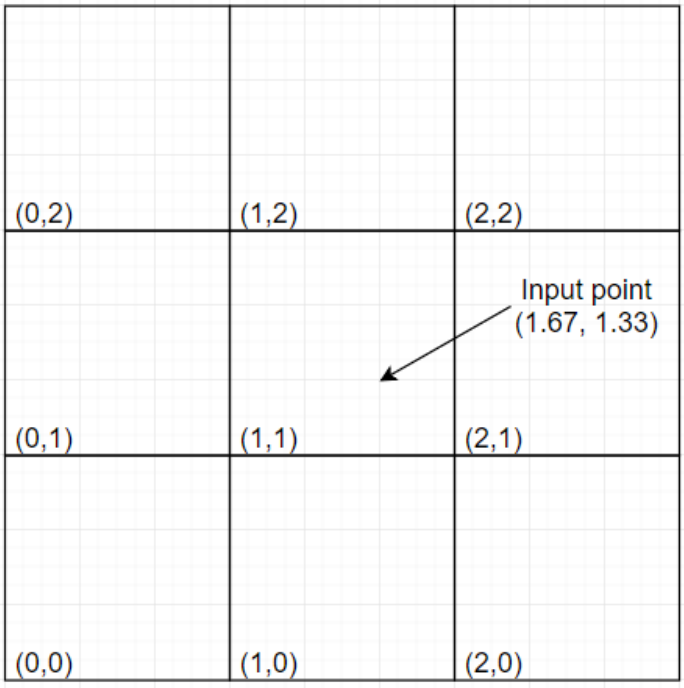


Рисунок 2.12 – Сітка вхідних данних

Перший вектор — це той, що вказує від точки сітки (кутів) до точки введення. Інший вектор є постійним вектором, призначеним кожній точці сітки

(Рисунок. 2.13). Він завжди має бути однаковим для тієї самої точки сітки, але він може змінитися, якщо ви зміните початковий seed алгоритму.

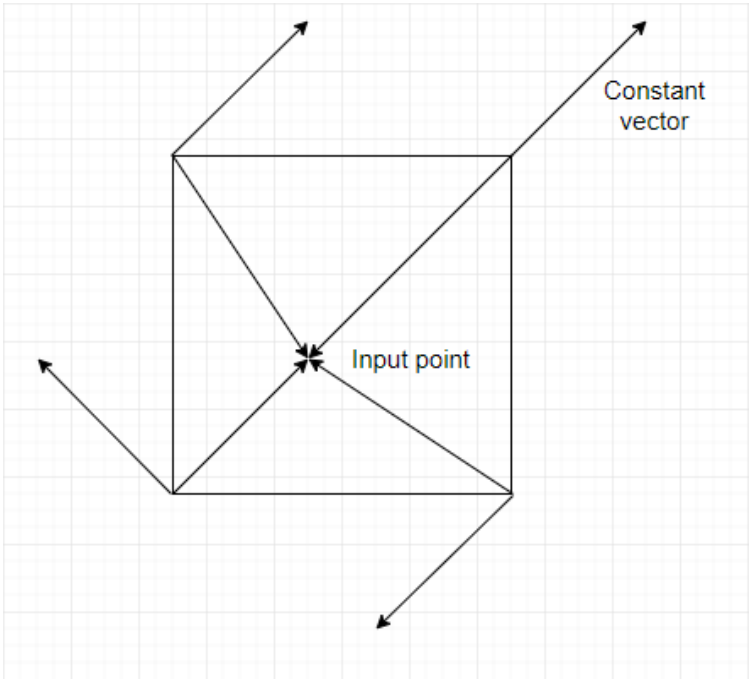


Рисунок 2.13 – Вектори для генерації значення

Загалом, у реалізаціях шуму Perlin шум «загортається» після кожного кратного 256 (назвемо це число w), тобто він повторюватиметься. Це тому, що, щоб надати кожній точці сітки постійний вектор, незабаром нам знадобиться щось під назвою таблиця перестановок. Це масив розміром w, який містить усі цілі числа від 0 до w-1, але перетасовані. Індекс для цього масиву дорівнює X або Y (або значення біля них), тому він має бути меншим за 256. Шум «загортається», оскільки якщо, наприклад, вхід x 256, X дорівнюватиме 0. Цей 0 буде використано для індексування таблиці перестановок, а потім для генерації випадкового вектора. Оскільки X дорівнює 0 у кожному кратному 256, випадковий вектор буде однаковим у всіх цих точках, тому шум повторюється. Можна мати більшу таблицю перестановок, скажімо, розміром 512, і в цьому випадку шум буде обертатися на кожному кратному 512.

Річ у тім, що це просто техніка, яку використав Кен Перлін, щоб отримати постійні вектори для кожної кутової точки. Можна використати інший спосіб, і, можливо, не буде обмежень обгортання, наприклад, використати генератор псевдовипадкових чисел для генерації постійних векторів, але в цьому випадку, ймовірно, будете краще, просто використовуючи шум значення.

Далі знадобиться значення з цієї таблиці для кожного з кутів. Однак існує обмеження: кут повинен завжди отримувати те саме значення, незалежно від того, яка з 4 клітинок сітки, що має його як кут, містить вхідне значення. Наприклад, якщо правий верхній кут комірки сітки (0, 0) має значення 42, тоді лівий верхній кут комірки сітки (1, 0) також має мати таке ж значення 42.

Оригінальна реалізація Кена Перліна використовувала функцію під назвою «gradient», яка обчислювала скалярний добуток безпосередньо для кожного кута. Її можна замінити на більш прості реалізації, але кращого результату ніж без орігінальної функції неможливо досягти.

Тепер, коли маємо скалярний добуток для кожного кута, нам потрібно якось змішати їх, щоб отримати єдине значення. Для цього скористаємося інтерполяцією. Інтерполяція — це спосіб знайти значення, яке лежить між 2 іншими значеннями (скажімо, a1 і a2), враховуючи деяке інше значення t між 0,0 і 1,0 (загалом відсоток, де 0,0 — 0%, а 1,0 — 100%). Наприклад: якщо a1 дорівнює 10, a2 дорівнює 20, а t дорівнює 0,5 (тобто 50%), інтерпольоване значення буде 15, оскільки воно знаходиться посередині між 10 і 20 (50% або 0,5). Інший приклад: a1=50, a2=100 і t=0,4. Тоді інтерпольоване значення буде на 40% шляху між 50 і 100, тобто 70. Це називається лінійною інтерполяцією, оскільки інтерпольовані значення є лінійною кривою.

Маємо 4 значення, які ипотрібно інтерполювати, але можемо інтерполювати лише 2 значення одночасно. Таким чином, спосіб використання інтерполяції для шуму Перліна полягає в тому, що інтерполюємо значення верхнього лівого та нижнього лівого кутів разом, щоб отримати значення, яке ми назвемо v1. Після цього ми робимо те саме для верхнього правого та нижнього правого кутів, щоб отримати v2. Потім, нарешті, ми інтерполюємо між v1 і v2, щоб отримати остаточне значення. Це значення, яке ми хочемо, щоб повернула наша функція шуму.

Якщо ми лише трохи змінимо вхідну точку, вектори між кожним кутом і вхідною точкою також трохи зміняться, тоді як постійний вектор не зміниться зовсім. Скалярний добуток також трохи зміниться, як і кінцеве значення, яке

повертає функція шуму. Навіть якщо вхідні дані змінюють квадрат сітки, наприклад, з (3.01, 2.01) на (2.99, 1.99), кінцеві значення все одно будуть дуже близькими, оскільки навіть якщо 2 (або 3) кути змінюються, інші 2 (або 1) не буде, і оскільки з обома вхідними даними ми близькі до кутів, інтерполяція призведе до того, що кінцеве значення буде дуже близьким до значення кутів. Оскільки з обома вхідними даними цей кут матиме однакове значення, кінцеві результати будуть дуже близькими.

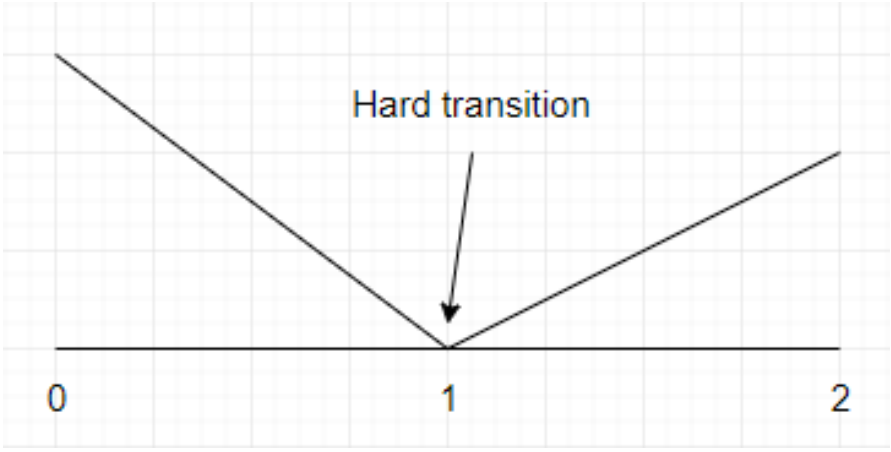
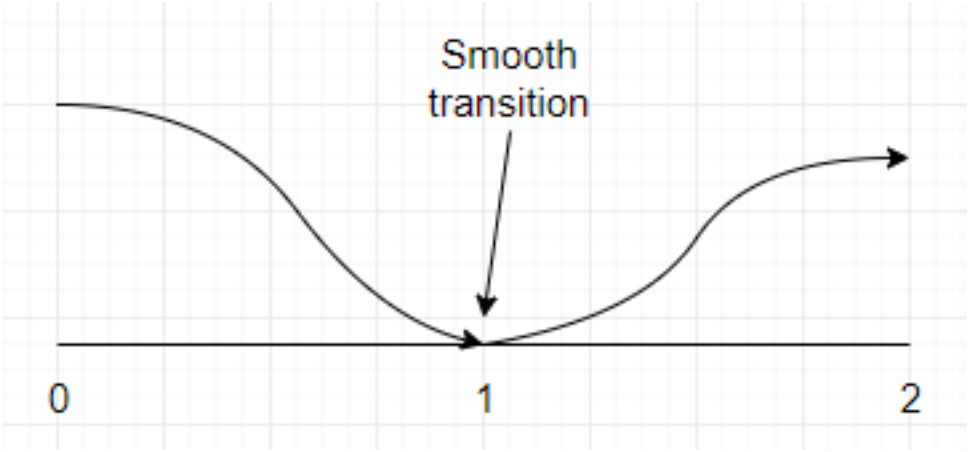


Рисунок 2.14 – Різкий перехід у результаті лінійної інтерполяції



Риснок 2.15– Плавний перехід у результаті нелінійної інтерполяції

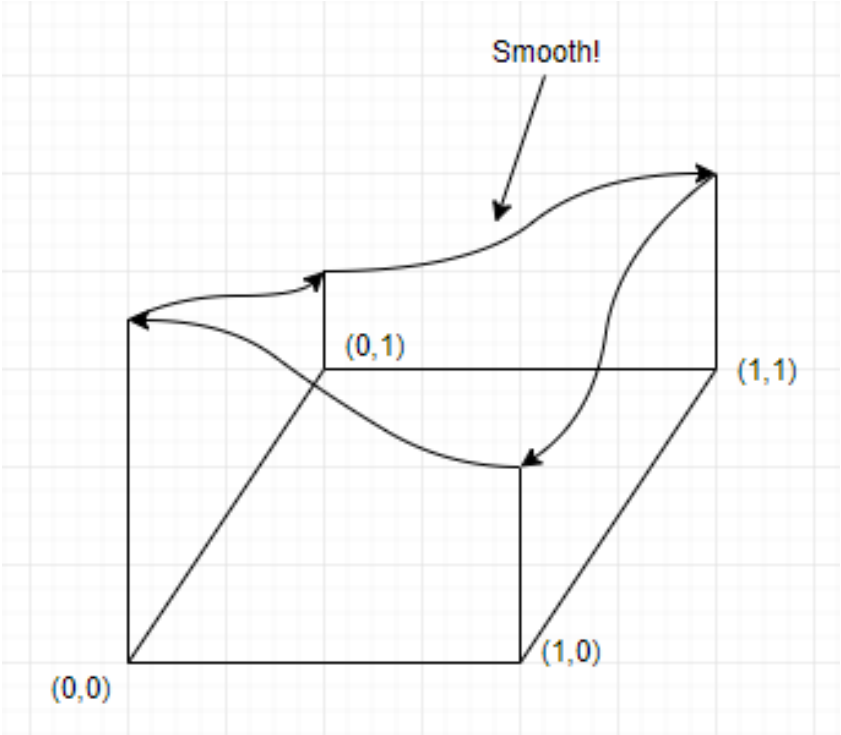


Рисунок 2.16 – Плавний перехід між кутами квадрата сітки

З лінійною інтерполяцією ми б використовували xf як значення інтерполяції (t). Замість цього ми збираємося перетворити xf і yf на u і v. Ми зробимо це таким чином, що, враховуючи значення t між 0,0 і 0,5 (за винятком), перетворене значення буде дещо меншим (але обмеженим 0,0). Крім того, враховуючи значення t між 0,5 (виключено) і 1,0, перетворене значення буде трохи більшим (але обмежене 1,0). Для 0,5 перетворене значення має бути 0,5. Це призведе до викривленого переходу як на рисунках 2.15 та 2.16.

