МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Кваліфікаційна робота бакалавра

# ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ІГРОВОЇ СИСТЕМИ НА ОСНОВІ UNREAL ENGINE 4

Здобувач освіти гр. ІН – 81 Нікіта АВРАМЕНКО

Науковий керівник,
доцент, кандидат техн. наук Віктор АВРАМЕНКО

Завідувач кафедри
доктор технічних наук, професор Анатолій ДОВБИШ

СУМИ 2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Затверджую_____
Зав. кафедрою Довбиш А.С.
"_____" _____2022 р.

**ЗАВДАННЯ**

**до кваліфікаційної роботи бакалавра**

здобувача вищої освіти четвертого курсу, групи ІН-81 спеціальності «122 – Комп'ютерні науки» денної форми навчання Авраменка Нікіти Олексійовича.

**Тема: «Інформаційне та програмне забезпечення ігрової системи на основі Unreal Engine 4»**

Затверджена наказом по СумДУ
№_____ від _____2022 р.

**Зміст пояснювальної записки**: 1) літературний огляд за обраною тематикою роботи; 2) постановка завдання для розробки; 3) практична реалізація.

Дата видачі завдання «_____»_____ 2022 р.
Керівник роботи _____ Віктор АВРАМЕНКО
Завдання прийняв до виконання _____ Владислав КАРПЕНКО

# РЕФЕРАТ

**Записка**: 45 стр., 49 рис., 1 додаток, 16 літературних джерел.

**Об’єкт дослідження** — інформаційне та програмне забезпечення ігрової системи на основі Unreal Engine 4

**Мета роботи** — створення демо-версії гри на движку Unreal Engine 4.

**Результати** — Створено програмне забезпечення з метою побудувати арену для комп’ютерної гри. Воно має перевагу в тому, що являє собою скорочений шаблон для розробки комп’ютерних ігор і заодно це навчальний матеріал для роботи в Unreal Engine 4.

ІНФОРМАЦІЙНА СИСТЕМА, UNREAL ENGINE 4, ДВИЖОК, BLUEPRINTS, КОМП'ЮТЕРНІ ІГРИ, АРЕНА.

# ЗМІСТ

# ВСТУП

Ігрова індустрія зародилася ще у минулому столітті, і з того самого часу її розвиток йде в ногу з розвитком всієї індустрії інформаційних технологій. Нині ігри - не лише розвага, це поєднання технологій з мистецтвом. Вони мають у собі сюжет та персонажів, красиво змодельовані моделі та локації, прекрасну музику і тд, це і є література, художнє та музикальне мистецтва відповідно. Проте в першу чергу ігри - це бізнес. Кожного року багато компаній випускають як великі ААА-проекти, так і невеликі інді-ігри, продаж котрих приносить гроші цим компаніям. На відміну від звичайного програмного забезпечення, ідей для створення ігор набагато більше, бо ПЗ спирається лише на функціональність, такі програми вже є в великій кількості, складнувато винайти оригінальну ідею, щоб в неї не було аналогів, в тей час як у ігор функціональність не так важлива, головне розважити користувача, а для цього можна придумати багато різних способів. Проте, хоч в ігровій індустрії легше придумати цікаву ідею, реалізувати її складніше. В першу чергу із-за різноманітності необхідних спеціалістів: це і сценаристи(якщо в грі буде сюжет), і художники по персонажам, і художники по локаціям, гейм-дизайнери, програмісти(як звичайні, так і мережеві), тестувальники, музиканти тощо. Та навіть при наявності усіх спеціалістів зазвичай розробка займає по декілька років. Проте також треба пам’ятати про бюджет. Наприклад інді-студії великими бюджетами не володіють, проте при відносно невеликому кошту розробки, якщо гра зможе знайти свою аудиторію, розробка принесе прибуток, а також, що головне, на мою думку, принесе репутацію компанії, завдяки чому, про неї дізнається більше людей, а це значить кращі продажі наступного проекту, а також потенційні інвестори. Проте, інвестори не можуть просто покладатися на репутацію компанії, вони мають бачити в що вкладуються. Для цього і створюють демо-версії ігор - неповна версія, що показує приблизний вектор розвитку, котру показують інвесторам або потенційним користувачам, з якого в майбутньому буде зроблена повноцінна гра.

Метою роботи є саме створення невеликої демо-версії за допомогою сучасного ігрового движку Unreal Engine 4.

# 1. ЛІТЕРАТУРНИЙ ОГЛЯД

## 1.1 Інструменти для розробки

В теперішній час майже ніхто не виконує розробку гри з нуля. Для цього використовують готове середовище розробки з необхідними інструментами. Таке середовище називається ігровим движком. Щоб обрати підходящий, візьмемо кілька найпопулярніших, порівняємо та оберемо.

Серед порівнюваних движків будуть наступні:

1. Unity
2. Unreal Engine 4
3. Godot
4. Game Maker Studio

Unity

Дуже популярний безкоштовний ігровий движок. В ньому об’єднані різноманітні програмні ресурси, такі як текстовий редактор, відладчик, компілятор та інше. Поріг входження в цьому движку доволі низький. Перш за все у движку використовується компонентно-орієнтований підхід, завдяки котрому розробник створює об’єкт та до нього додає різноманітні компоненти, завдяки зручному інтерфейсу та графічному редактору можна створювати карти, розставляючи об’єкти в режимі реального часу, та одразу тестувати свою роботу.

По-друге, движок має обширну бібліотеку ассетів та плагінів, котрі значно поскорюють розробку. Можна також додавати заготовки рівнів, 3d моделей персонажів і навіть патернів поводження ботів. Багато з них доступні безкоштовно, але за деякі прийдеться платити.

По-третє, движок підтримує багато платформ, окрім ПК, хоча це на даний момент не дуже потрібно.

Проте, якщо розробляється не клікер або звичайний платформер, то для розробки знадобляться поглибленні знання мови C#.

Також цей движок є трохи повільним. Він не підходить для створення масштабних сцен з великою кількістю персонажів.

Також в безкоштовній версії є багато обмежень.

Ці висновки були зроблені на основі особистого огляду движку, а також основуючись на досвіді гри у проекти на цьому движку, а також офіційної документації [1].

Unreal Engine 4

Серед порівнюваних движків цей має найкращу графіку. Як і в Unity поріг входження достатньо малий, проте, на відміну від нього, повністю безкоштовний. Платити знадобиться лише в тому випадку, коли дохід від вашого продукту перевищує 3000$ за квартал.

Також є багата кількість гайдів та стрімів від розробників движку, котрі на пальцях роз'яснюють роботу того чи іншого елементу функціоналу.

Хоча при всіх плюсах движку, він найбільш вибагливий до системних характеристик та більше всіх займає дискового простору.

Ці висновки також були винесені завдяки особистому використанню движку, ігор, створених на ньому, та офіційній документації [2].

Godot

Цей движок має відкритий вихідний код, який можна редагувати як захочеться.

Хоч він і має кросплатформеність, але не підтримує нинішнє покоління ігрових консолей.

Підтримує одразу декілька мов програмування: C++, C#, GDScript, Visual Script.

Цей движок, а також ігри, створені в ньому, займають менше дискового простору, у порівнянні з тими ж Unity та Unreal Engine, бо має набагато менше об'єктів, що може слугувати як плюсом так і мінусом.

Проте інструментарій Godot-а більше підходить для створення 2D ігор, він не дуже заточений під 3D. На відміну від попередніх движків, такі речі як виділення та звільнення пам'яті, управління шейдерами та

матеріалами ви повинні прописувати самі. Рендеринг та система управління пам’яттю не дуже оптимізовані. Велика кількість матеріалів у сцені дуже сильно вплине на кількість кадрів у секунду.

Хоч в Godot можна робити красиві ігри, налаштовувати їх складніше, ніж у попередніх движках.

Даны висновки були винесені на основі документації [3] та зібраних автором відгуків від користувачів.

Game Maker Studio

Цей движок гарно підходить для створення 2D ігор, але хоч він і має інструменти для роботи з 3D графікою вона виглядає злегка постарілою та програє у порівнянні з графікою інших движків.

Движок має інструменти для створення своїх спрайтів, звуків, анімації.

Також має кросплатформеність і заготовки деяких ігрових об’єктів, проте заготовки не безкоштовні.

Ігри можна створювати як за допомогою графічного інтерфейсу, так і за допомогою влаштованої мови GML, створеної на основі Delphi, котра має синтаксис JavaScript та Pascal.

Дані висновки були винесені на основі документації [4] та ігор, використовуючих цей движок.

Отже, треба обрати один з них.

Одразу відкидається варіант використання Game Maker Studio, так як задачею стоїть створення саме 3d гри. Приклади таких ігор на цьому движку складно знайти, що вже йде на користь зробленому рішенню. Наприклад, Global Anarchy 3 [5], про цю гру та подібні їй не знає майже ніхто, і це не дивно, бо графіка залишилася на рівні середини позаминулого десятиліття.

Godot також відкидається. Як вже зазначалося, створення 3d проектів на цьому движку більш складне та ризиковане(проект може просто бути погано оптимізован для гри), в тей час, як у інших залишившихся кандидатів такої проблеми немає.

Залишилися лиши Unity та Unreal Engine 4. Взагалі, серед комьюніті постійно йдуть суперечки, котрий з движків краще. Проте для великих компаній зазвичай вибір ситуаційний, залежно від проекту. Зазвичай при створенні 3d ігор надають перевагу Unreal Engine 4, а при створенні 2d - Unity. Про це можна судити виходячі зі статистики виходу ігор на обидвах движках: [6] для Unreal Engine 4 та [7] для Unity. Про різницю у графіці також можна судити виходячи з ігор, котрі зараз у розробці, так наприклад узяти Escape from Tarkov [8], що ще з 2017 року знаходить в стані бета-версії, та гру Atomic Hearts [9], що має вийти під кінець 2022-го року. Різницю в їх графіці можна легко побачити. В основному любов до Unity викликана за рахунок того, що він раніше став безкоштовним. Проте інструментарій Unreal Engine 4 більш зручний, наприклад створення кат-сцени у ньому спрощено так, що можна одразу приступити до процессу без поглиблених знань, у той час як у Unity на це знадобиться витратити 1-2 дні на освоєння. Найважливішою перевагою Unreal Engine 4 є Blueprints - система візуального програмування, яка в рази пришвидшує і полегшує розробку. Хоч у Unity тепер також є аналог цієї системи, проте він з’явився лише під кінець 2020 року, в тей час як система Blueprints була в Unreal Engine 4 ще з 2014 року, тому і функціонал в неї більш обширний. Також Unreal Engine 4 підтримує мову програмування C++, проте є модуль, за допомогою якого можна інтегрувати C#.

## 1.2 Постановка задачі

Створити програмне забезпечення з метою побудувати арену для комп’ютерної гри. Передбачити виконання наступних пунктів:

1. Побудова локації завдяки внутрішнім інструментам bsp геометрії;
2. Створення та застосування матеріалів, аналога текстур;
3. Робота з освітленням;
4. Налаштування анімації та взаємодії з інтерактивними об’єктами;
5. Створення додаткових механік пересування персонажу;
6. Реалізація механіки стрільби;
7. Налаштування ботів та їх поведінки;
8. Реалізація показника здоровья персонажа та його смерть;
9. Доповнення.

### 1.3 Вибір методу розв’язання задачі

На основі всіх переваг та недоліків, в решті решт, за основу розробки гри було обрано движок Unreal Engine 4.

В ході розробки вистачить одного лише Unreal Engine 4 з його інструментарієм. Роботу з освітленням, матеріалами, анімацією та моделюванням можна зробити в самому движку, більш складні моделі можна скачати з інтернету. Для розробки буде обрано один з готових шаблонів, подальша розробка буде проходити за допомогою технології Blueprints, як вже зазначалося це система візуального програмування, тобто з її допомогою не прийдеться писати код власноруч, замість цього треба маніпулювати вузлами(node), кожен з яких представляє ту або іншу функцію, прописану у движку або створену самим розробником. Тобто, наприклад, замість того щоб текстом писати cout << “Hello, world” треба просто витягнути відповідний вузол PrintString та у поле виводу вписати необхідне значення.

# 2 МЕТОДИКА ВИРІШЕННЯ ПОСТАВЛЕНОЇ ЗАДАЧІ

## 2.1 Створення проекту

Для початку роботи потрібно створити проект. На рисунку 1.1 показано меню створення та відкриття існуючих проектів. Щоб створити, просто тиснемо двічі на кнопці “Games”.