Для цього нам знадобиться щось, що називається кривою легкості: це просто математична крива (Рисунок 2.17)

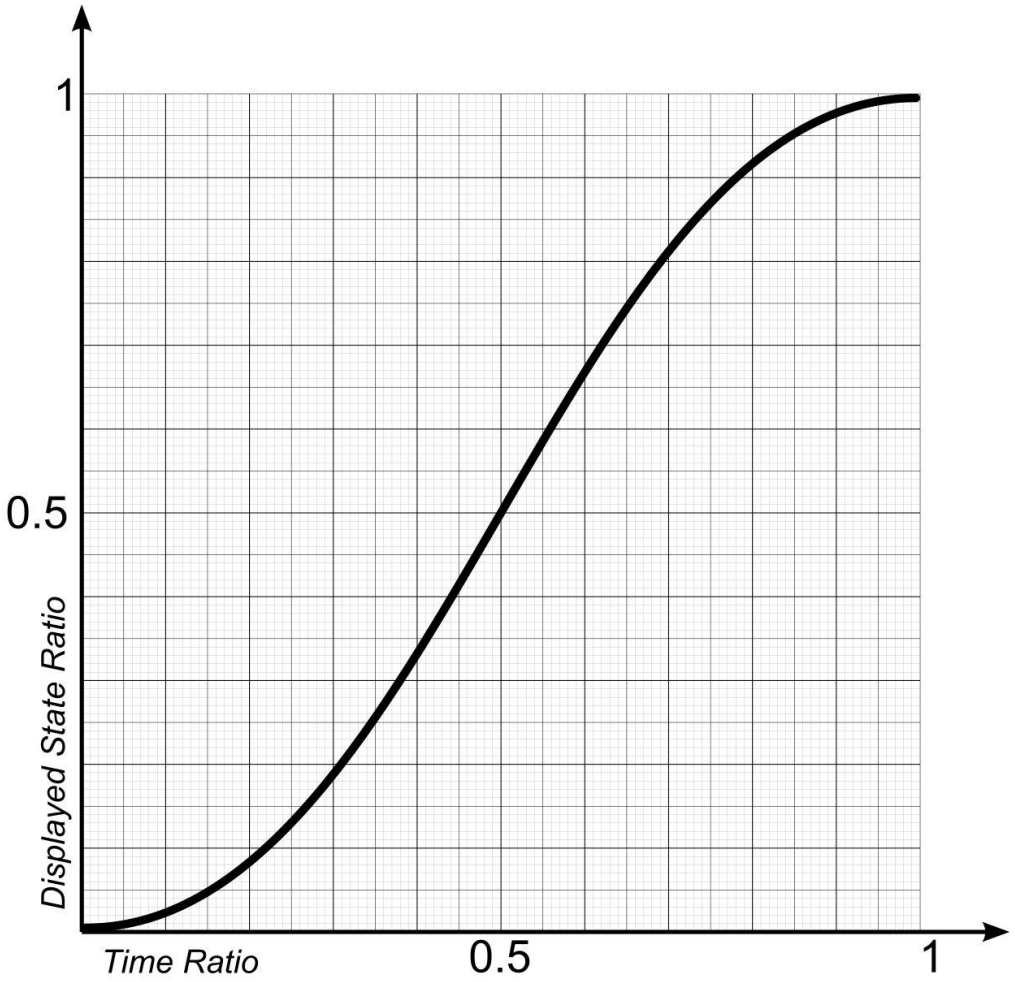


Рисунок 2.17 - Крива легкості

Коли всі вхідні дані для алгоритму є цілими числами, скажімо (5,3), вектор від точки сітки (5,3) до вхідних даних буде вектором (0,0), оскільки вхідні дані також є (5,3). Скалярний добуток для цієї точки сітки дорівнюватиме 0, і оскільки вхідні дані знаходяться саме в цій точці сітки, інтерполяція призведе до того, що результат буде саме таким скалярним добутком, тобто 0. Щоб вирішити цю невелику проблему, ми зазвичай множимо на введення невеликим значенням, яке називається частотою.

Для приведення до найбільш природнього вигляду генерації ландшафту, бажано використовувати фрактальний броунівський рух. ФБР не є частиною основного шумового алгоритму Перліна, але майже завжди використовується з ним. Він має декілька параметрів: частота та октави, які впливають на гладкість

генерованої поверхні.

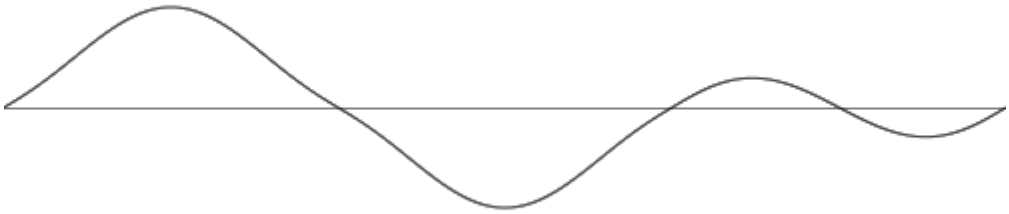


Рисунок 2.18 – Одновимірний шум Перліна з частотою 1

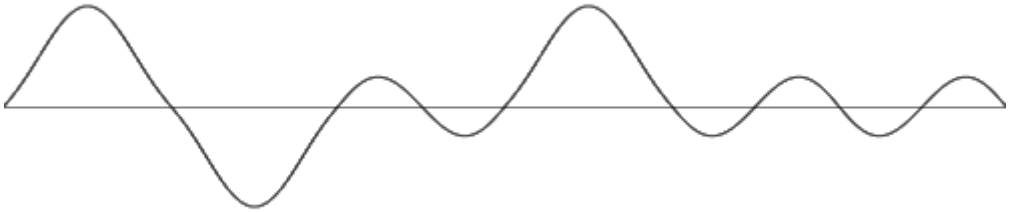


Рисунок 2.19 – Одновимірний шум Перліна з частотою 2

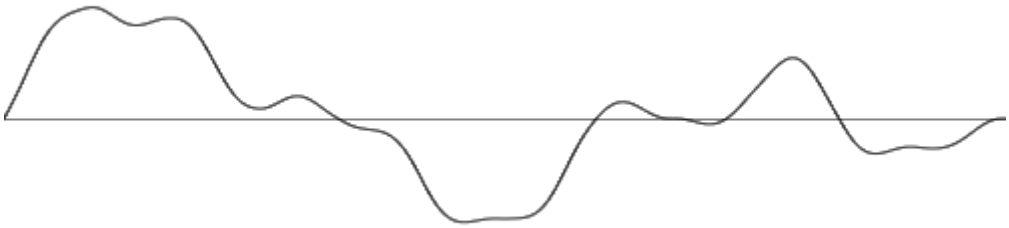


Рисунок 2.20 Одновимірний шум Перліна з 2 октавами

Перша октава становить загальну форму нашого ланцюга гір. Він має невелику частоту і амплітуду. Друга октава додасть більш шумні деталі до гірського масиву. Ми можемо продовжувати це робити - додавати все дрібніші деталі до гір - доки не отримаємо остаточний результат. (Рисунок 2.10)

Рисунок 2.21 Одновимірний шум Перліна з 8 октавами

3 РЕАЛІЗАЦІЯ ЗАСТУСУНОКУ ГЕНЕРАЦІЇ ЛАНДШАФТА НА РУШІЇ UNREAL ENGINE 5

3.1 Налаштування робочого простору

Перш ніж почати розробку застусунку треба підготувати робочій простір. Для засутсунків на платформі UE5 потрібно завантажити та налаштувати такі програми: Unreal Engine 5, JetBrains Rider, Visual Studio з бібліотека для розробки ігр на C++.

Почнемо з найголовнішего – ігрового рушія. UE5 можна скачати на офіційному за посиланням https://www.unrealengine.com/en-US . За допомогою застусунку Epic Games Launcher у вкладці «Unreal Engine» треба обрати останню верісю рушія та натиснути скачати або оновити, якщо у вас вже був рушій попередніх версій. (Рисунок 3.1)

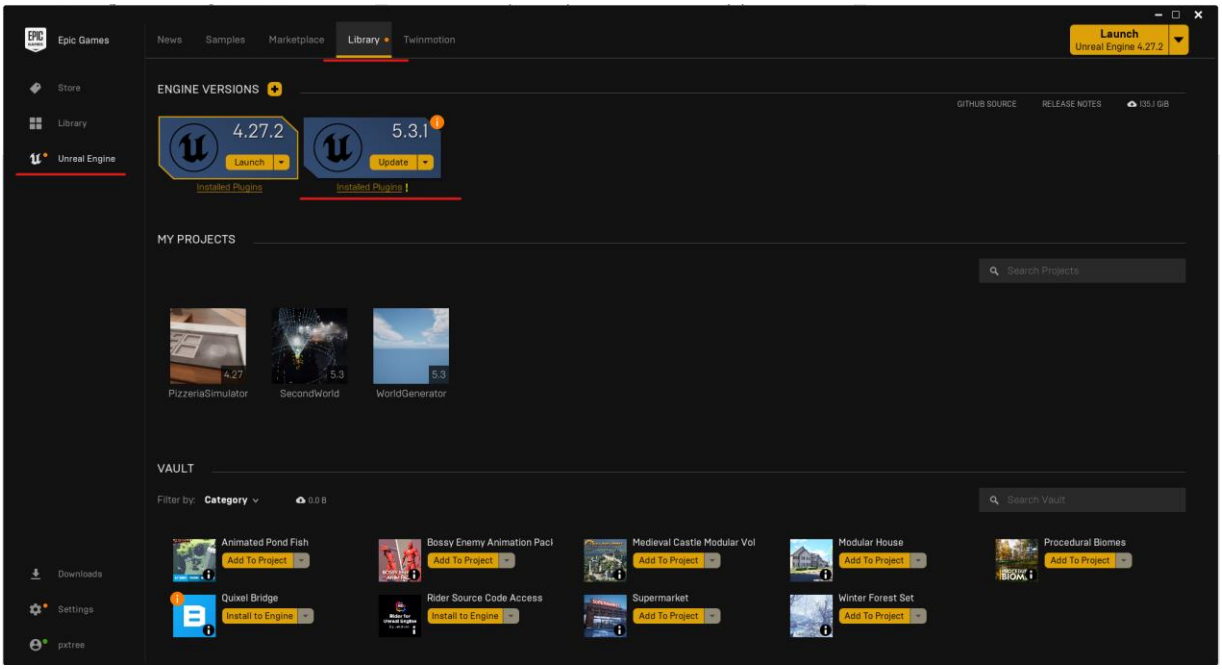


Рисунок 3.1 – Завантаження рушія з Epic Games Launcher

До того як створити проект, потрібно заватажити та інсталювати Visual Studio з пакетами бібліотек для розробки ігр на C++. Завантажити інтегроване середовище розробки можна з офіційного сайту https://visualstudio.microsoft.com/ . В застусунку Visual Studio Installer треба

обрати останню версію застусунку та у вкладці «modify» обов'язково обрати пункт «Game development with C++». Це включить у інсталяцію застусунку всі необхідні бібліотеки для роботи з Unreal Engine.

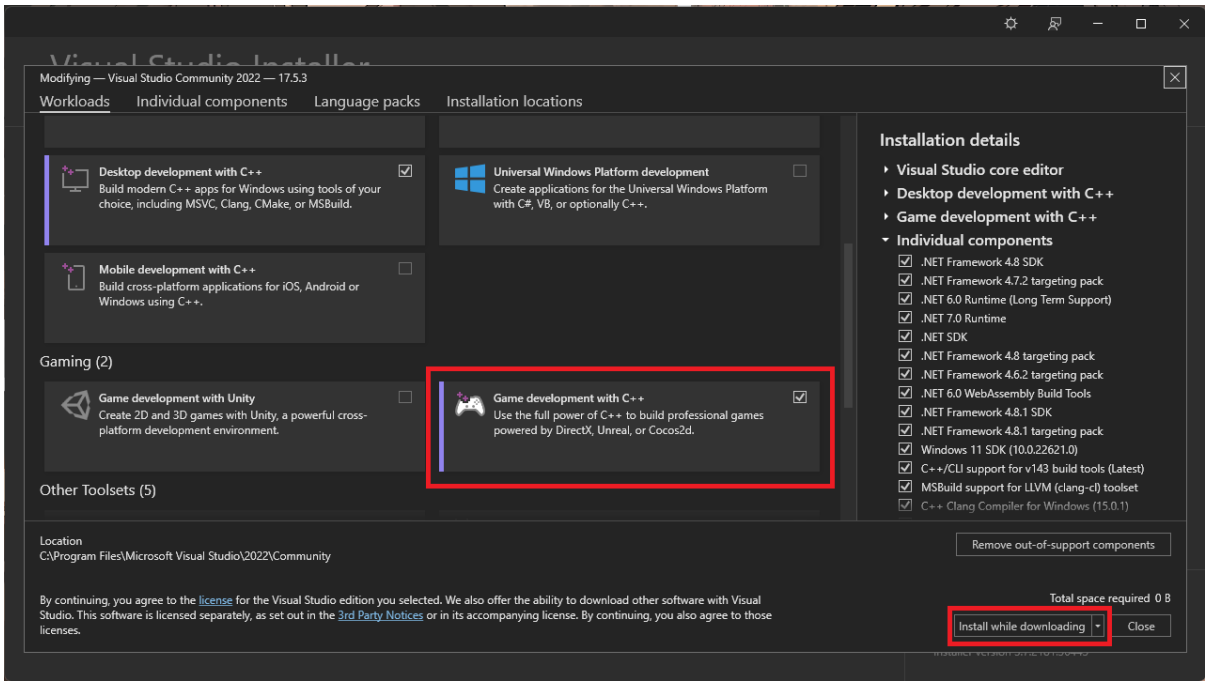


Рисунок 3.2 – Інсталяція Visual Studio з пакетами для розробки ігр на UE5

Після завантаження необхідних бібіотек та компілятора, які є компонентами Visual Studio, можна завантажити JetBrains Rider. IDE завантажується з сайту розробників https://www.jetbrains.com/rider/ . У налаштуванні Rider немає ніяких важливих для нас опцій, тому при інсталюванні налаштовуємо під особисті потреби.

Для комфортної робити з Rider та Unreal Engine потрібно встановити плагіни Rider Link для ігрового редактора та Unreal Link для інтегрованого середовища розробки.

Плагін UnrealLink забезпечує розширену інтеграцію між JetBrains Rider і Unreal Editor від Epic Games. Плагін передає інформацію Blueprints у редактор, додає налаштування для керування запуском гри та надає більш зручну версію журналу Unreal Editor.

UnrealLink входить у комплект JetBrains Rider. Починаючи з Rider для Unreal Engine 2020.2.1, він також поширюється через JetBrains Marketplace.

RiderLink встановлюється JetBrains Rider самостійно, немає необхідності встановлювати його вручну. Коли ви вперше відкриваєте проект Unreal Engine у JetBrains Rider, ви побачите сповіщення про те, що плагін RiderLink відсутній, і запрошення встановити його. Якщо ви пропустите це спливаюче повідомлення, ви зможете встановити плагін пізніше на сторінці Languages & Frameworks | Сторінка Unreal Engine налаштувань JetBrains Rider.

І спливаюче повідомлення, і сторінка налаштувань пропонують два варіанти встановлення:

- Рушій: виберіть цей параметр, щоб установити плагін у рушію та використовувати його для всіх ігрових проектів на основі поточної версії рушія. Плагін з’явиться в папці Engine/Plugins/Developer.
- Гра: виберіть цей параметр, щоб установити плагін у проекті гри та використовувати його лише для поточного проекту. Плагін
- з’явиться в папці Game/Plugins/Developer.

Якщо пізніше ви вирішите змінити місце встановлення, скористайтеся пунктом «Примусова інсталяція RiderLink у рушій»/«Примусова інсталяція RiderLink у грі» за допомогою «Знайти дію».

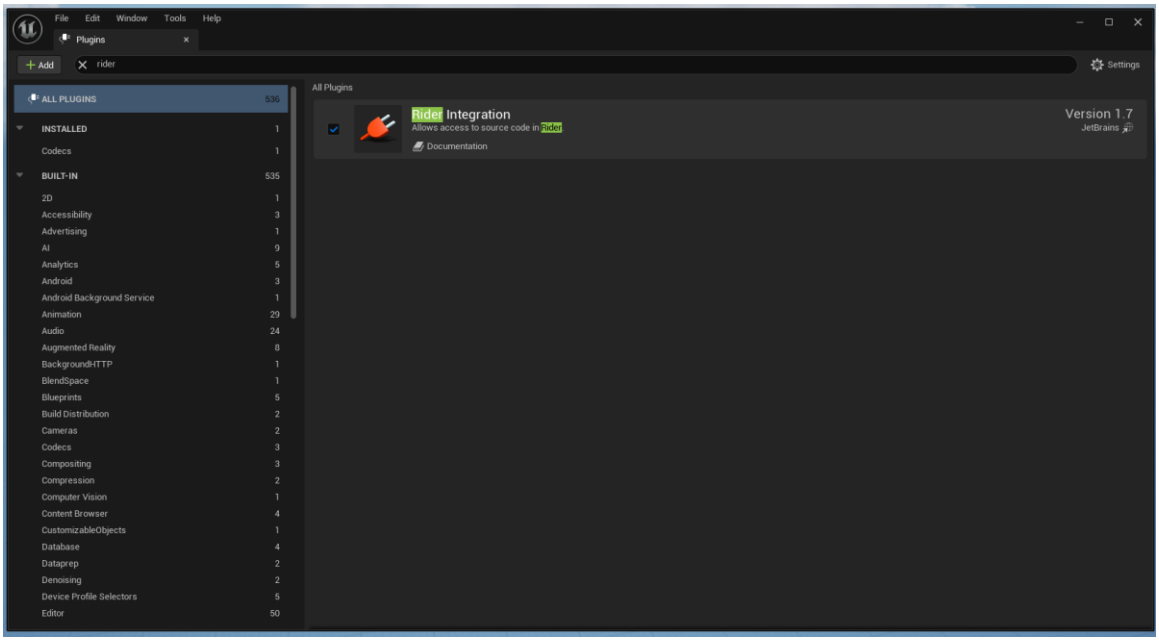


Рисунок 3.3 – Підключення плагіну Rider Link

Налаштувавши усі програми для написання коду можемо створити Unreal Engine проект. Для цього у застусунку Epic Games Launcher обираємо завантажену версію рушія і натискаємо «Launch». Після невеликої загрузки вікриється вікно з нещодавніми проектами, та пресетами для створення нових проектів. Для створення проекту потрібно перейти у розділ «Games». Обираємо шаблон «Blank» в якому немає ніякого заготовоеного коду. Також, обираемо тип проекта як «C++» та називаемо наш проект. Після обраних опцій натискаємо «Create».

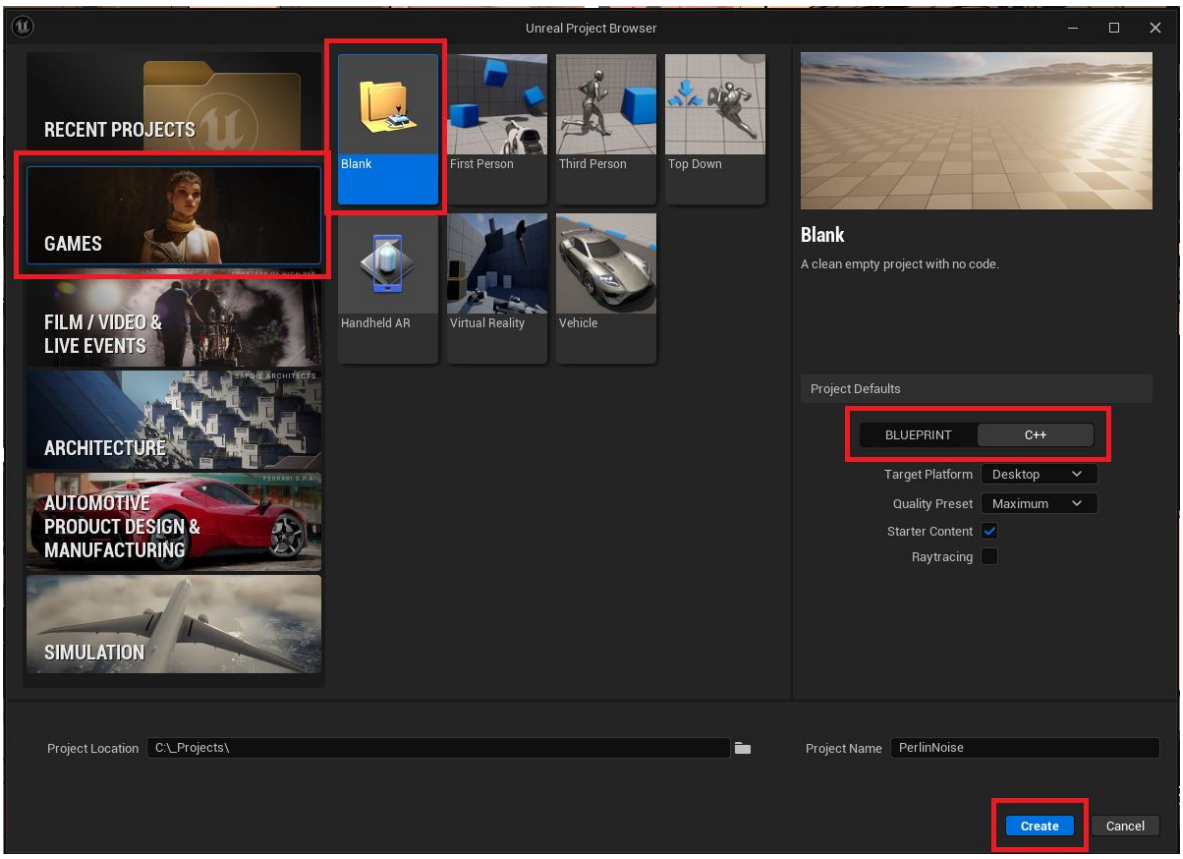


Рисунок 3.4 – Створення проекту на UE5

Як проект згенерувався, треба встановити для рушія застусунок за замочуванням для редагування коду. У вкладці «Edit» треба відкрити меню «Editor Preferences». У вікні налаштування редактора, у вкладці «Source code», подтрібно виставити «Rider». (Рисунок 3.5)