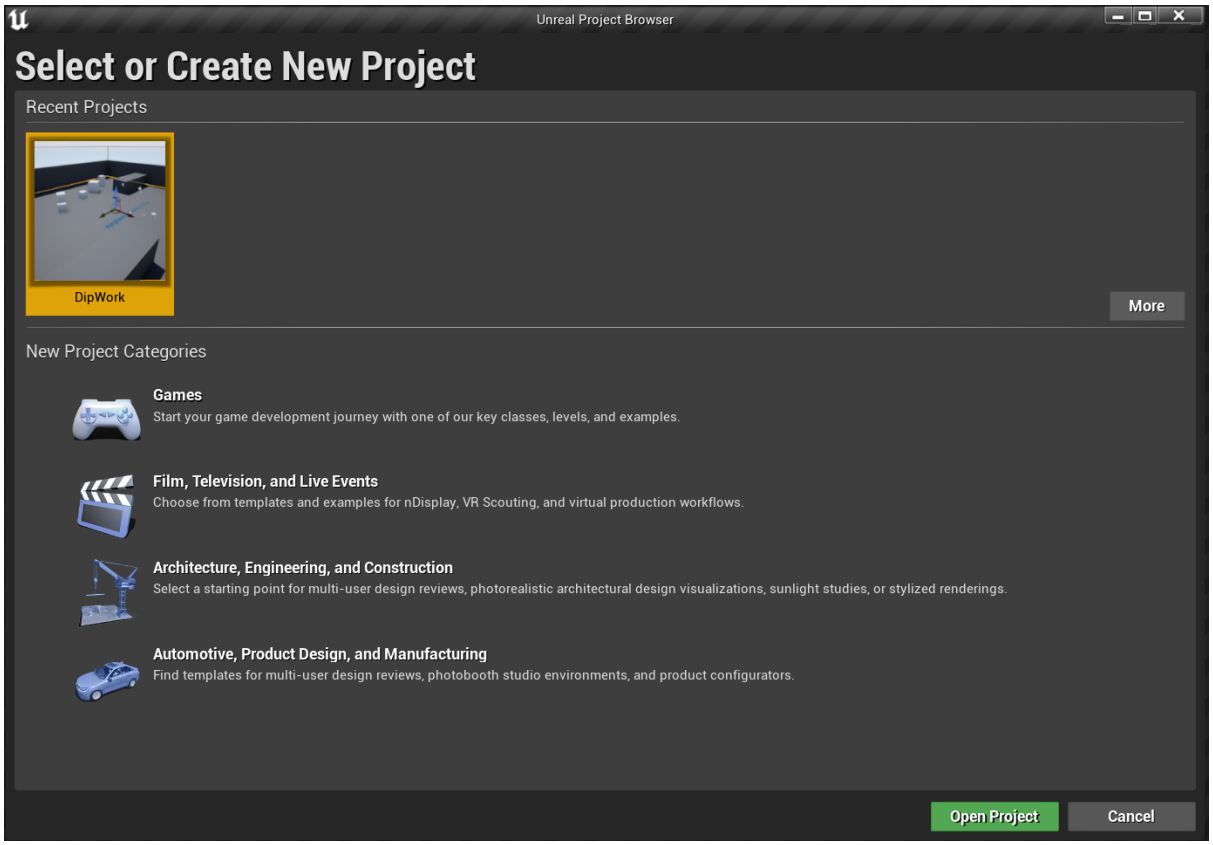



рис 1.1

Далі на рисунку 1.2 бачимо меню з вибором шаблонів. Для проекту оберемо шаблон First Person.

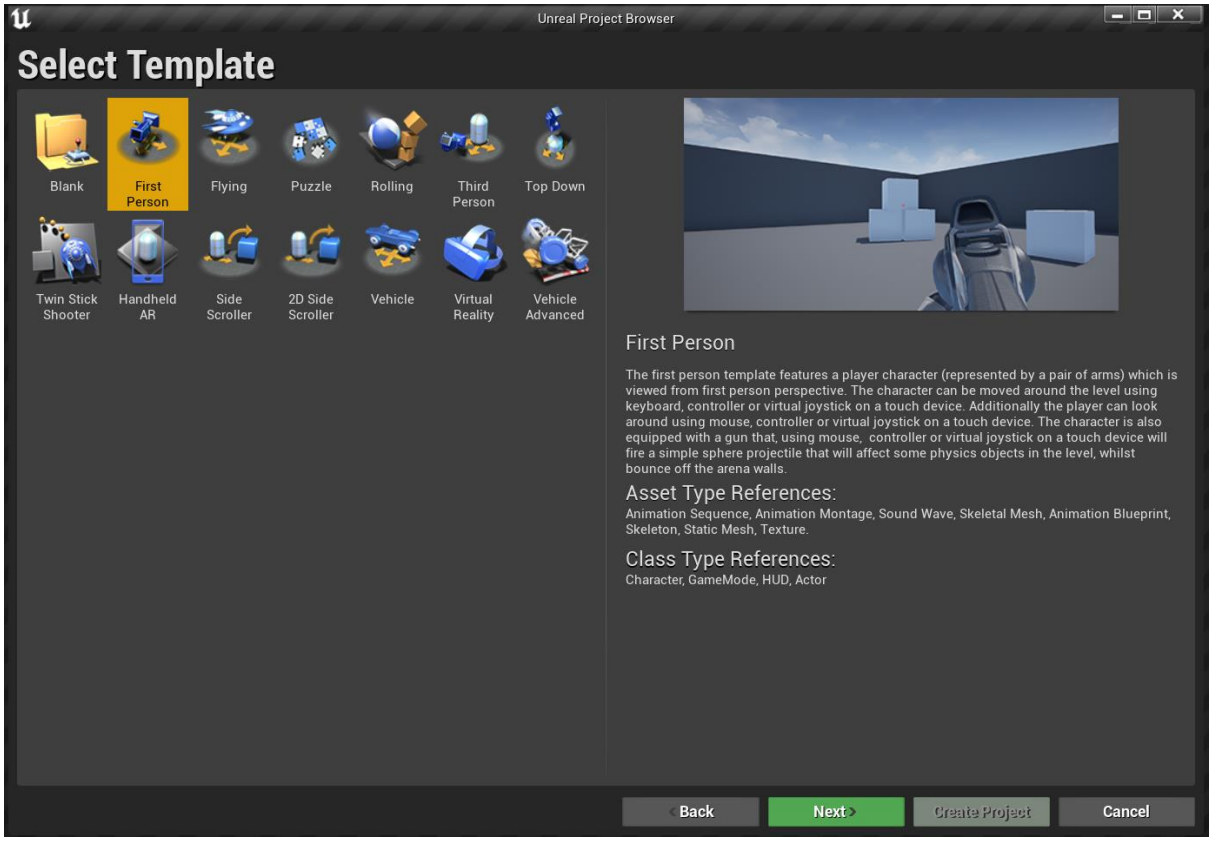



рис 1.2

І на рисунку 1.3 зображено останній крок при створенні проекту. Тут обираються налаштування та директорія для проекту.

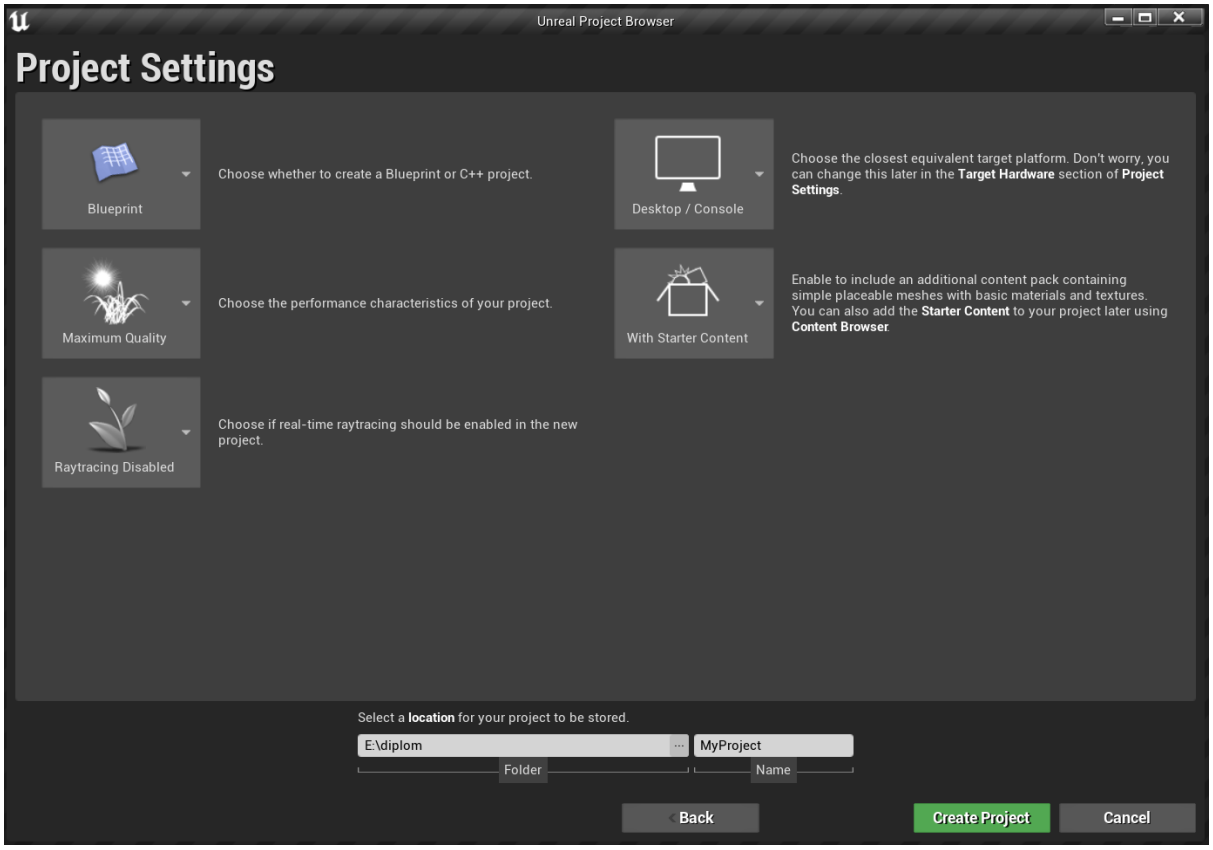



рис 1.3

І ось на рисунку 1.4 показано створений проект зі стартовим контентом. Тут є невелика кімната з кубами, джерелом світла та ігровим персонажем, котрий являє собою камеру від першого лиця, прикріплені до неї руки, зброю, яка стріляє круглими кульками та прописані механіки ходьби та стрибка. Пізніше все це буде або змінено, або просто прибрано.

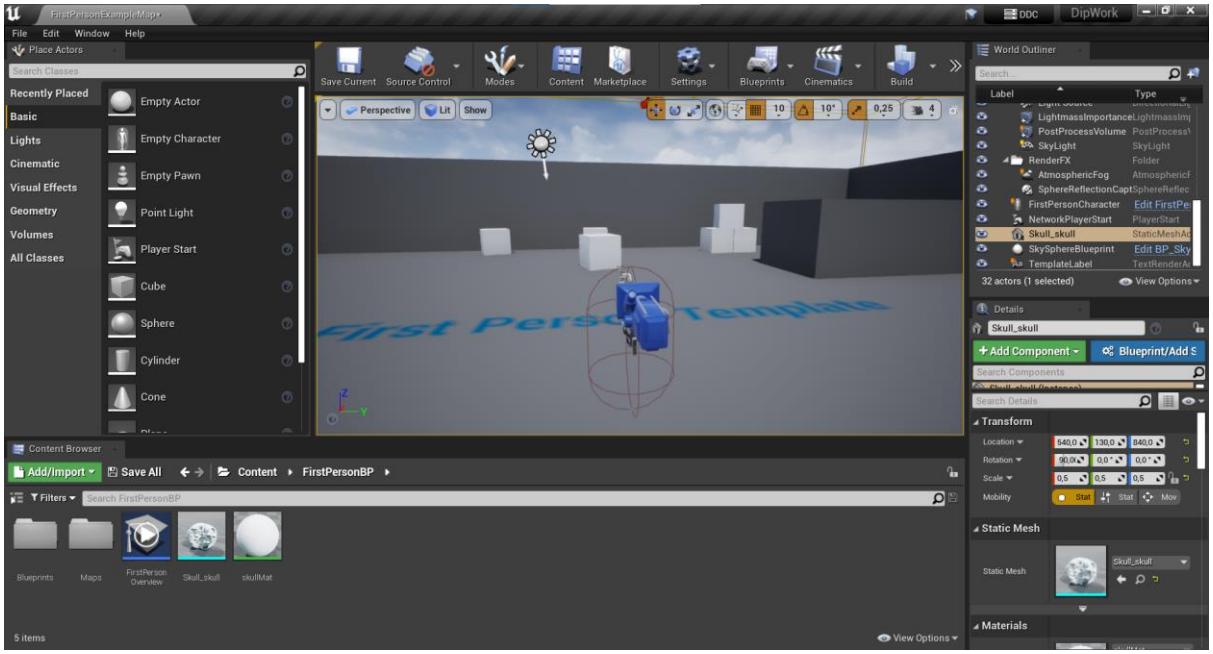


рис 1.4

## 2.2 Побудова локації

На минулому малюнку можна бачити, що зліва є редактор, котрий містить примітиви геометрії(куби, сфери, циліндри). З їх допомогою, а також допомогою деяких готових моделей, буде будуватися локація. Фігури можна як додавати до сцени, так і відрізати їх від інших об’єктів. На рисунку 2 можна бачити попередній результат. В ході роботи локація буде приймати інший вигляд.