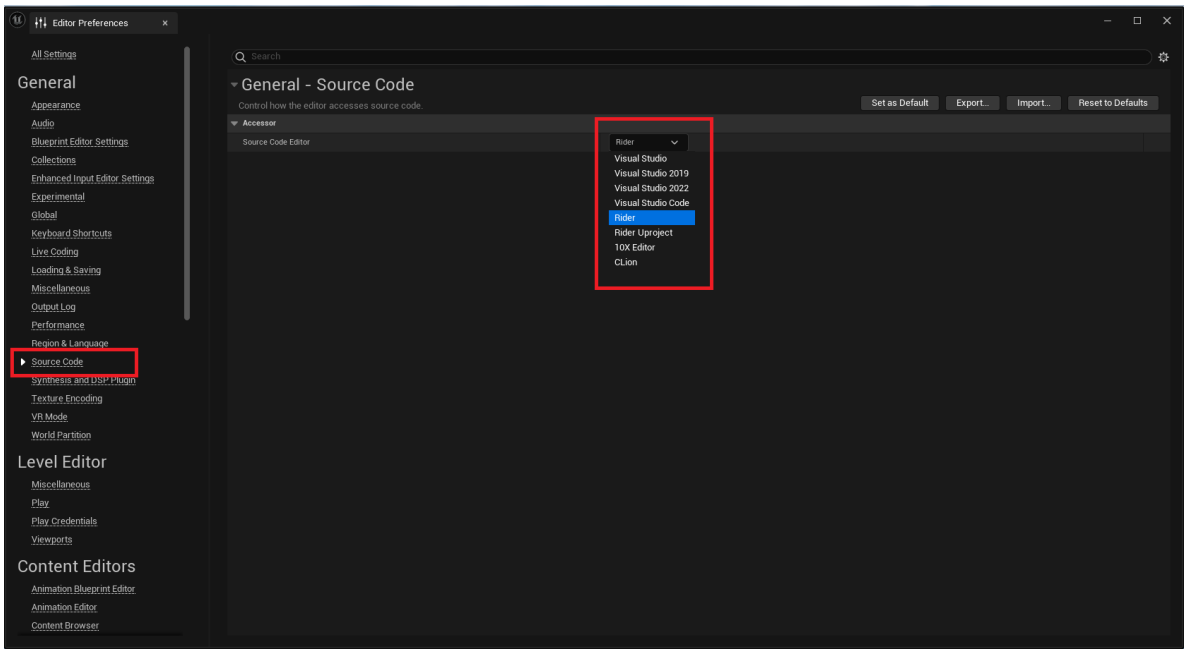


Рисунок 3.5 – Встановлення застусунку для редагування коду за замочуванням

3.2 Розробка генерації ландшафту

Створивши та налаштувавши проект, можна переходити до реалізації генерування ландшафту за допомогою аглоритму Perlin Noise.

Першою задачею є створення ігрової сцени, де буде генеруватися ландшафт. Створемо окремий ігровий рівень. Щоб на рівні було щось видно необхідно додати до нього світло. Також можна додати текстуру яка буде відображати небо з хмарами. Всі компоненти можна скопіювати з тестового рівня, який буде включений у проект, якщо при створенні проекту ви обрати пункт «Started Content». (Рисунок 3.6)

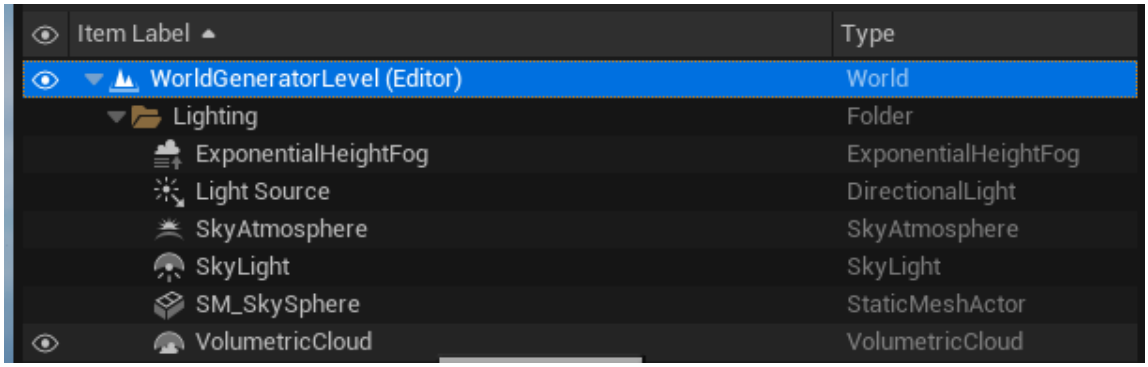


Рисунок 3.6 – Список компонентів для сворення світла на сцені

На рисунку 3.7 зображено результат після додавання необхідних компонентів.

Рисунок 3.7 – Початкова сцена рівня

Сцена готова, перейдемо до реалізації алгоритму Perlin Noise. У вкладці «Tools» виберемо пункт «Open Rider» для того щоб перейти у C++ проект для написання ігрової логіки. У вікні IDE у розділі «Solution Explorer» створюємо новий класс який буде включати реалізацію алгоритму Perlin Noise. Класс включає 4 функції: randomGradient, dotGridGradient, interpolate, perlin. (Рисунок 3.8)

```
#pragma once

class PerlinNoise
{
public:
    static FVector2D randomGradient(int ix, int iy);
    static float dotGridGradient(int ix, int iy, float x, float y);
    static float interpolate(float a0, float a1, float w);
    static float perlin(float x, float y);

};
```

Рисунок 3.8 – Функції реалізації алгоритму шуму Перліна

Функція randomGradient повертає двовимірний вектор з випадкомими значеннями. Відсутність попередньо обчислених градієнтів означає, що це працює для будь-якої кількості координат сітки.

```
FVector2D PerlinNoise::randomGradient(int ix, int iy) {
    // No precomputed gradients mean this works for any number of grid coordinates
    const unsigned w = 8 * sizeof(unsigned);
    const unsigned s = w / 2;
    unsigned a = ix, b = iy;
    a *= 3284157443;

    b ^= a << s | a >> (w - s);
    b *= 1911520717;

    a ^= b << s | b >> (w - s);
    a *= 2048419325;
    float random = a * (3.14159265 / ~(~0u >> 1)); // in [0, 2*Pi]

    // Create the vector from the angle
    FVector2D v;
    v.X = sin(random);
    v.Y = cos(random);

    return v;
}
```

Рисунок 3.9 – Функція randomGradient

Функція dotGridGradient обчислює скалярний добуток векторів відстані та градієнта.

```
// Computes the dot product of the distance and gradient vectors.
float PerlinNoise::dotGridGradient(int ix, int iy, float x, float y) {
    // Get gradient from integer coordinates
    FVector2D gradient = randomGradient(ix, iy);

    // Compute the distance vector
    float dx = x - (float)ix;
    float dy = y - (float)iy;

    // Compute the dot-product
    return (dx * gradient.X + dy * gradient.Y);
}
```

Рисунок 3.10 – Функція dotGridGradient

Функція interpolate обчислює інтерполяцію між двома координатами.

```
float PerlinNoise::interpolate(float a0, float a1, float w)
{
    return (a1 - a0) * (3.0 - w * 2.0) * w * w + a0;
}
```

Рисунок 3.11 - Функція interpolate

Функція perlin обчислює значення шуму Перліна на заданних координатах використовуючи функції наведені вище. (Рисунок 3.9). Спочатку визначається координати кутів клітинки. Потім обчислюється ваги інтерполяції та обчислюється та інтерполюються спочатку два кути верхні кути клітинки, а потім нижні. Останій крок функції – інтерполяція між двома попередніми значеннями, але по координаті Y.

```
float PerlinNoise::perlin(float x, float y) {

    // Determine grid cell corner coordinates
    int x0 = (int)x;
    int y0 = (int)y;
    int x1 = x0 + 1;
    int y1 = y0 + 1;

    // Compute Interpolation weights
    float sx = x - (float)x0;
    float sy = y - (float)y0;

    // Compute and interpolate top two corners
    float n0 = dotGridGradient( ix: x0,  iy: y0, x, y);
    float n1 = dotGridGradient( ix: x1,  iy: y0, x, y);
    float ix0 = interpolate( a0: n0,  a1: n1,  w: sx);

    // Compute and interpolate bottom two corners
    n0 = dotGridGradient( ix: x0,  iy: y1, x, y);
    n1 = dotGridGradient( ix: x1,  iy: y1, x, y);
    float ix1 = interpolate( a0: n0,  a1: n1,  w: sx);

    // Final step: interpolate between the two previously interpolated values, now in y
    float value = interpolate( a0: ix0,  a1: ix1,  w: sy);

    return value;
}
```

Рисунок 3.12 – Реалізація шуму Перліна для заданих координат

Реалізувавши шум Перліна, треба використати його для генерації ландшафту. Unreal Engine надає компонент ProceduralMeshComponent за допомогою якого можна створювати 3D модель об’єкту передаючи як параметри массив трикутників, вершин та метеріалу для відображення. Вершини як раз будуть виступати як координтатною сіткою, а висота вершини – значення шуму Перліна у його координатах. У Unreal Engine координатна одиниця дорівнює одному сантиметру, тому потрібно маштабувати координатну сітку для більшого ландшафту.

Модель всіх 3D об’єктів складається з трикутників. Кожна клітина має 4 кути і виглядає як квадрат. Тому для створення квадратної клітини, потрібно генерувати 2 трикутника, враховуючи маштаб клітин. Потрібно створювати трикутники по вершинам. Індекси вершин мають йти проти годинникової стрілки. На рисунку 3.13 та 3.14 показано як розраховувати індекси для верхин трикутників.