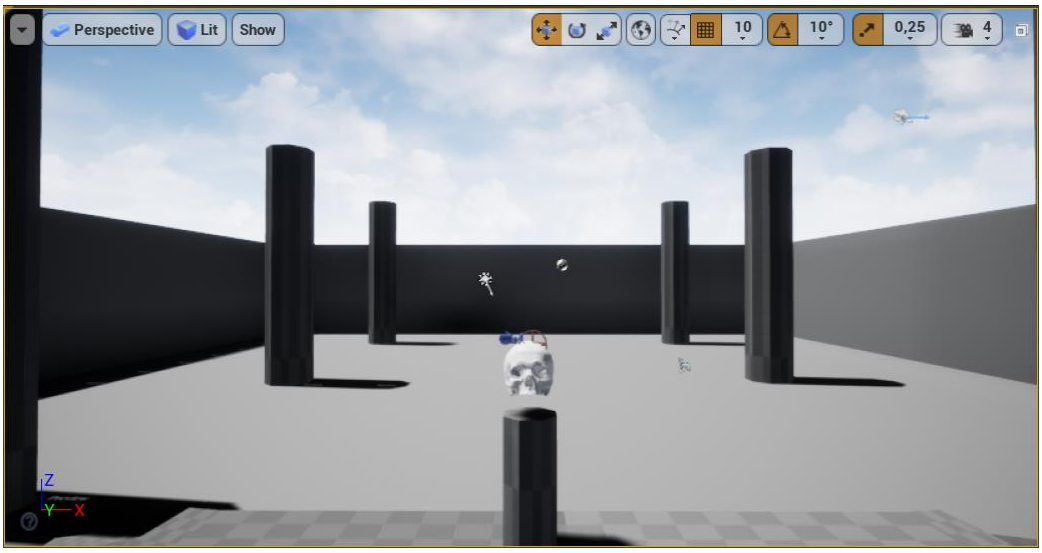



рис. 2

В кінці кімнати стоїть п'єдестал з черепом, по бокам від нього заокруглені сходи, в залі стоять шість колон. Далі ще будуть додані стеля, в другому кінці кімнати будуть розташовані ворота, які будуть відчинятися при натисканні кнопки взаємодії на череп, також будуть включатися джерела світла розташовані на колонах.

## 2.3 Робота з матеріалами

В Unreal Engine 4 неможливо просто натягнути текстуру на об’єкт. Для придання потрібного вигляду тут використовуються матеріали. Для його створення просто тиснемо правою кнопкою миші у контент браузері і обираємо функцію створення матеріалу та даємо йому назву. Після цього двічі тиснемо на нього та можемо починати налаштування. Див. рисунок 3.1.

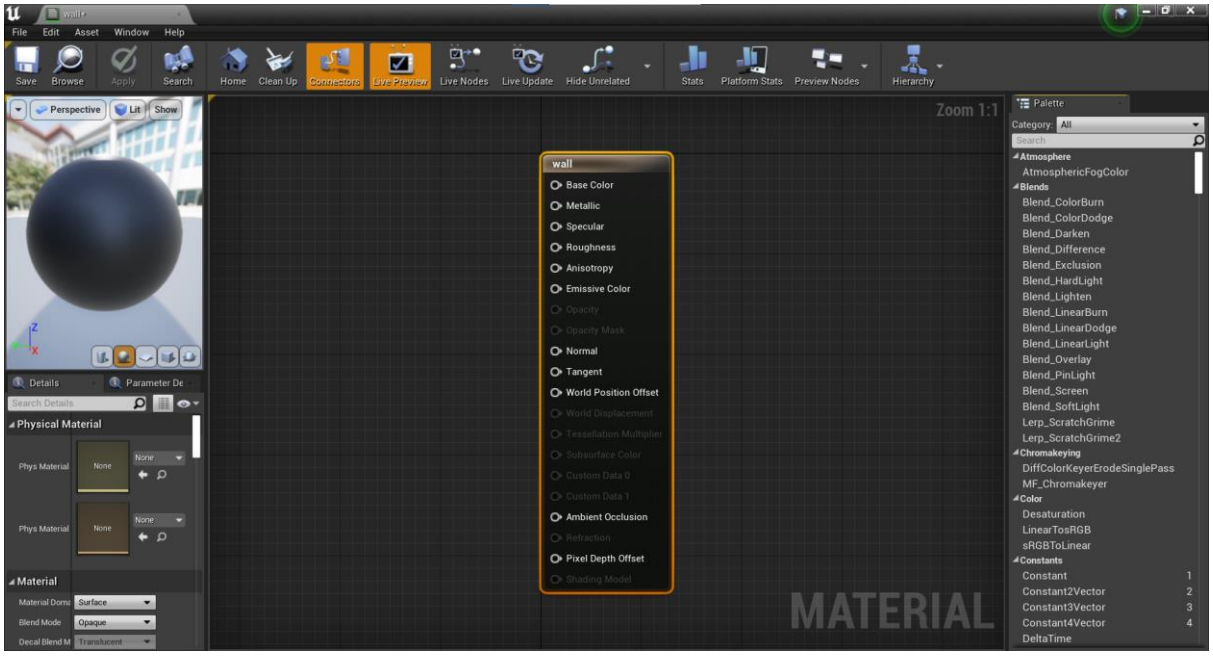



рис. 3.1

Звісно для побудови матеріалу вистачить і лише однієї текстури, яку під’єднають до каналу Base Color, проте зазвичай радять використовувати ще хоча б одну текстуру, з’єднану з каналом Normal, яка буде слугувати текстурою нормалі. На рисунку 3.2 можна бачити приклад виготовленого матеріалу, що буде використовуватися на стінах.

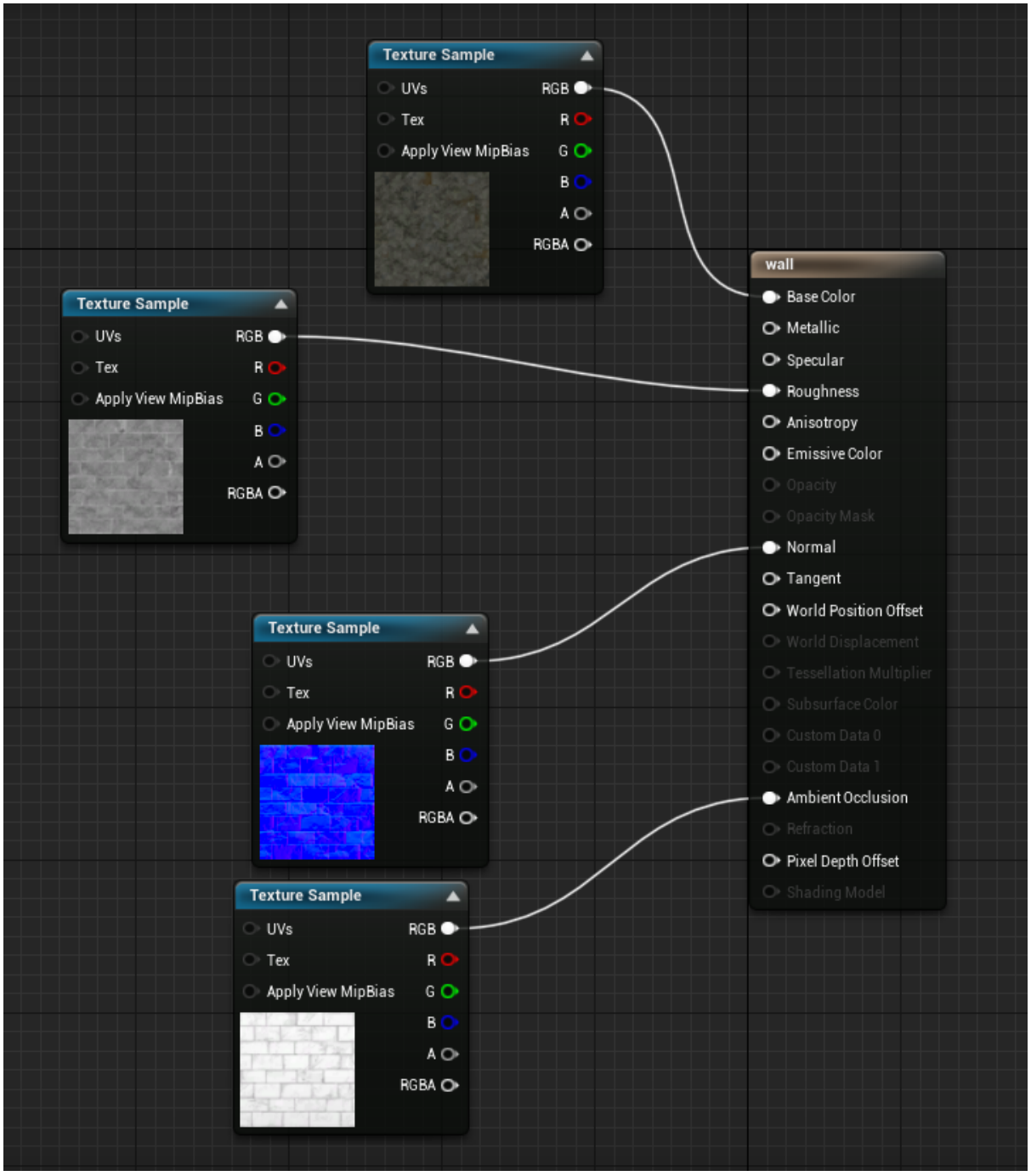



рис 3.2

Далі створимо матеріал для черепа, що буде світитися. Для цього не знадобиться жодної готової текстури, спочатку створюється вузол, котрий буде містити в собі параметри кольору, обирається звичайний білий. Потім створюється константа і змінюється її значення, за замовчуванням значення дорівнює 1, чим більше значення буде стояти, тим сильніше буде ефект

світила. Далі створюється вузол множення та перемножається колір на константу. Результат можна бачити на рисунках 3.3 та 3.4.

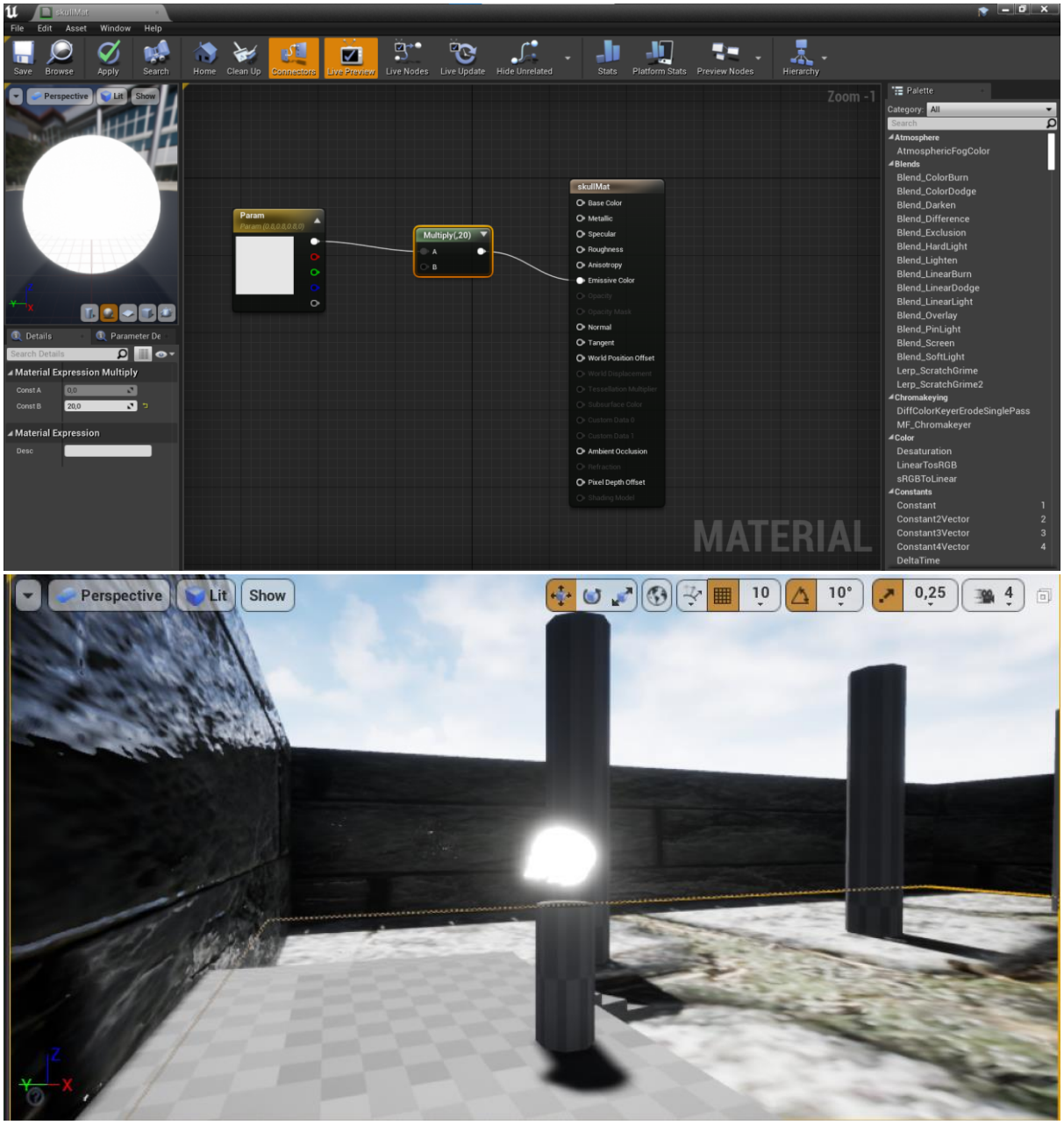



Рис 3.3 та 3.4

Наступним цікавим матеріалом буде матеріал вогню. Він буде наноситися на пласку поверхню, при цьому весь зайвий простір поверхні буде повністю прозорим, а сам вогонь анімованим. Результат можна побачити на рисунку 3.5.

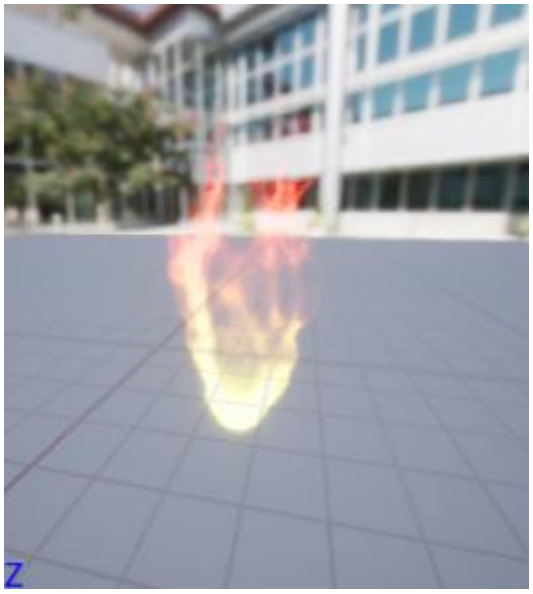


Рис. 3.5

Для цього знадобляться два зображення: спрайти вогню з рисунка 3.6 (використані були сгенеровані на сайті [12]) та звичайний градієнт, що можна скачати з інтернету або зробити власноруч у графічному редакторі.

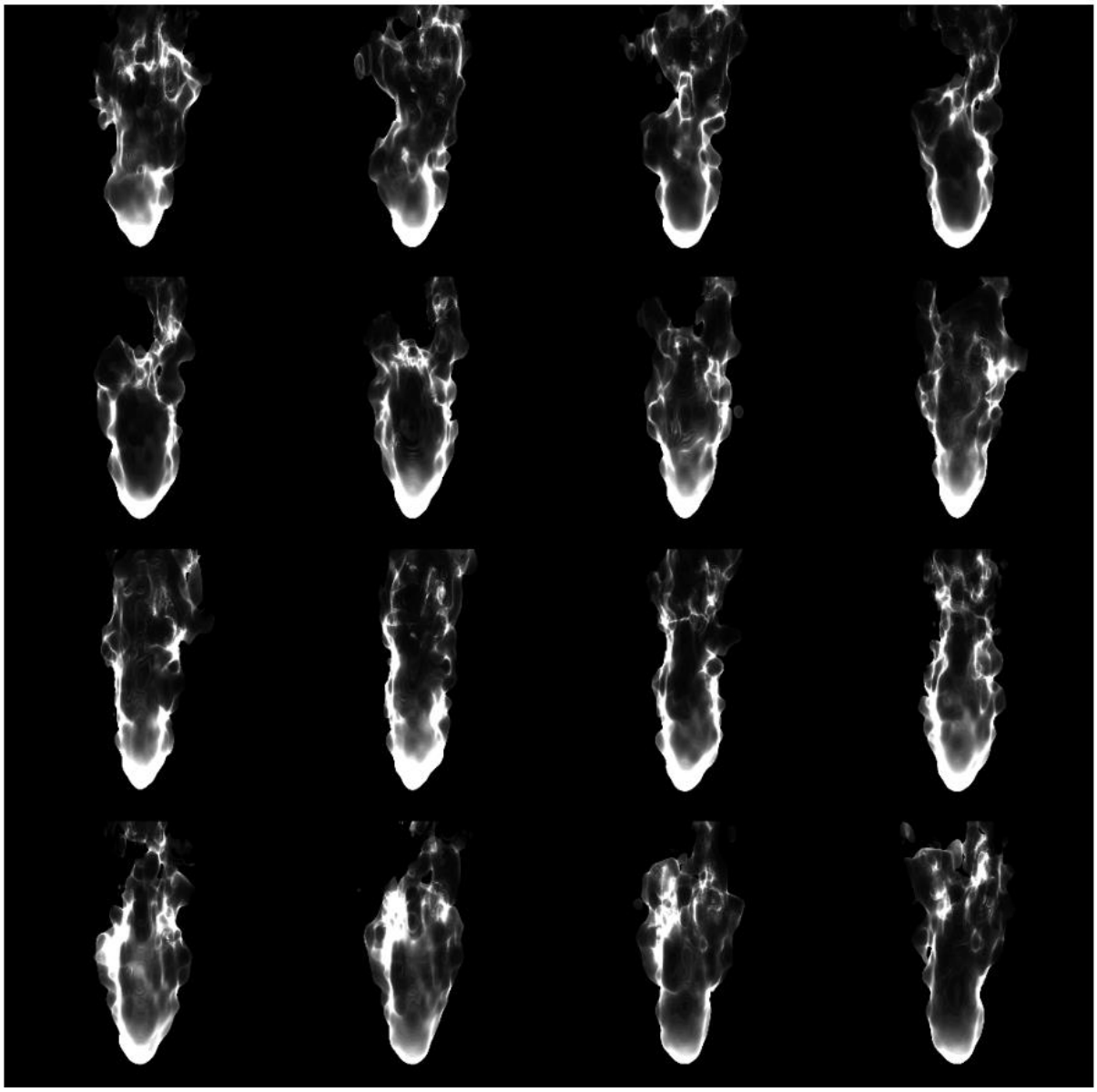


рис 3.6

Далі за допомогою події Time та функції FlipBook будуємо наступну схему з малюнка 3.7.

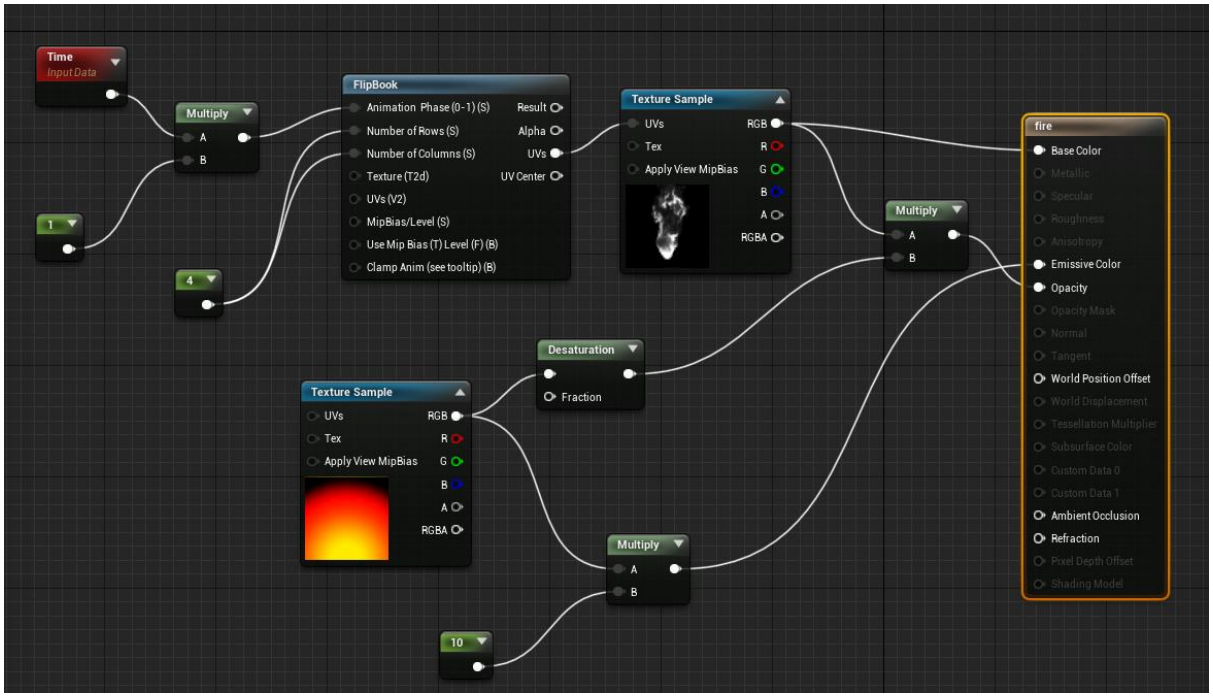



рис 3.7

І таким чином виходить вогонь, проте поки що він майже нічого не освічує, можна бачити лише слабкі відблиски на інших матеріалах, що мають у собі функцію бліків. Також цей матеріал застосовується на площинах, тому і сам він плаский, з однієї сторони його видно, але зайшовши на другу - ні. Проте це лише матеріал, його недоліки будуть виправлені у наступному пункті.

## 2.4 Робота з освітленням

По задумці на локації повинна бути стеля, тому освітлення, яке було додано при створенні проекту за шаблоном ми приберемо. В якості світил будуть використовуватися невеликі вогнища, розташовані на колонах, тому освітлення буде тьмяним, приближеним до справжнього вогню.

Спочатку створимо blueprint, додамо до нього об’єкт static mash зробивши його пласкої форми та в якості матеріалу обравши створенний нами раніше. Далі додамо джерело світла point light обравши йому колір схожий на вогонь.

Отже, тепер вогонь освітлює територію навколо, проте він все ще плаский. Ця проблема вирішується наступним чином: вогонь так і залишиться пласким, проте він буде повертатися в залежності від вектора камери гравця.

Для цього створимо нову функцію під назвою FireRotation. На рисунку 4.1 зображено реалізацію цієї функції.

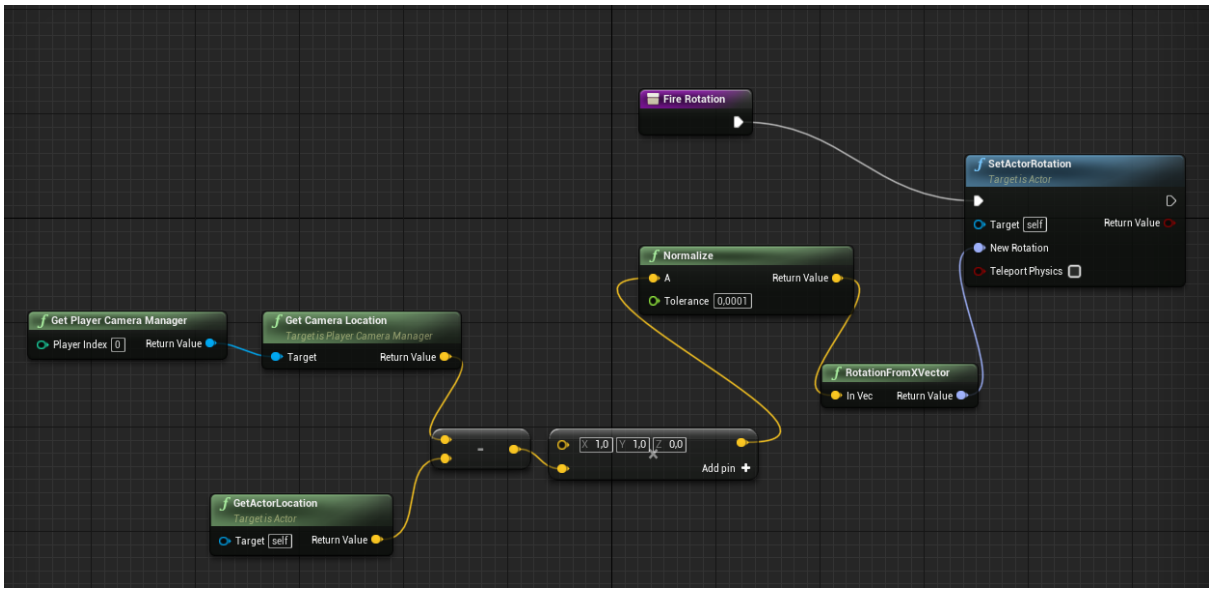



рис 4.1

Завдяки функції Get Camera Location отримуються дані про координати камери. Після деяких маніпуляцій з координатами функція SetActorRotation повертає поверхню лицем до камери гравця. І останнє, що залишилося перед розташуванням вогню на сцені, це реалізувати виклик створеної функції. Для цього просто зайдемо в схему blueprint та з'єднаємо подію та функцію. Оберемо подію Event BeginPlay, вона з самого початку гри буде викликати функцію в реальному часі, тому і вогонь увесь час буде дивитися на гравця. Див рис 4.2.