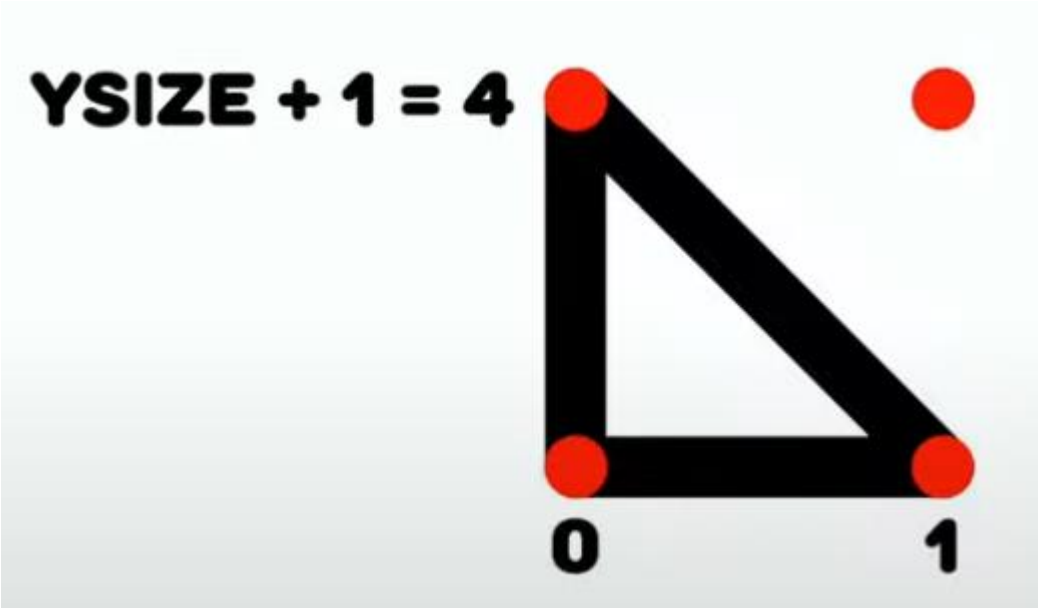


Рисунок 3.13 – Індекеси вершин першого трикутника клітини

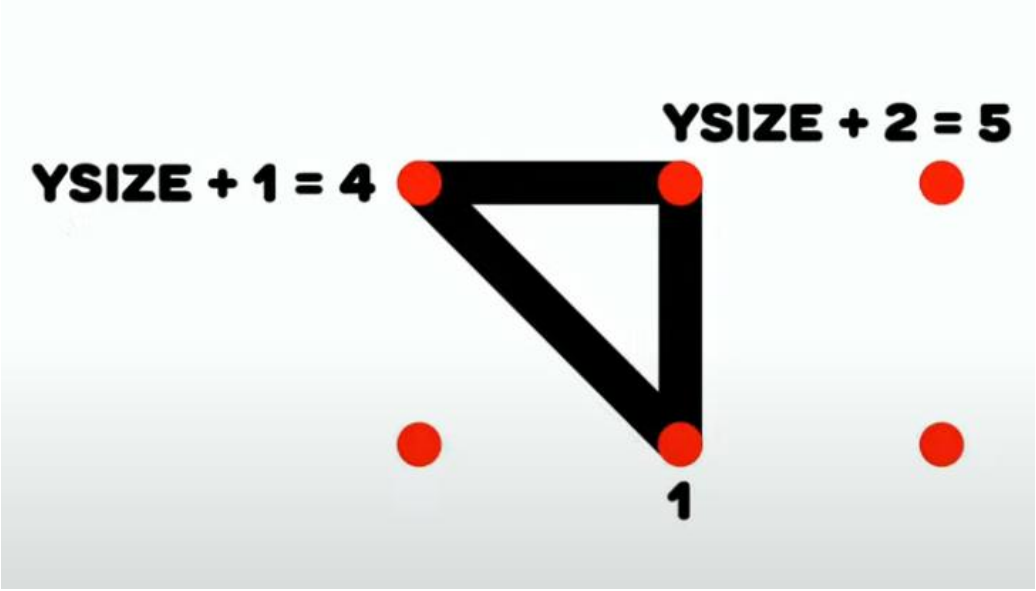


Рисунок 3.14 – Індекеси вершин другого трикутника клітини

У C++ проекті створюєтмо новий клас в якому буде реалізовано створення 3D об’єкту ландшафту за допомогою ProceduralMeshComponent. Клас ATerrain наслідується від клас AActor, який є частиною архітектури UE5. Об’єкт класу AActor має можливість бути поставленим на ігровий рівень. В класі об’явемо такі змінні, розмір поля, кількість октав, насіння та маштаб клітин. Найголовніша змінна – це меш ландшафту, який буде відображати модель.

Список функцій класу:

- createVertices – створення сітки вершин;
- createTriangles – створення трикуників для кожної клітині сітки;
- createPerlinMap – створення массиву значень шуму Перліна;
- generate – функція запуску генерації ландшафту.

```
8       UCLASS(Blueprintable)
        1 derived blueprint class
9       class ATerrain : public AActor
10      {
11          GENERATED_BODY()
12      public:
13          ATerrain();
14
15          UPROPERTY(EditAnywhere, Category="Generation")
16          int width;  "128" (BP_Terrain_C_2)
17
18          UPROPERTY(EditAnywhere, Category="Generation")
19          int height;  "128" (BP_Terrain_C_2)
20
21          UPROPERTY(EditAnywhere, Category="Generation")
22          int octaves;  "12" (BP_Terrain_C_2)
23
24          UPROPERTY(EditAnywhere, Category="Generation")
25          float fBias;  "0.05" (BP_Terrain_C_2)
26
27          UPROPERTY(EditAnywhere, Category="Generation")
28          float seed;  Unchanged in assets
29
30          UPROPERTY(EditAnywhere, Category="Generation")
31          float scale;  "100" (BP_Terrain_C_2)
32
33          UPROPERTY(EditAnywhere, Category="Generation")
34          float UVScale;  "1" (BP_Terrain_C_2)
35
36          UPROPERTY(EditAnywhere, Category="Generation")
37          UMaterial* terrainMaterial;  "M_Ground_Grass" (BP_Terrain_C_2)
38
39          void generate();
40
41      protected:
42
43          UPROPERTY(EditDefaultsOnly)
44          UProceduralMeshComponent* _terrainMesh;  Changed in 2 blueprints
45
46          virtual void BeginPlay() override;
47
48      private:
49          void createVertices(TArray<FVector>& verticles, TArray<FVector2D>& uv0);
50          void createTriangles(TArray<int>& triangles);
51          void createPerlinMap(TArray<float>& perlinMap);
52      };
53
```

Рисунок 3.15 – Об’явлення класу ATerrain

На рисунку 3.16 продемонстрована найголовніша функція createPerlinMap. У функції обчислюється амплітуда, частота та додається шум Перліна для кожної клітини та створюється массив зі значень.

```
void ATerrain::createPerlinMap(TArray<float>& perlinMap)
{
    for (int x = 0; x < width; x++) {
        for (int y = 0; y < height; y++) {
            int index = (y * width + x);

            float value = 0;
            float freq = 1;
            float amp = 1;

            for (int i = 0; i < octaves; i++) {
                value += PerlinNoise::perlin(x: x * freq / scale, y: y * freq / scale) * amp;
                freq *= 2;
                amp /= 2;
            }

            // Contrast
            value *= 1.2;

            // Clipping
            if (value > 1.0f) {
                value = 1.0f;
            } else if (value < -1.0f) {
                value = -1.0f;
            }

            perlinMap.Add(value);
        }
    }
}
```

Рисунок 3.16 – Створення массиву значень шуму Перліна

В редакторі UE5 створюємо блупрінт клас, який наслідуємо від C++ классу ATerrain. (Рисунок 3.17). Створенний клас треба петянгути на сцену, для того щоб можна було налаштовувати його параметри.

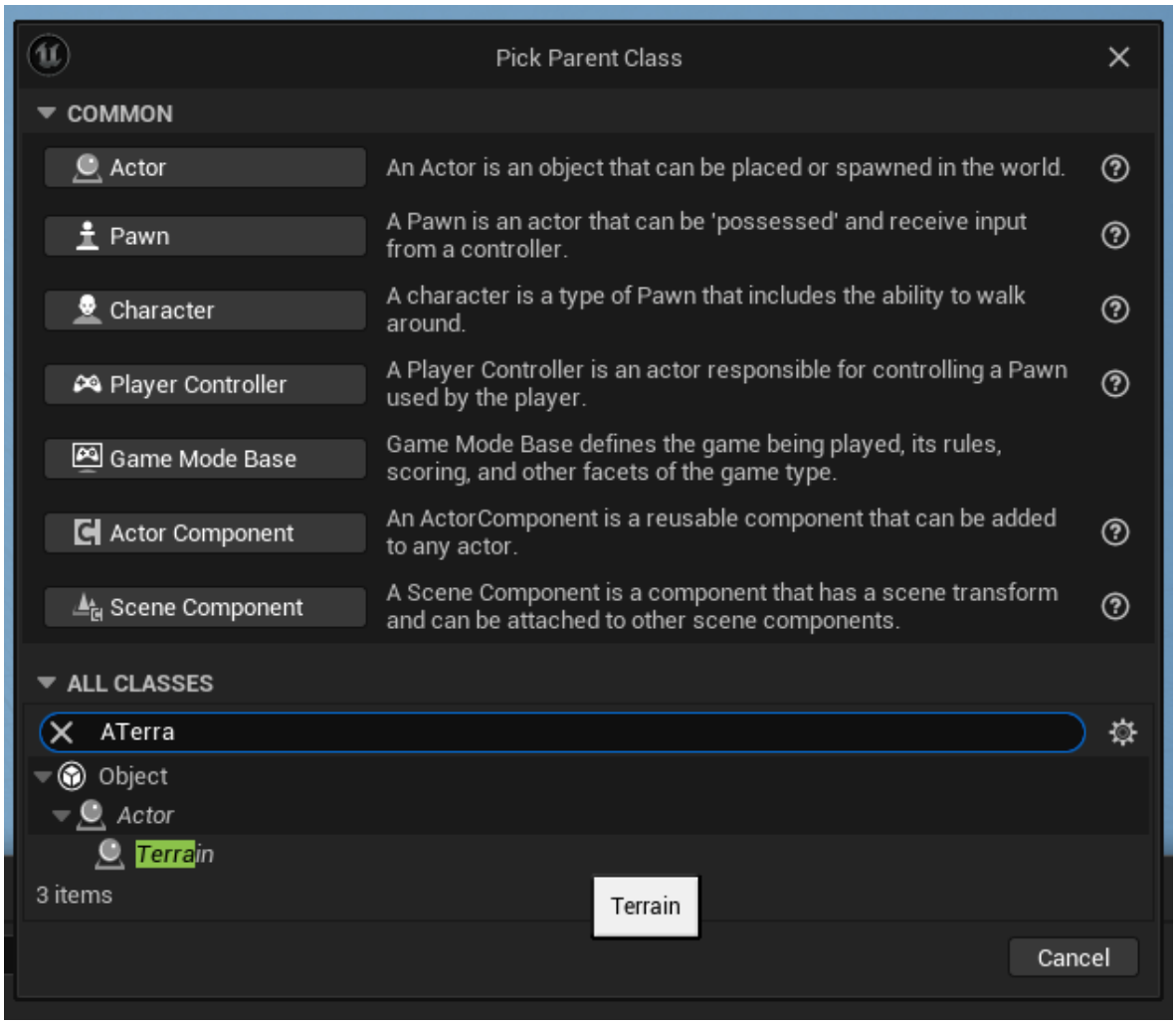


Рисунок 3.17 – Створення блупрінт класу

Налаштувавши параметри, натискаючи «Play», почнеться генерація ландшафту. На рисинку 3.18 продемонстровано генерація ландшафту з полем 128 на 128 клітин. Октав 12, маштаб клітини 100 юнітів.

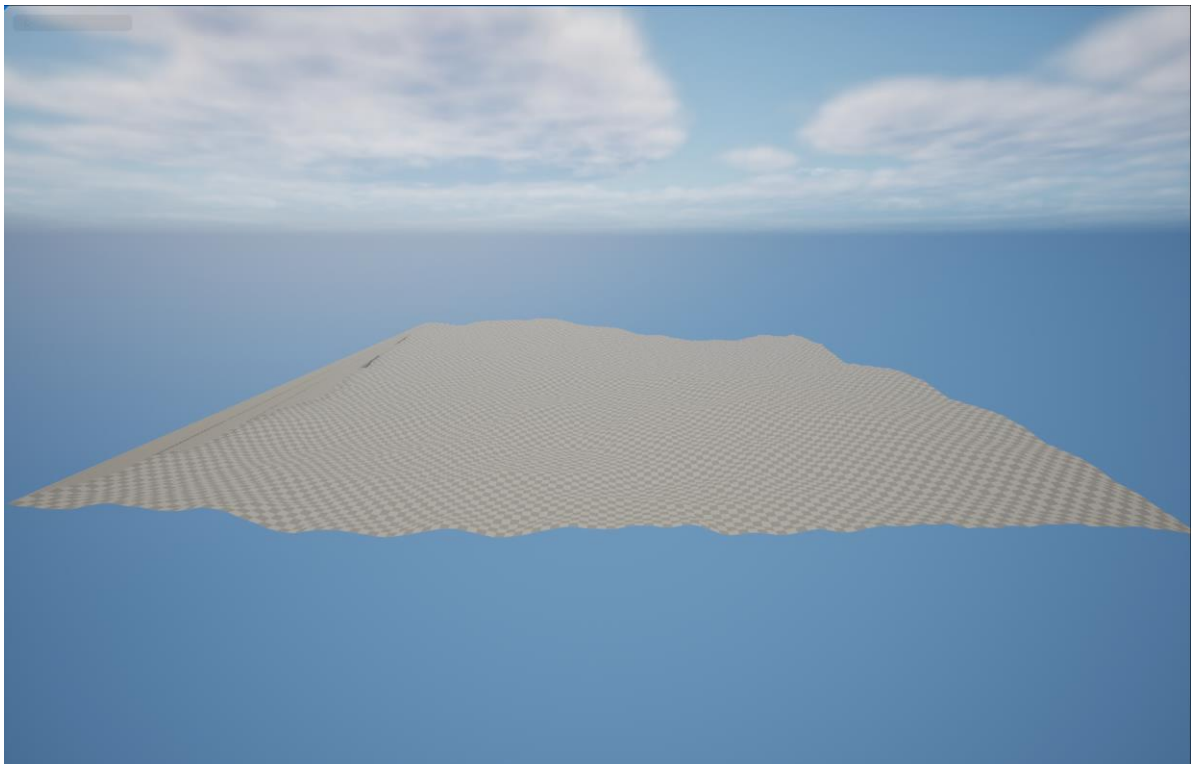


Рисунок 3.18 – Процедурно згенерований ландшафт

Можна додати параметер «Material» для того, щоб виглядало як зелений луг. (Рисунок 3.19). Чем менше октав, тим пласкіший ландшафт.