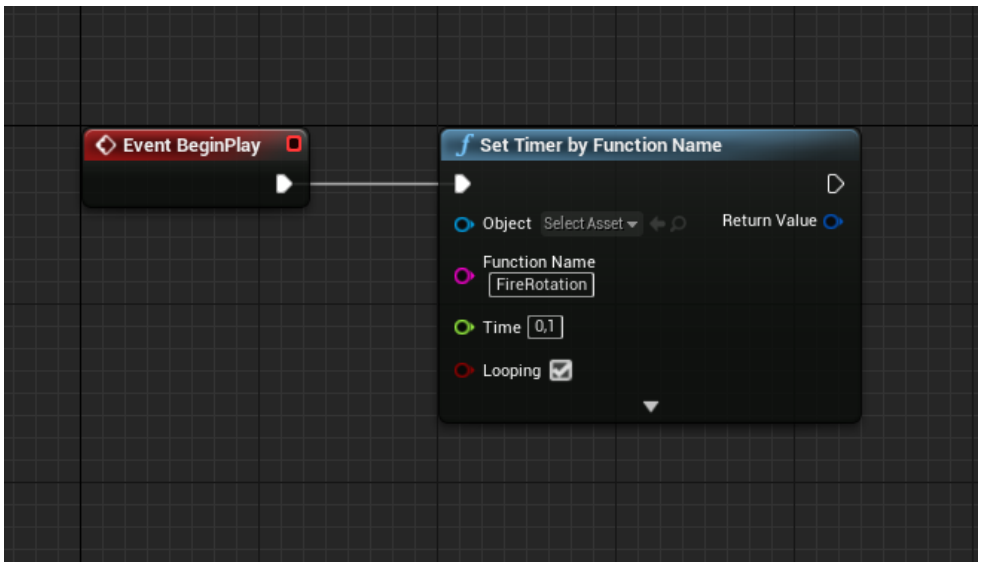



Рис. 4.2

На рисунках 4.3 та 4.4 можна бачити результат роботи.

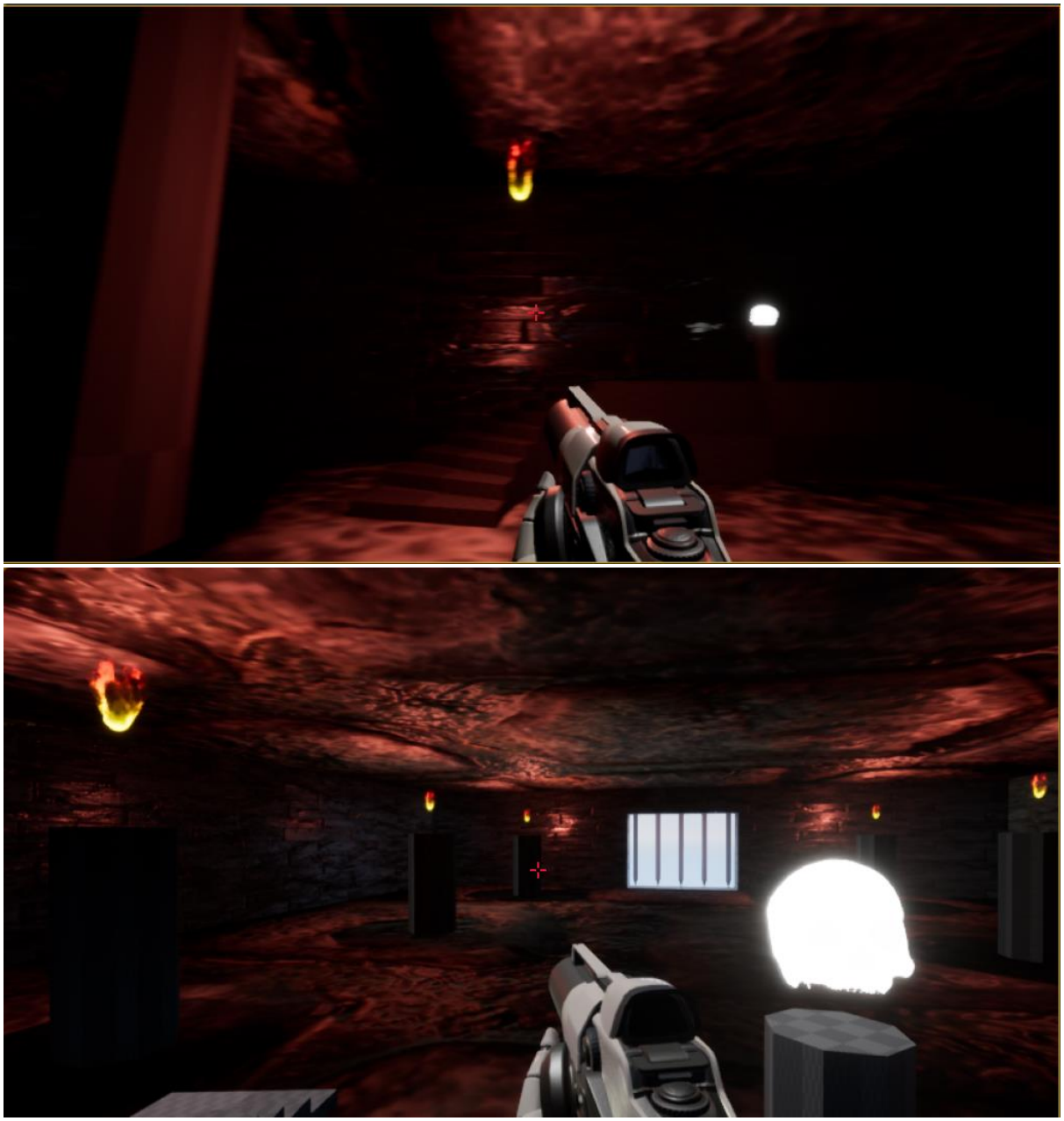


Рис 4.3 та 4.4

## 2.5 Робота з анімацією

Далі йде робота з анімацією та створення інтерактивного об’єкта. Буде зроблена анімація відкриття воріт, що запуститься після натискання на череп відповідною кнопкою.

Перед тим, як будувати ворота, спочатку налаштуємо взаємодію з черепом. Взаємодія з інтерактивним об’єктом буде проходити як і в більшості ігор: гравець наводить камеру на об’єкт, при цьому будучи неподалеку від нього,

та тисне відповідну кнопку(зазвичай E, як і в даному випадку). Для реалізації буде використана технологія RayTracing, тобто трасування променів.

Для початку потрібно створити новий об’єктний канал колізії, щоб трасування променів взаємодіяло не з усіма об’єктами, а лише з тими в котрих тип колізії буде таким, що треба. Тому заходимо в налаштування проекту, і у вкладці Collision створюємо новий object channel, як на рисунку 5.1.

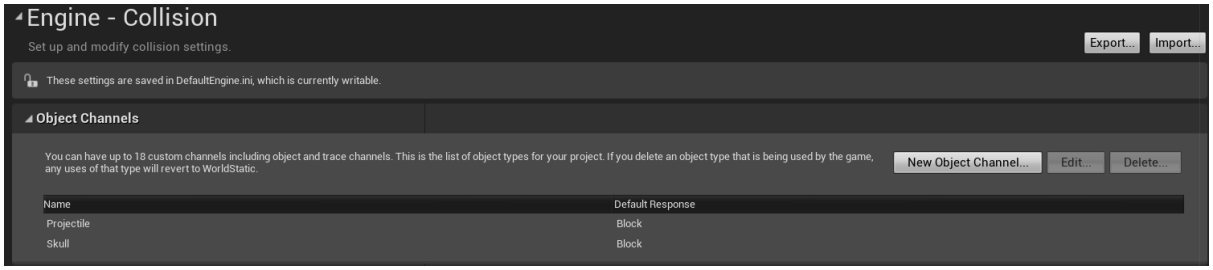



рис. 5.1

Одразу обираємо blueprint черепа, розташований на сцені, та присвоюємо йому нову колізію. Для цього в Collision Presets ставиться custom, а в object type створений варіант як на рисунку 5.2.

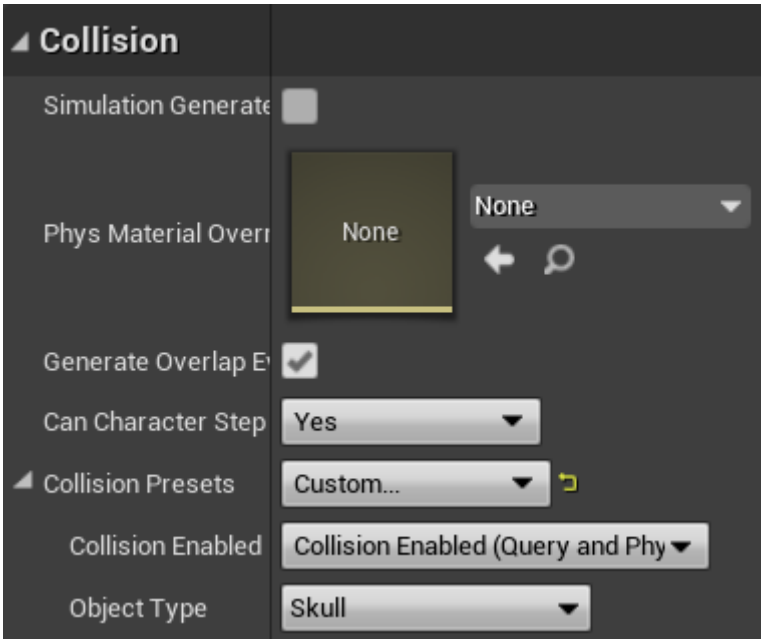



рис 5.2

Далі, так як трасування променів буде йти в напрямку погляду гравця, переходимо в blueprint ігрового персонажа, за замовчуванням FirstPersonCamera, та додаємо до нього наступну схему з рисунка 5.3.

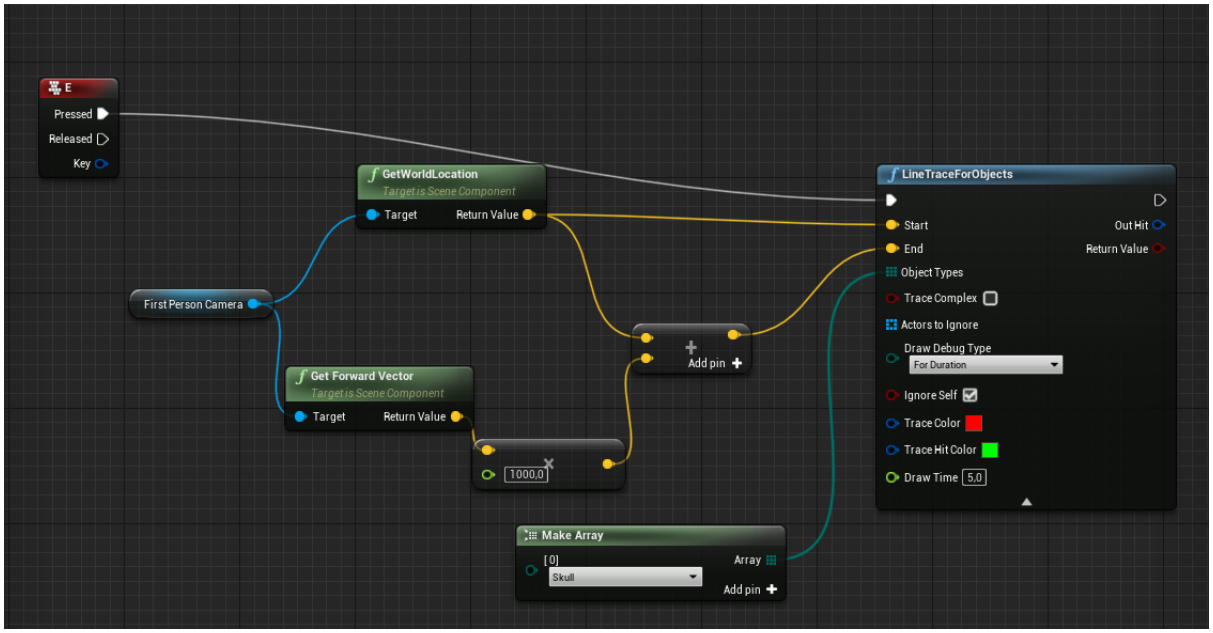



рис 5.3

В вузлі LineTraceForObjects у полі Draw Debug Type змінюється значення за замовчуванням на for duration. Це потрібно, щоб протестувати, чи правильно було налаштовано трасування, навівшись на об’єкт і натиснувши кнопку E будуть з’являтися червоні промені. Протестувавши, значення змінюється на те, що було.

Далі створюємо blueprint самих воріт за допомогою геометрії, рис. 5.4. Треба присвоїти їм колізію типу BlockAll, бо за замовчуванням будуючи геометрію в blueprint фігури блокують лише знаряди, а не самого гравця. Також потрібно додати сплайн, невидиму лінію, по напрямку якої буде йти анімація.

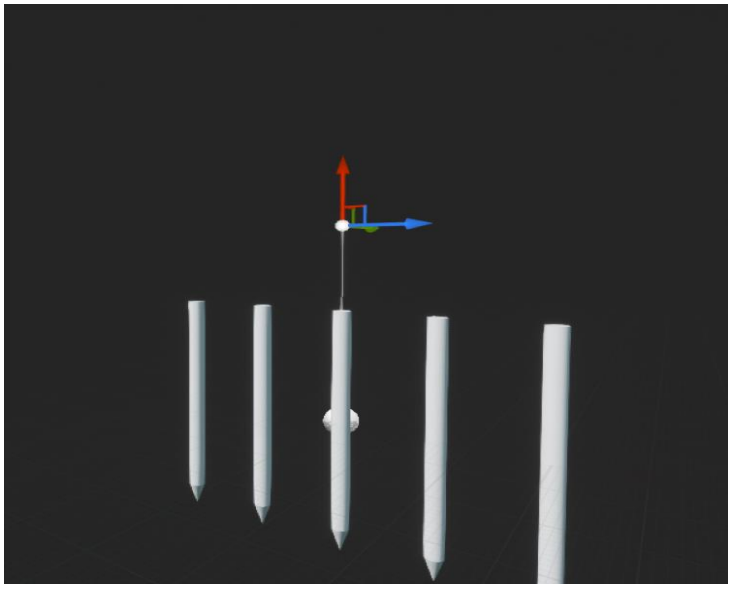


рис 5.4

Далі переходимо з вьюпорту до налаштування blueprint та створюємо вузол Timeline, рис 5.4.

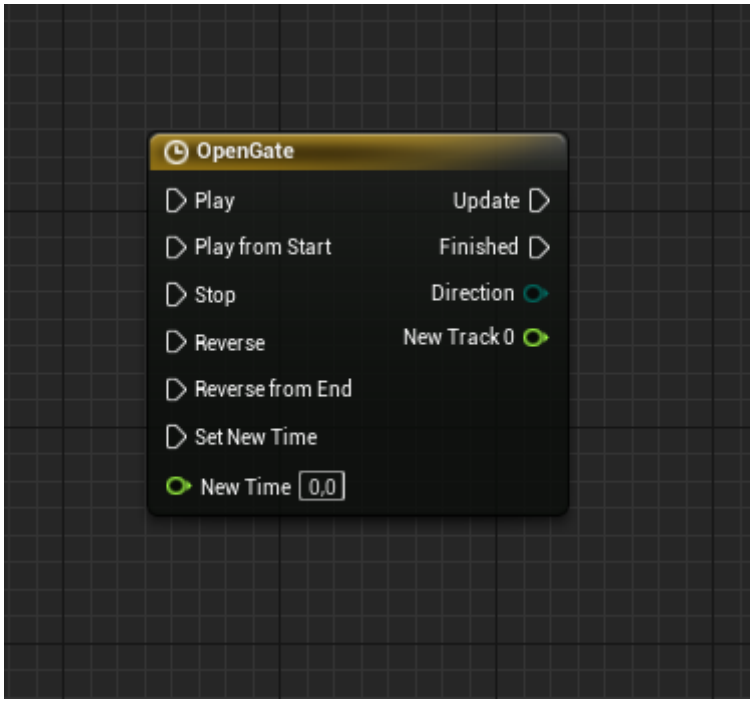



рис 5.5

Двічі тиснемо, щоб перейти до налаштувань вузла анімації, та одразу створюємо графік з двома точками, рис 5.6. Ці точки мають дві координати, x - час, y - значення, котре буде повертати функція через канал New Track 0.

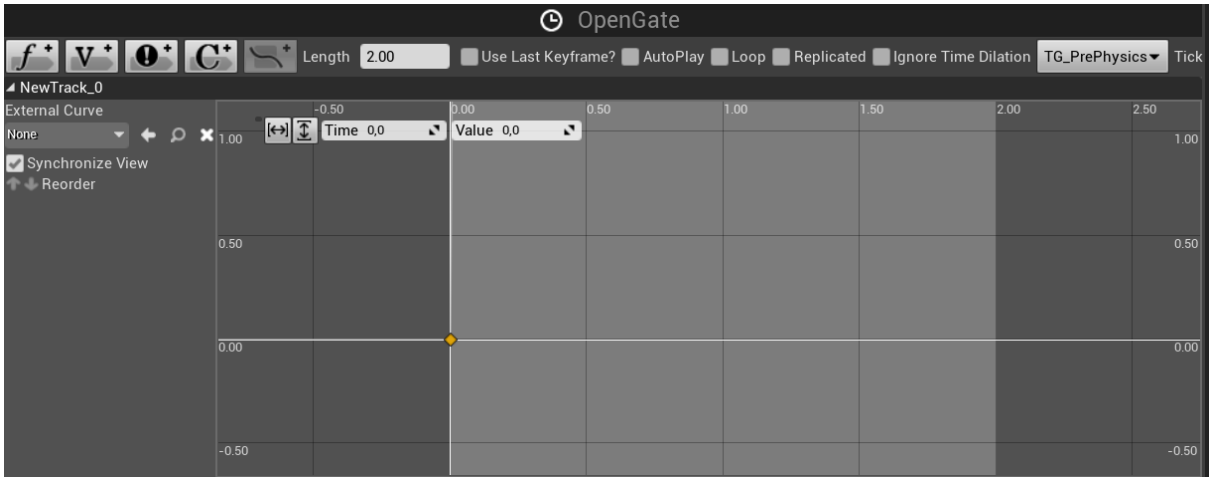



рис 5.6

Спочатку у полі Length, над графіком, ставиться значення часу, за який анімація повинна завершитися. Першій точці присвоюємо значення (0,0),

другій (2, 430) - тобто дві секунди, та координата, в котрій повинні бути ворота наприкінці анімації.

Далі додаємо ще три вузла до blueprint, рис 5.7. Змінну, котра саме і є воротами, подію та функцію SetRelativeLocation.

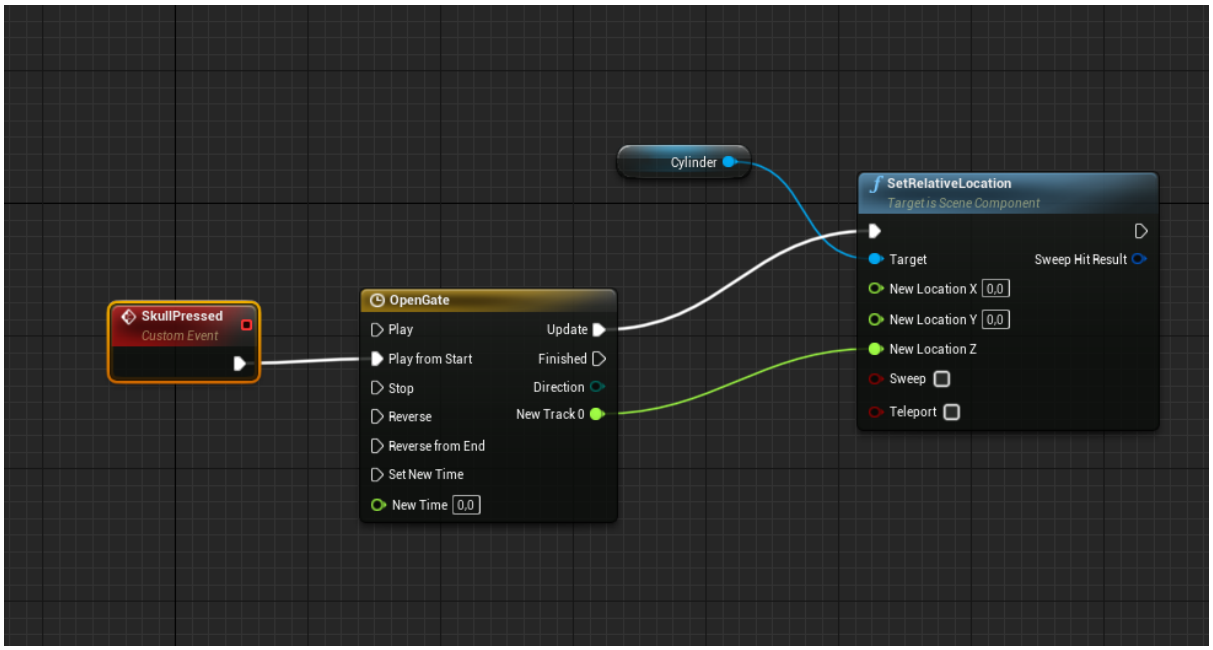



рис 5.7

При виклику події SkullPressed розпочинається таймлан OpenGate, що у кожен момент часу передає значення координати, в котрій мають знаходитися ворота, а функція SetRelativeLocation як раз і здійснює переміщення воріт.

Пересування воріт вже прописано, проте, як вже зазначено, для цього спочатку треба почати подію. Так як подія поки не викликається ні за яких умов, цим і займемося. Повернемося до blueprint камери гравця та налаштованого трасування променів. Додаємо наступну схему з рисунку 5.8. В цьому доповненні схеми беруться всі актори з blueprint воріт, та за допомогою get витягується потрібна подія SkullPressed. Далі потрібно, щоб при натисканні E ця подія викликалася. Для цього б вистачило просто провести зв’язок від каналу out hit вузла LineTraceForObject до каналу події, проте тоді воно буде викликати подію кожного разу коли промінь перетинає будь-який об’єкт з необхідним типом колізії. Щоб такого не було береться назва об’єкту, що перетинає промінь, та порівнюється з необхідною кнопкою.

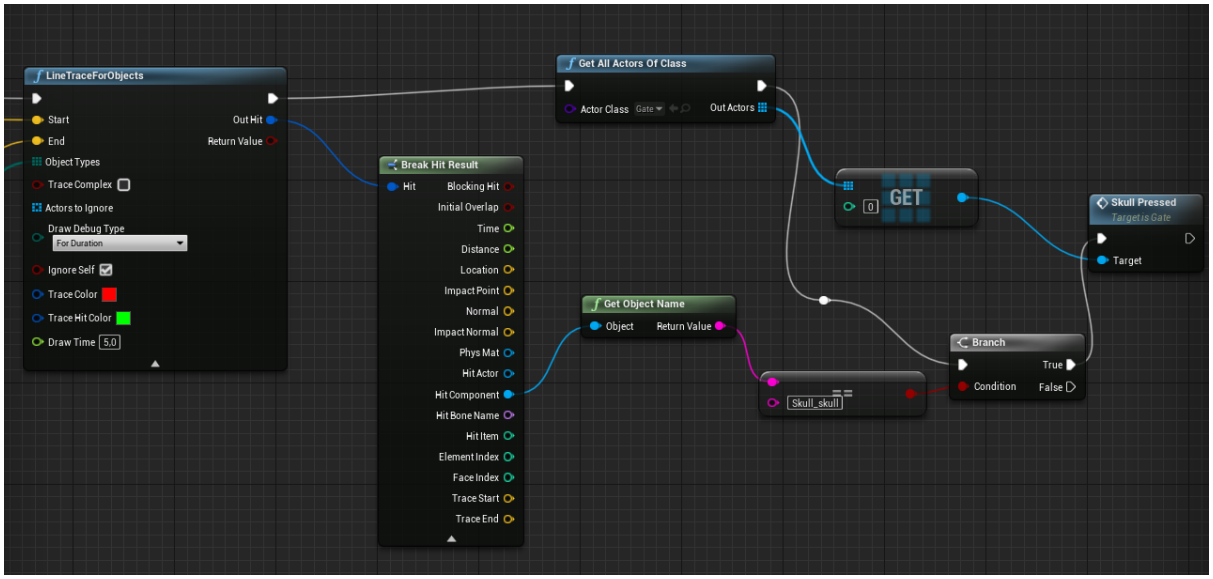



Рис 5.8

Все готово для використання. На рисунку 5.9 можна бачити результат роботи.

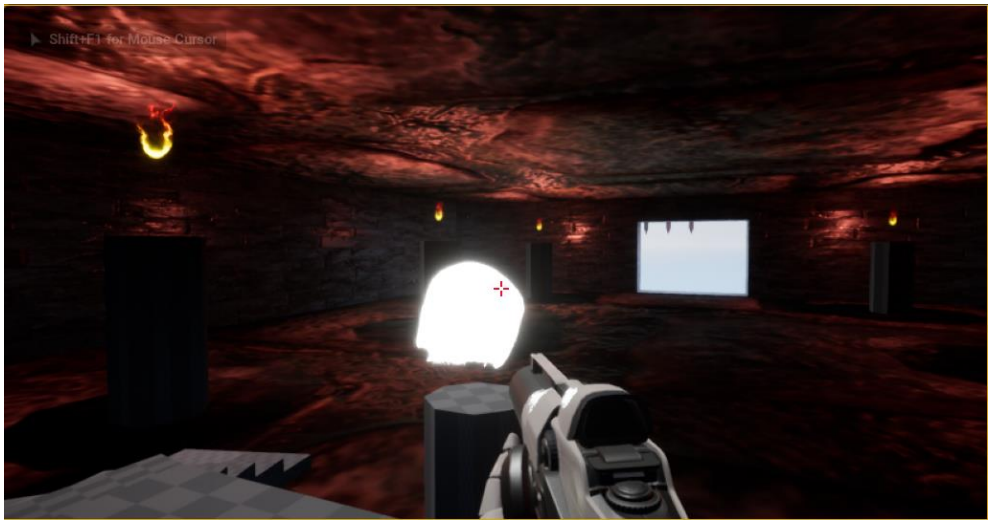


рис 5.9

## 2.6 Розширення механік пересування

Одна механіка, що була прописана у шаблоні, буде повністю змінена, а також додана ще одна. Це механіки подвійного стрибку та ривку.

Для початку, стара механіка стрибку, зображена на рисунку 6.1, видаляється. Залишається лише подія натискання кнопки, а також додається кастомна подія MyJump, рисунок 6.2.