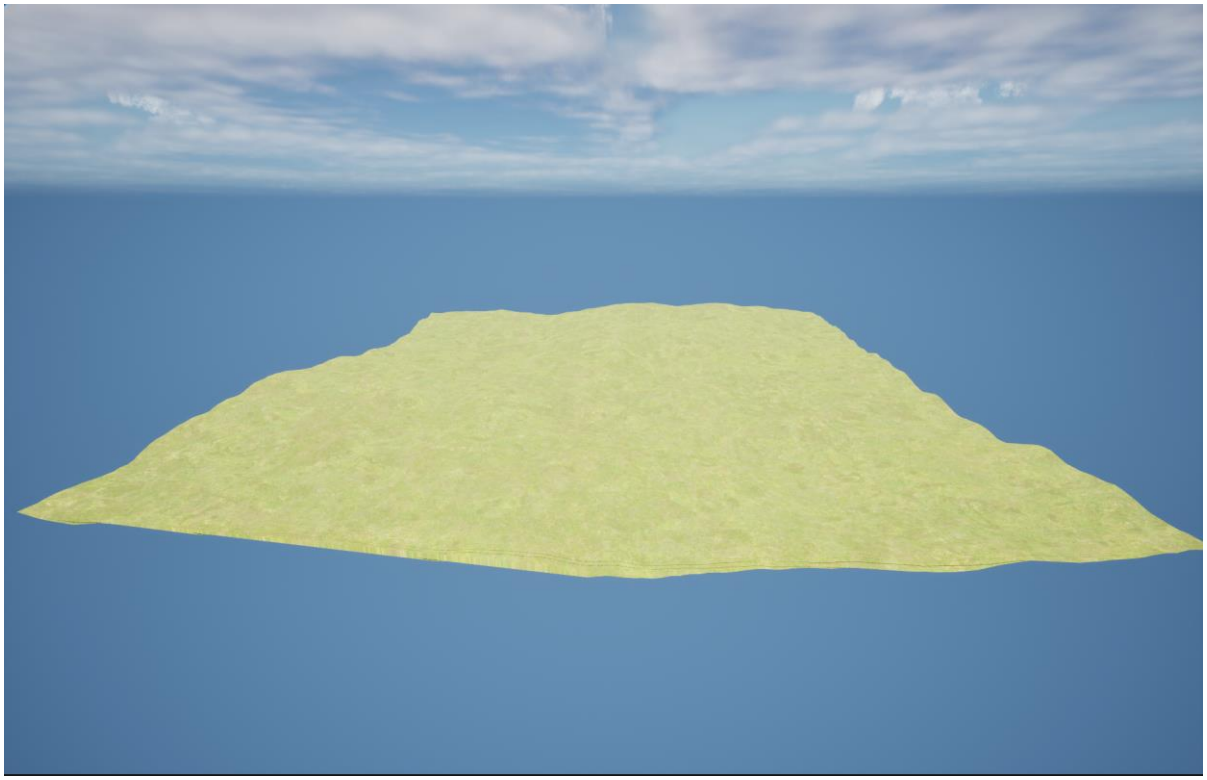


Рисунок 3.20 – Процедурно згенерований ландшафт з матеріалом

3.3 Створення незалежного застусунку

Перш ніж проект Unreal можна буде розповсюдити серед користувачів, його потрібно правильно запакувати. Упаковка гарантує, що весь код і вміст оновлені та мають належний формат для роботи на бажаній цільовій платформі.

У процесі пакування виконується кілька етапів. Спочатку буде скомпільовано вихідний код конкретного проекту. Після компіляції коду весь необхідний вміст буде перетворено або «приготовано» у формат, який може використовуватися цільовою платформою. Після цього скомпільований код і готовий вміст буде об’єднано в набір файлів, який можна розповсюджувати, наприклад інсталятор.

У головному меню «Файл» є пункт «Пакетувати проект» із підменю. Це підменю надає список усіх підтримуваних платформ, для яких ви можете запакувати свій проект.

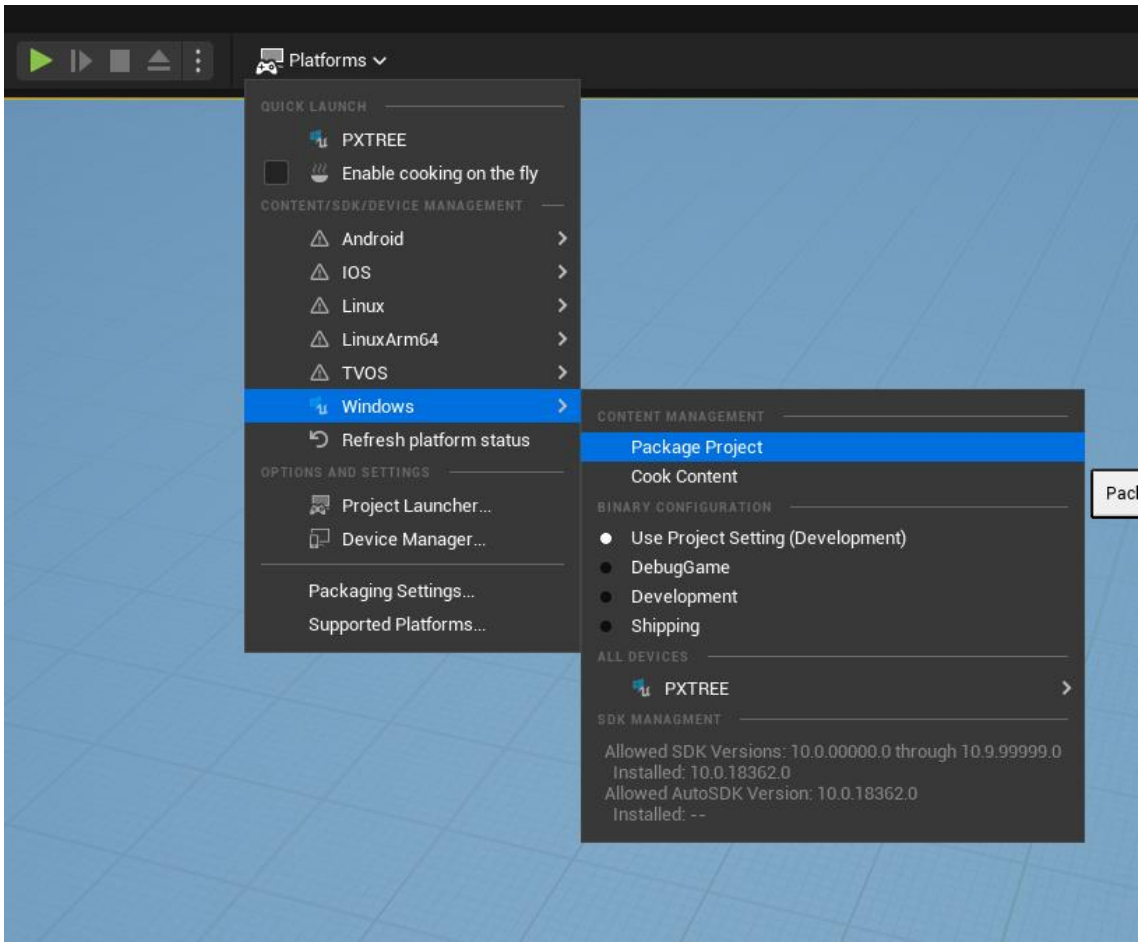


Рисунок 3.20 – Пакування проекту

Перш ніж пакувати гру, спочатку потрібно встановити стандартну карту гри, яка завантажуватиметься під час запуску упакованої гри. Якщо не налаштували карту то буде лише чорний екран під час запуску пакетної гри. Якщо було використано одну з карт шаблонів, як-от шаблон від першої особи або шаблон від третьої особи, буде завантажено початкову карту.

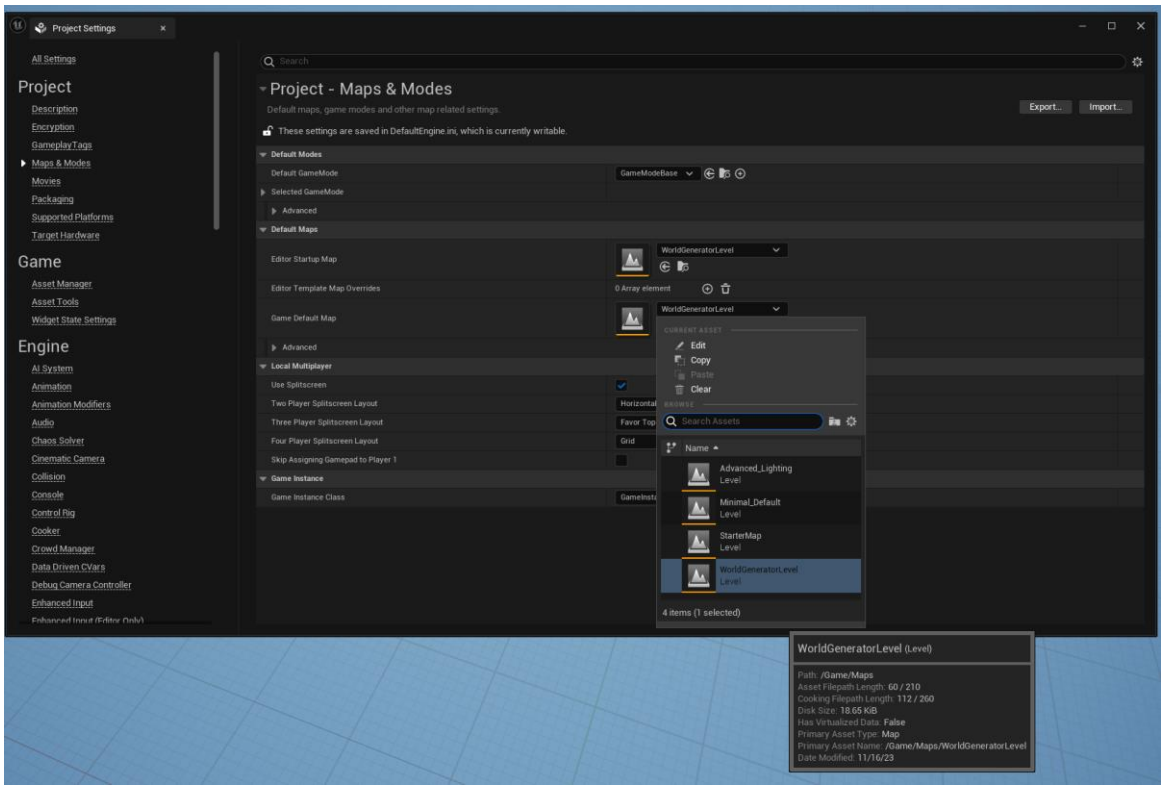


Рисунок 3.21 – Встановлення стандартну карту для гри

Буде запропоновано діалогове вікно для вибору цільового каталогу. Якщо пакування завершиться успішно, цей каталог міститиме упакований проект.

Після підтвердження цільового каталогу розпочнеться фактичний процес пакетування проекту для вибраної платформи. Оскільки пакування може займати багато часу, цей процес виконується у фоновому режимі, і ви можете продовжувати використовувати редактор. У нижньому правому куті Редактора відображатиметься індикатор стану, який вказуватиме на прогрес. (Рисунок 3.22)

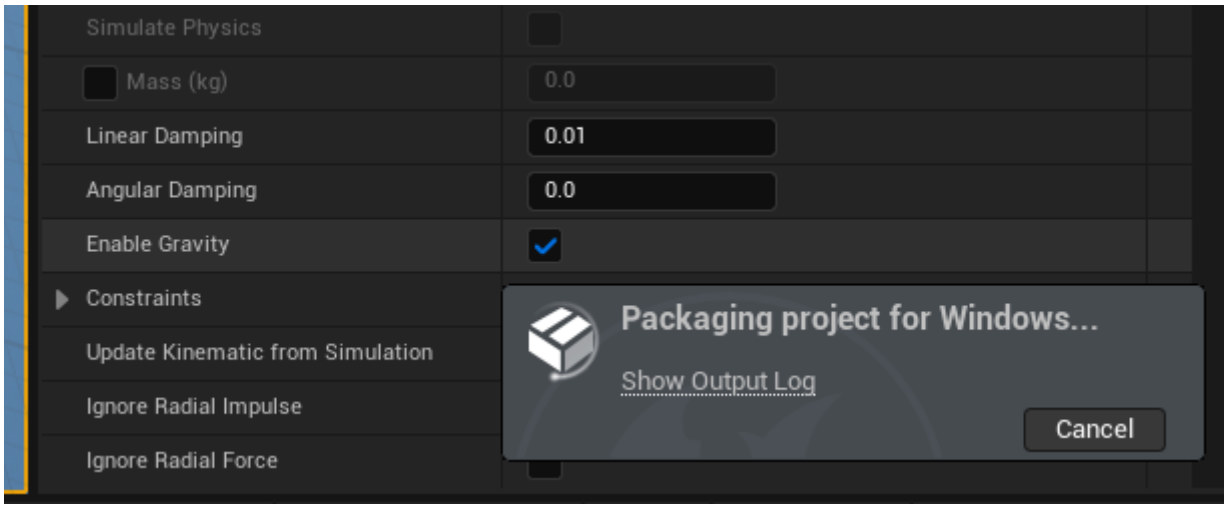


Рисунок 3.22 - індикатор стану пакування проекту

Індикатор стану також містить кнопку «Скасувати», щоб зупинити процес пакування. Крім того, посилання «Показати журнал» можна використовувати для відображення розширеної інформації журналу виводу, яка корисна для з'ясування того, що пішло не так, якщо процес пакування не вдається, або для виявлення попереджень, які можуть виявити потенційні помилки в продукті. (Рисунок 3.23)

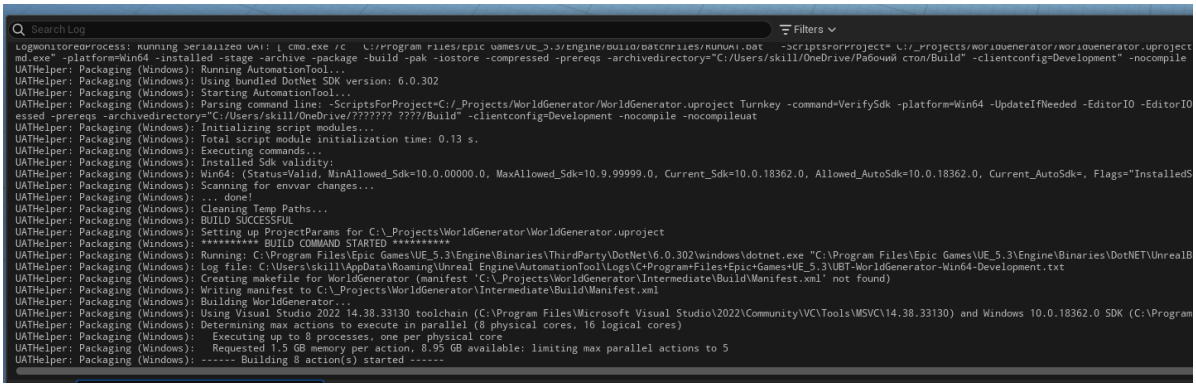


Рисунок 3.23 – Журнал виводу

Для більш тонкого налаштування пакування проекту, потрібно перейти Редагувати > Параметри проекту > Упаковка або Файл > Упакувати проект > Параметри упаковки.

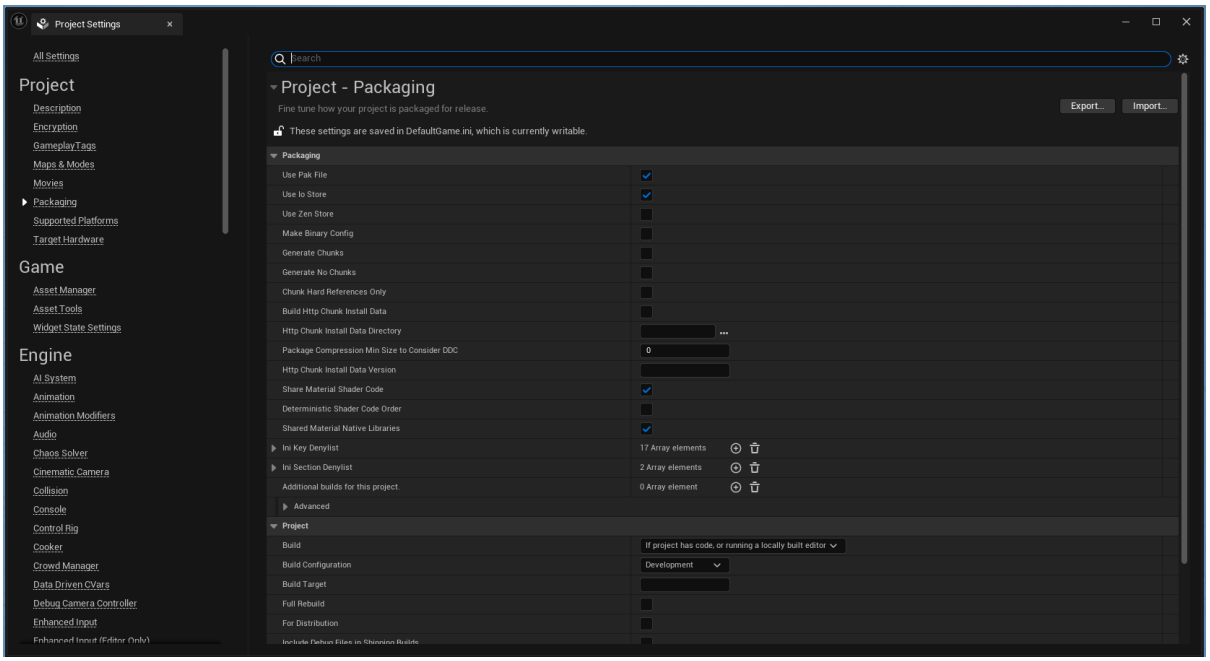


Рисунок 3.24 – Параметри упаковки проекту

Build Configuration – конфігурація збірки для компіляції вашого проекту на основі коду. Для налагодження проекту коду виберіть DebugGame. Для більшості інших розробок із мінімальною підтримкою налагодження, але кращою продуктивністю виберіть Розробка. Для остаточної збірки доставки, яка не матиме інформації про налагодження та функцій, орієнтованих на налагодження (таких як малювання форм налагодження або друк повідомлень налагодження на екрані), виберіть «Доставка».

Staging Directory – каталог, який міститиме вашу упаковану збірку. Це буде оновлено автоматично, коли ви виберете інший каталог у вибраному цільовому каталозі.

Full Rebuild – чи потрібно компілювати весь ваш код. Якщо вимкнено, буде скомпільовано лише змінений код. Це може прискорити процес пакування. Для транспортних збірок ви завжди повинні виконувати повне відновлення, щоб переконатися, що нічого не пропало або не застаріло. Цей параметр увімкнено за замовчуванням.

Use Pak File – упаковувати активи вашого проекту як окремі файли чи єдиний пакет. Якщо ввімкнено, усі ресурси буде поміщено в один файл .pak

замість копіювання всіх окремих файлів. Якщо у вашому проекті використовується багато файлів активів, використання файлу Pak може полегшити розповсюдження, оскільки це зменшує кількість файлів, які потрібно передати. Цей параметр вимкнено за замовчуванням.

Generate Chunks – чи генерувати фрагменти файлів .pak, які можна використовувати для потокового встановлення.

Build Http Chunk Install Data – чи генерувати дані для інсталятора блоку HTTP. Це дозволяє розмістити ці дані на веб-сервері, який буде встановлено під час виконання.

Http Chunk Install Data Directory – це каталог, куди буде встановлено дані після їх створення.

Http Chunk Install Data Version – це назва версії для даних інсталяції блоку HTTP.

Include Prerequisites Installer – це вказує, чи слід включати інсталятори для попередніх умов упакованих ігор, наприклад компонентів операційної системи, які можна розповсюджувати.

Після налаштування параметрів пакування можна пакувати проект. Якщо проект запакувався без помилок, то в директорії має з’явитися папка з назвої плафторми на яку ми збирали проект. У нашому випадку «Windows». В цій папці є декілька папок і файлів, в яких запакований проект. (Рисунок 3.24)

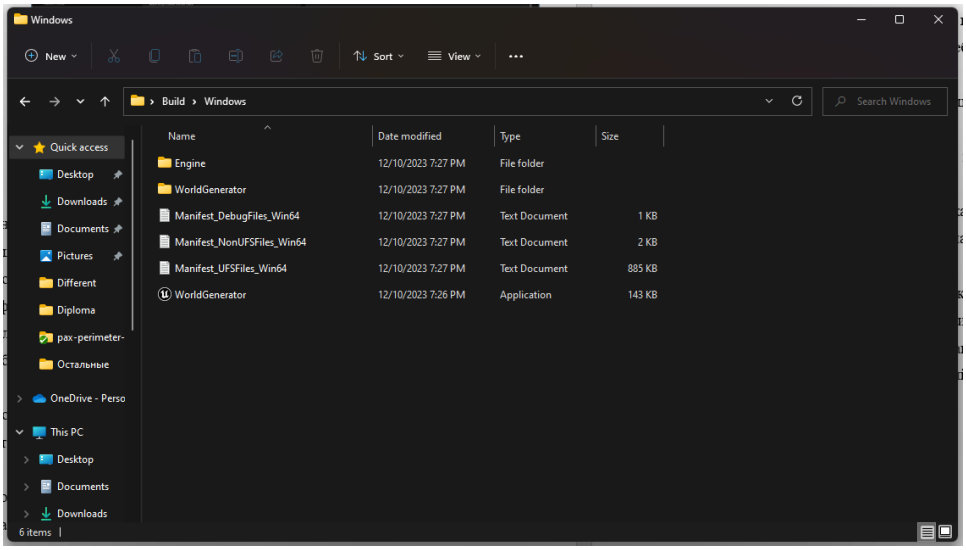


Рисунок 3.25 – Вид запакованого проекта

Для запуску гри потрібло відкрити файл WorldGenerator.exe. Після запуску ми побачемо генерування ландщафту з параметрами які ми налаштували ще в редакторі.