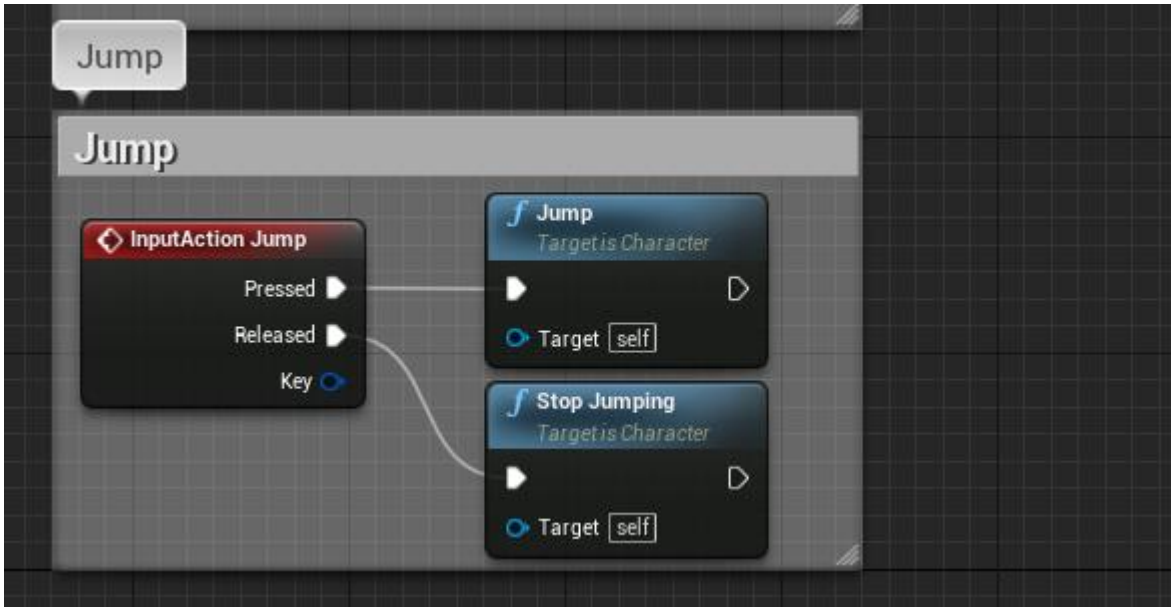



рис 6.1

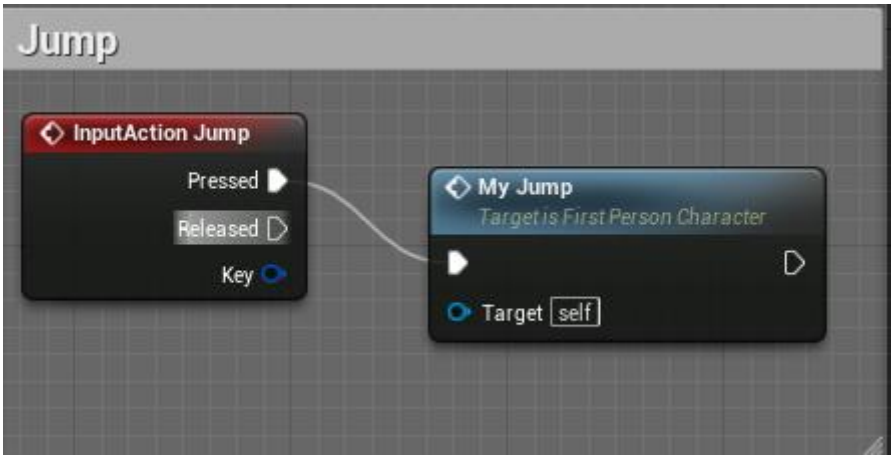



рис. 6.2

Далі до blueprint персонажа додаються змінні JumpPower - висота типу float, на яку буде здійснюватися стрибок, JumpCount - лічильник кількості стрибків типу int, необхідний для того, щоб стрибків було не більше двох, за замовчуванням 0, MaxJumps - максимальна кількість стрибків типу int,

має значення за замовчуванням 2. Отже маємо схему реалізації події MyJump на рисунку 6.3.

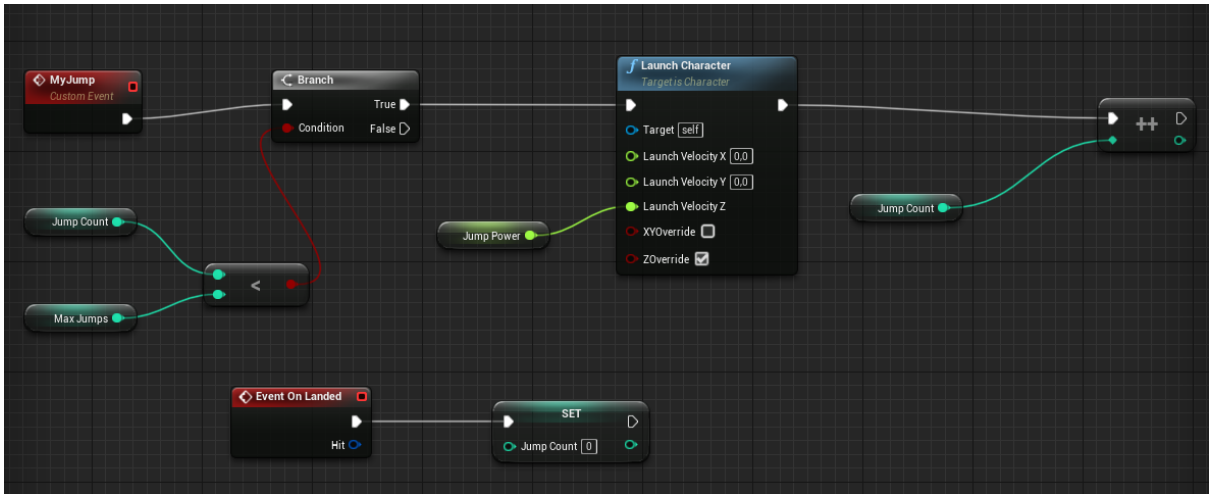



рис. 6.3

Після натискання кнопки йде перевірка чи не перевищує кількість зроблених стрибків максимальну, після першого натискання персонаж підстрибує один раз, JumpCount збільшується на один. Поки персонаж ще знаходиться у повітрі можна стрибнути ще раз, якщо кількість зроблених стрибків уже не дорівнює двом. Після приземлення подія On Landed скидає кількість зроблених стрибків до 0.

Також для придання реалізму додається ефект легкого трясіння камерою при приземленні за допомогою функції Play World Camera Shake, як показано на рисунку 6.4.

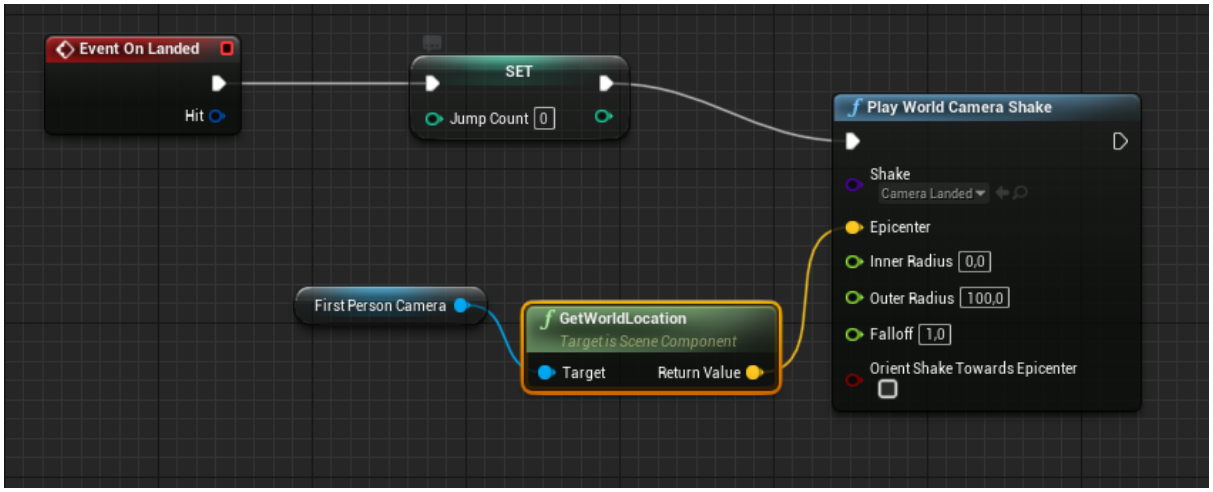



рис. 6.4

Наступна механіка - ривок. Щоб не визначати значення кнопки у blueprint, прив’яжемо її значення в налаштуваннях проекту. Для цього перейдемо у

вкладку input та в Action Mappings створимо подію Dash назначивши за нею клавішу Left Shift, рисунок 6.5.

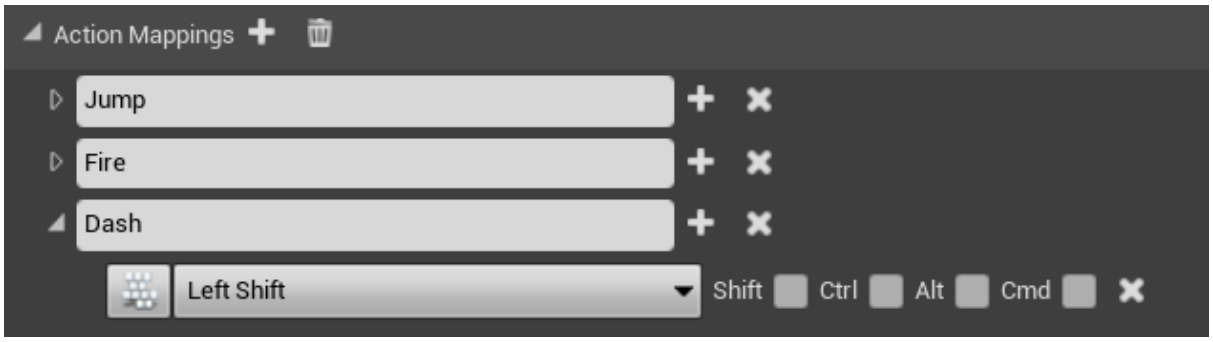



рис 6.5

Створюється змінна Dash Power типу float, відповідаюча за дальність ривку. Витягується вузол події InputAction Dash, котра спрацьовує при натисканні на лівий shift, також створюється кастомна подія Dash. Більш наглядно на рисунку 6.6.

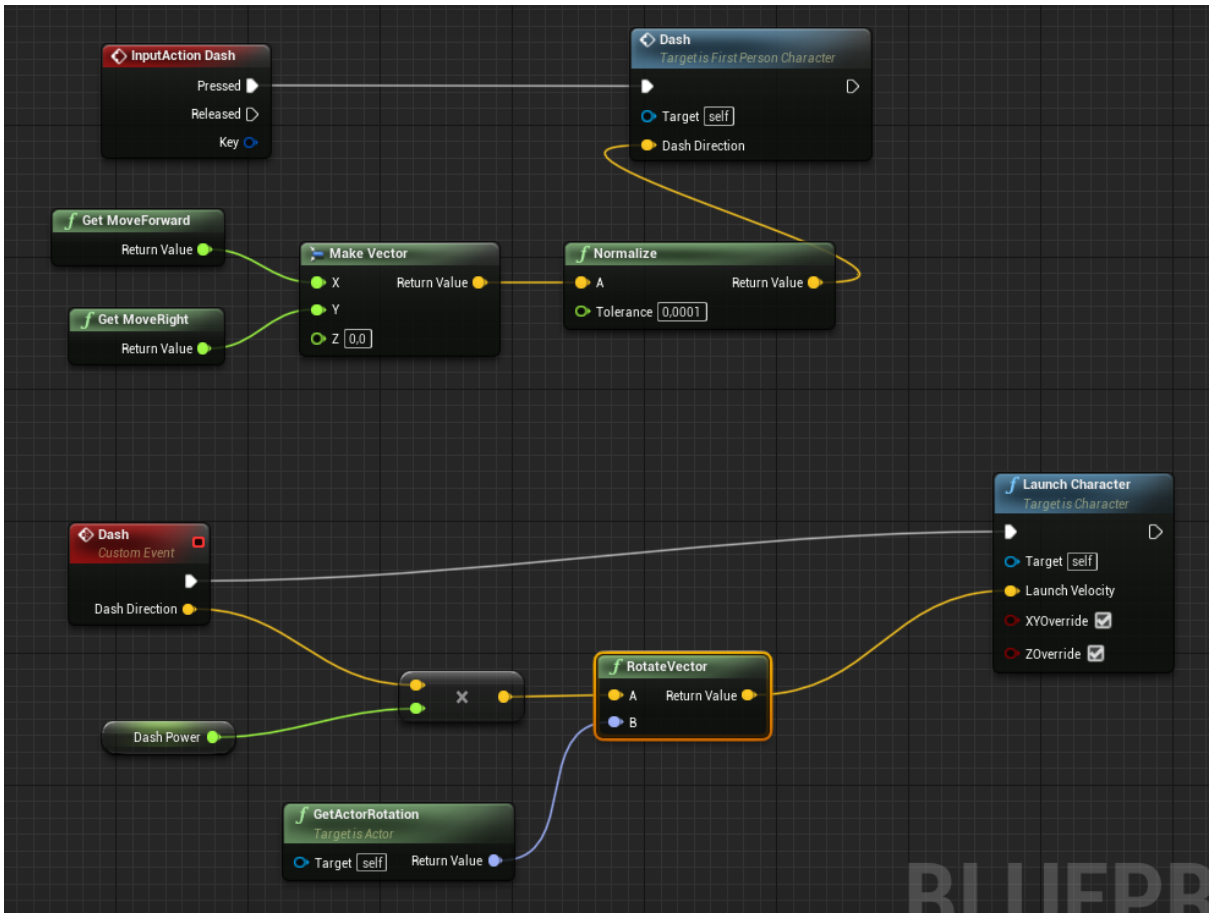



рис 6.6

Отже, натискається shift, викликається перша подія, яка бере направлений вектор, у напрямку котрого рухається персонаж, і запускає та передає його

до другої події. Друга подія, отримуючи вектор, множить його на дальність ривку та переносить персонажа у необхідному напрямку.
Проте ця механіка не буде працювати, якщо всі три координати вектора, що ми отримаємо, будуть дорівнювати нулю. Для цього, як показано на рисунку 6.7, додаємо перевірку довжини вектора. Якщо довжина дорівнює нулю, то ривок буде зроблено по напрямку погляду.