Рисунок 3.26 – Генератор ландшафту у незалежному застусунку

ВИСНОВКИ

Було проведено докладне дослідження можливостей використання алгоритму Perlin Noise у контексті процедурного генерування ландшафту для створення ігор. Отримані результати та виявлені в ході дослідження висновки дозволяють зробити кілька ключових висновків:

- Ефективність алгоритму Perlin Noise: Виявлено, що алгоритм Perlin Noise є дуже ефективним інструментом для генерації природних та реалістичних ландшафтів у відомих іграх. Його здатність до створення природних форм та текстур робить його особливо цінним для розробників ігор, які прагнуть до максимальної реалістичності віртуальних світів.
- Важливість налаштувань параметрів: Результати дослідження підкреслюють важливість належного налаштування параметрів алгоритму для досягнення бажаних ефектів у генерації ландшафту. Оптимальний вибір параметрів великою мірою залежить від конкретного завдання та вимог проекту.
- Можливості подальшого вдосконалення: Зазначено, що хоча алгоритм Perlin Noise є потужним інструментом, його можливості можна подальшим чином розширити і вдосконалити. Дослідження вказує на необхідність подальших наукових та технічних вивчень з метою розробки нових версій алгоритму або комбінування його з іншими методиками для досягнення ще більшої реалістичності та унікальності згенерованих ландшафтів.

Perlin Noise відіграє важливу роль у сучасній розробці ігор та має значний потенціал для подальшого розвитку та вдосконалення. Його використання сприяє не лише підвищенню реалістичності ігрових світів, але й сприяє творчій та інноваційній роботі розробників у створенні унікальних ігрових вражень.

ПЕРЕЛІК ПОСИЛАНЬ

1. Perlin noise [Електронний ресурс] Режим доступу: https://en.wikipedia.org/wiki/Perlin_noise
2. Perlin Noise: A Procedural Generation Algorithm [Електронний ресурс] Режим доступу: https://rtouti.github.io/graphics/perlin-noise-algorithm
3. Perlin Noise Maker [Електронний ресурс] Режим доступу: http://kitfox.com/projects/perlinNoiseMaker/
4. Generating Digital Worlds Using Perlin Noise [Електронний ресурс] Режим доступу: https://medium.com/nerd-for-tech/generating-digital-worlds-using-perlin-noise-5d11237c29e9
5. Landscape generation using procedural generation techniques Noise [Електронний ресурс] Режим доступу: https://s3.eu-central-1.amazonaws.com/ucu.edu.ua/wp-content/uploads/sites/8/2021/07/Melnychuk-Vladyslav_188572_assignsubmission_file_VladyslavMelnychuk.pdf
6. Fundamentals of Terrain Generation [Електронний ресурс] Режим доступу: https://www.cs.cmu.edu/~112/notes/student-tp-guides/Terrain.pdf
7. Visual Studio 2022 [Електронний ресурс] Режим доступу: https://visualstudio.microsoft.com/
8. Microsoft Visual Studio [Електронний ресурс] Режим доступу: https://uk.wikipedia.org/wiki/Microsoft_Visual_Studio
9. Rider part of dotUltimate Fast & powerful cross-platform .NET IDE [Електронний ресурс] Режим доступу: https://www.jetbrains.com/rider/
10. Rider для Unreal Engine Умная IDE с поддержкой C++ и Blueprints для разработки игр [Електронний ресурс] Режим доступу: https://www.jetbrains.com/ru-ru/lp/rider-unreal/
11. What Is Unreal Engine 5 (UE5)? [Електронний ресурс] Режим доступу: https://www.perforce.com/blog/vcs/unreal-engine-5
12. Unity (ігровий рушій) [Електронний ресурс] Режим доступу: https://uk.wikipedia.org/wiki/Unity_(%D1%96%D0%B3%D1%80%D0%BE%D0%

B2%D0%B8%D0%B9_%D1%80%D1%83%D1%88%D1%96%D0%B9)

13. Что такое Unity 3D [Електронний ресурс] Режим доступу: http://web.spt42.ru/index.php/chto-takoe-unity-3d

14. Procedural Mesh [Електронний ресурс] Режим доступу: https://docs.unrealengine.com/5.3/en-US/BlueprintAPI/Components/ProceduralMesh/

15. UnrealLink and RiderLink [Електронний ресурс] Режим доступу: https://www.jetbrains.com/help/rider/Unreal_Engine__UnrealLink_RiderLink.html

16. 6 причин, чому JetBrains Rider краще ніж Visual Studio [Електронний ресурс] Режим доступу: https://ua.softlist.com.ua/articles/6-prichin-pochemu-jetbrains-rider-lyche-chem-visual-studio/

17. Visual Studio Code [Електронний ресурс] Режим доступу: https://en.wikipedia.org/wiki/Visual_Studio_Code

18. UnrealLink [Електронний ресурс] Режим доступу: https://plugins.jetbrains.com/plugin/14989-unreallink

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра Інженерії програмного забезпечення автоматизованих систем

ДОСЛІДЖЕННЯ МОЖЛИВОСТЕЙ ВИКОРИСТАННЯ АЛГОРИТМУ PERLIN NOISE ДЛЯ ПРОЦЕДУРНОГО ГЕНЕРУВАННЯ ЛАНДШАФТУ ПРИ СТВОРЕННІ ІГР

Виконав: студент групи ІСД-61 Гераймович Д.С.
Керівник: Полоневич О. В.

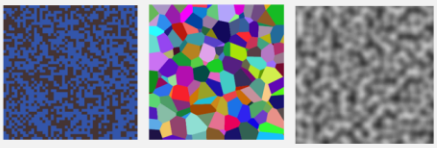
Київ - 2023

МЕТА, ОБ'ЄКТ, ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи** – виявити можливості та обмеження використання алгоритму Perlin Noise для генерації ігрового ландшафту.
- **Об'єкт дослідження** – алгоритми генерування ландшафту.
- **Предмет дослідження** – характеристики та параметри алгоритмів генерування ландшафту.

ВИДИ АЛГОРИТМІВ ГЕНЕРУВАННЯ ЛАНДШАФТІВ

- Алгоритм генерації одновимірних схилів
- Клітинний автомат генерації печер
- Шум Вороного
- Алгоритм ромбовидного квадрата
- Алгоритм Perlin Noise
- Алгоритм Simplex Noise



ПРИКЛАДИ ІГОР З ПРОЦЕДУРНОЇ ГЕНЕРАЦІЄЮ ЛАНДШАФТІВ

ВИБІР ПЛАТФОРМИ ДЛЯ РОЗРОБКИ ЗАСТУСУНКУ ГЕНЕРАЦІЇ 3D ЛАНДШАФТУ

ВИБІР СЕРЕДОВИЩА РОЗРОБКИ

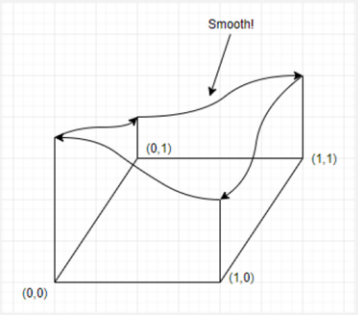
ПЕРЕВАГИ RIDER

- Rider працює набагато стабільніше і швидше ніж Visual Studio
- Rider є кросплатформним, він може працювати на платформах Windows, Mac або Linux з однаковою функціональністю і стабільністю.
- Середовище Rider включає більшість функцій, популярного розширення Visual Studio для розробників .NET – ReSharper.
- Рішення і проекти, з якими працює JetBrains Rider, повністю сумісні з Visual Studio, і не використовують власні формати.

ПРОЕКТУВАННЯ ЗАСТУСУНКУ ГЕНЕРАЦІЇ 3D ЛАНДШАФТУ

Головні задачі розробки застусунку:

- Створення сітки модели тривіріного ландшафту
- Імплементувати аглоритм Perlin Noise
- Реалізувати інтерфес для налащтування параметрів для генерування ландшафту
- Запакувати проект у незалежний застусунок



РЕАЛІЗАЦІЯ АЛГОРИТМУ PERLIN NOISE

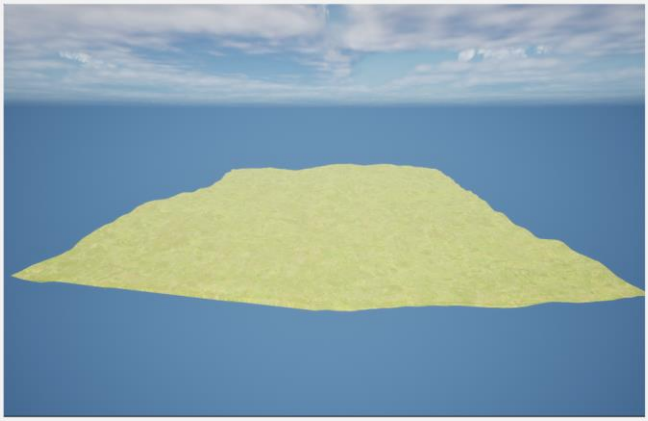
```cpp
// Sample Perlin noise at coordinates x, y
float PerlinNoise::perlin(float x, float y) {

    // Determine grid cell corner coordinates
    int x0 = (int)x;
    int y0 = (int)y;
    int x1 = x0 + 1;
    int y1 = y0 + 1;

    // Compute Interpolation weights
    float sx = x - (float)x0;
    float sy = y - (float)y0;

    // Compute and interpolate top two corners
    float n0 = dotGridGradient(x0, y0, x, y);
    float n1 = dotGridGradient(x1, y0, x, y);
    float ix0 = interpolate(n0, n1, sx);

    // Compute and interpolate bottom two corners
    n0 = dotGridGradient(x0, y1, x, y);
    n1 = dotGridGradient(x1, y1, x, y);
    float ix1 = interpolate(n0, n1, sx);

    // Final step: interpolate between the two previously interpolated values, now in y
    float value = interpolate(ix0, ix1, sy);

    return value;
}
```


РЕАЛІЗАЦІЯ ІНТЕРФЕЙСУ НАЛАШТУВАННЯ ПАРАМЕТРІВ ГЕНЕРУВАННЯ

```cpp
class UProceduralMeshComponent;

UCLASS(Blueprintable)
class ATerrain : public AActor
{
    GENERATED_BODY()
public:
    ATerrain();

    UPROPERTY(EditAnywhere, Category="Generation")
    int width;

    UPROPERTY(EditAnywhere, Category="Generation")
    int height;

    UPROPERTY(EditAnywhere, Category="Generation")
    int octaves;

    UPROPERTY(EditAnywhere, Category="Generation")
    float fBias;

    UPROPERTY(EditAnywhere, Category="Generation")
    float seed;

    UPROPERTY(EditAnywhere, Category="Generation")
    float scale;
```

```cpp
    UPROPERTY(EditAnywhere, Category="Generation")
    float UVScale;

    UPROPERTY(EditAnywhere, Category="Generation")
    UMaterial* terrainMaterial;

    void generate();

protected:

    UPROPERTY(EditDefaultsOnly)
    UProceduralMeshComponent* _terrainMesh;

    virtual void BeginPlay() override;

private:
    void createVertices(TArray<FVector>& verticles, TArray<FVector2D>& uv0);
    void createTriangles(TArray<int>& triangles);
    void createPerlinMap(TArray<float>& perlinMap);
};
```

РЕЗУЛЬТАТИ ГЕНЕРУВАННЯ



ВИСНОВКИ

Було проведено докладне дослідження можливостей використання алгоритму Perlin Noise у контексті процедурного генерування ландшафту для створення ігор.

- Виявлено, що алгоритм Perlin Noise є дуже ефективним інструментом для генерації природних та реалістичних ландшафтів у відомих іграх. Його здатність до створення природних форм та текстур робить його особливо цінним для розробників ігор, які прагнуть до максимальної реалістичності віртуальних світів.
- Результати дослідження підкреслюють важливість належного налаштування параметрів алгоритму для досягнення бажаних ефектів у генерації ландшафту. Оптимальний вибір параметрів великою мірою залежить від конкретного завдання та вимог проекту.
- Зазначено, що хоча алгоритм Perlin Noise є потужним інструментом, його можливості можна подальшим чином розширити і вдосконалити. Дослідження вказує на необхідність подальших наукових та технічних вивчень з метою розробки нових версій алгоритму або комбінування його з іншими методиками для досягнення ще більшої реалістичності та унікальності згенерованих ландшафтів.