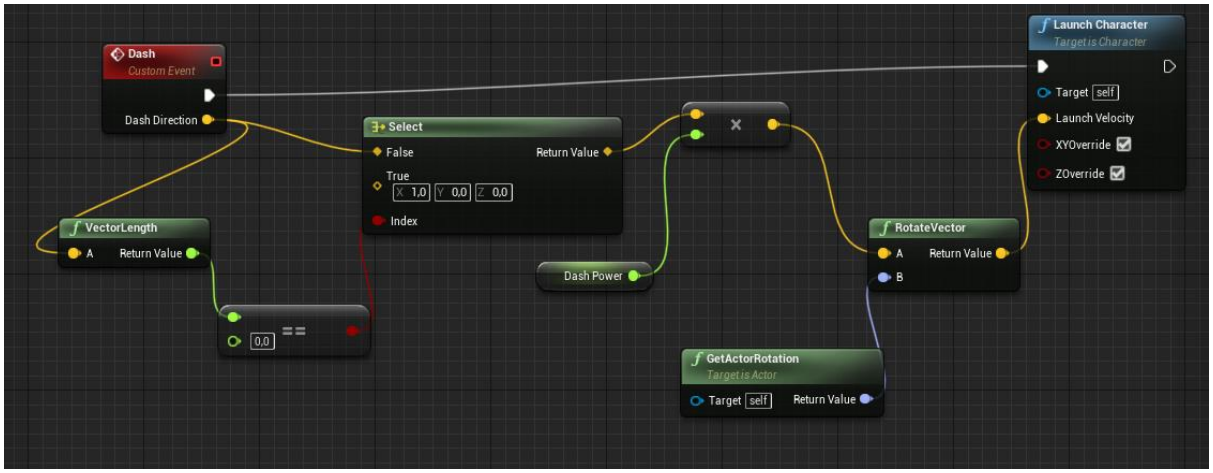
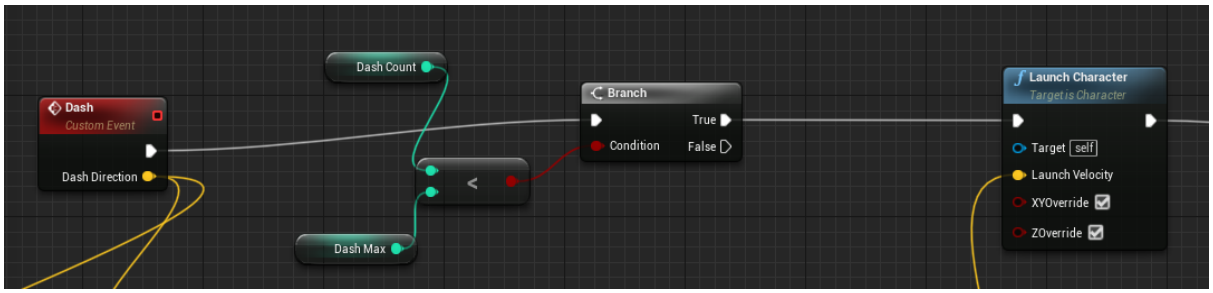



рис 6.7

Проте ще не все. При такому розкладі персонаж зможе робити ривки безліч разів. Введемо обмеження на один ривок у секунду. Для цього до схеми з минулого рисунка додамо блоки з рисунків 6.8 та 6.9.



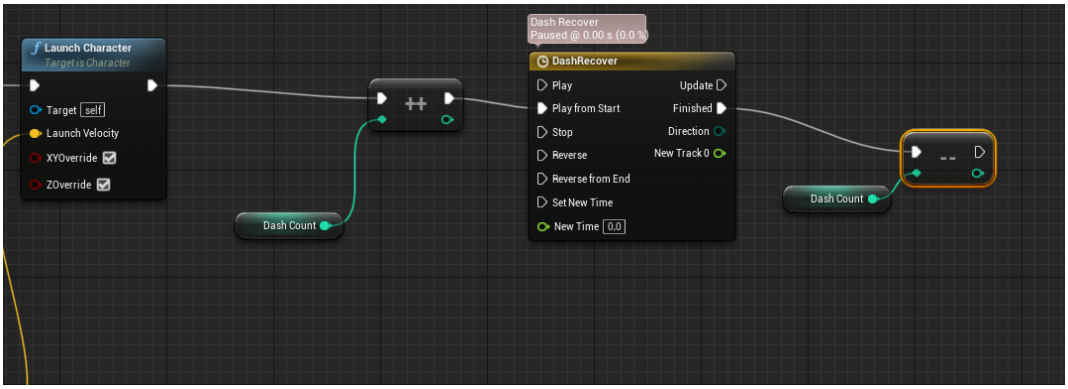




рис. 6.8 та 6.9

Вводимо дві змінні по аналогії зі стрибками і порівнюємо. Максимальна кількість ривків дорівнює одному, після виконання вузла Launch Character кількість зроблених ривків стає рівна максимальній і можливість ривків блокується. Далі під'єднуємо вузол таймлайну, що буде йти 1 секунду і після цього зменшує кількість зроблених ривків, тим самим даючи змогу зробити ще один.

## 2.7 Реалізація стрільби

При реалізації стрільби зазвичай є два варіанти. Перший, коли зброя створює кулі, так звані projectiles, які є фізичними об'єктами. Такий спосіб є задовільним, коли кулям, наприклад, треба прописати балістику, або зброя замість куль, що не помічає око, використовує, наприклад, ракети, котрі летять повільніше, проте такий спосіб має і досить немало недоліків, такі як незручність налаштувань, а також навантаження великою кількістю створюваних фізичних об'єктів. Другий спосіб вже знайомий RayTracing, що використовувався для взаємодії з черепом. Тут не створюється ніяких фізичних об'єктів, а просто отримується інформація, що за об'єкт перетнув промінь, та причиняє шкоду напряму об'єкту. В ході роботи буде реалізовано другий спосіб.

Він реалізовується майже так само, як вже прописана схема трасування променів. Спочатку додаються функції Set Timer by Event та Clear and Invalidate Timer by Handle, що відповідають за автоматичну стрільбу. Далі від нода Break Hit Result протягується зв'язок до Spawn Emitter at Location та Spawn Sound at Location, функцій, що відповідають за появу ефекту та звуку влучання кулі у об'єкт. Рисунок 7.1.

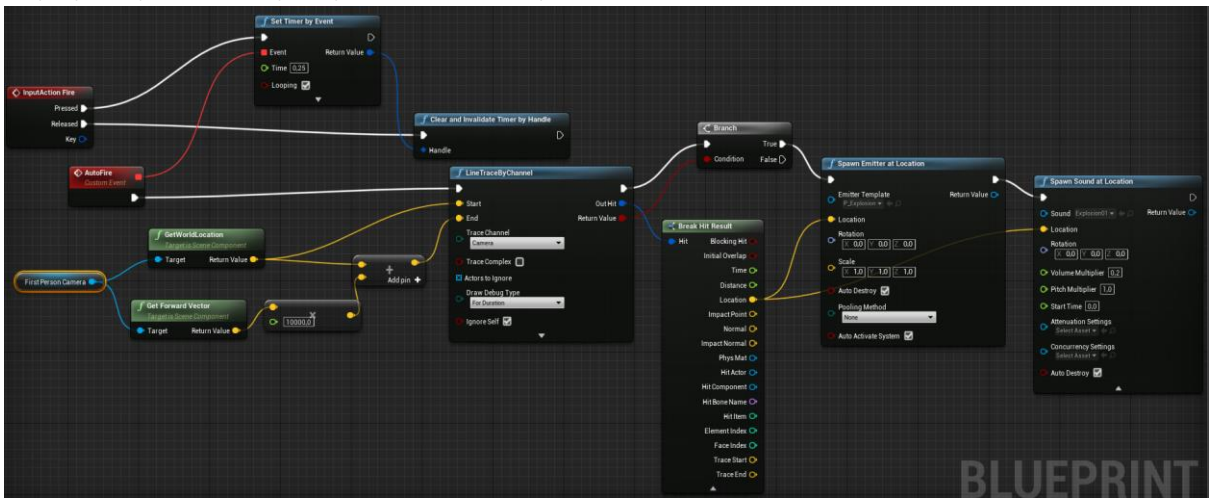



рис. 7.1

Проте поки що це лише косметичні функції, нанести шкоди зброєю все ще неможливо. Для цього додаємо функцію Apply Damage. Проте це ще також не все. Ставимо на арену об’єкт ThirdPersonCharecter, котрий буде слугувати ворогом, надалі буде зватися просто бот. Далі в blueprint бота створюється змінна Health, що буде слугувати його показником здоров’я. Надалі просто додається наступна схема з рисунку 7.2. В ній показник урону, отриманого від нашого персонажа, віднімається від показника здоров’я, а коли цей показник стає рівним або нижче нуля, бот знищується.

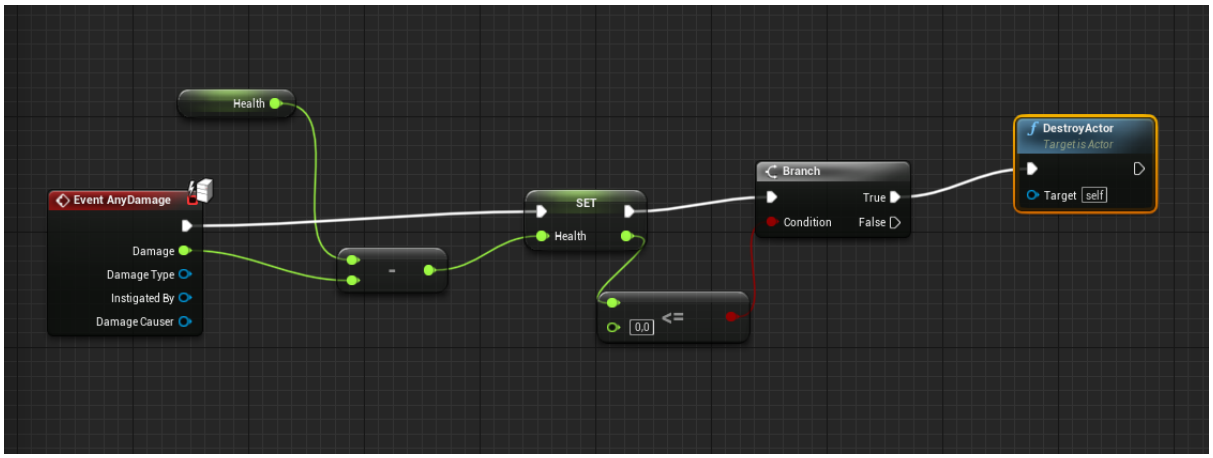



рис. 7.2

## 2.8 Налаштування ботів та їх поведінки

Будуть створені боти, що мають механіки зору, слуху, можуть переслідувати гравця, коли замітять його, та функцію самопідриву.
Для початку створимо AIController, в якому буде прописано можливості та поведінка ботів, та назвемо його Bot1. Потім треба додати створений контроллер до бота, для цього заходимо до blueprint бота та ставимо AI Controller Class на потрібний. Рис 8.1.

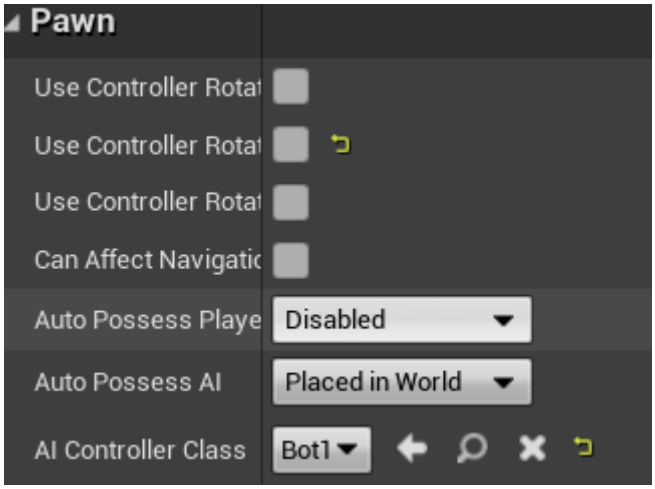



рис 8.1

Далі потрібно додати до арени NavMesh, це зона в якій контроллер прораховує куди він може дібратися та як зробити це найкоротшим шляхом. Для цього в контент браузері просто витягуємо navmesh на локацію. Рис 8.2.


рис 8.2

Далі треба додати ботам зір та слух. Для цього треба зайти в блупринт контроллера та в налаштуваннях активувати AI Sight Config та AI Hear Config та підправити параметри при необхідності, в основному це радіус зору та слуху, та кут обзора. Рис 8.3.

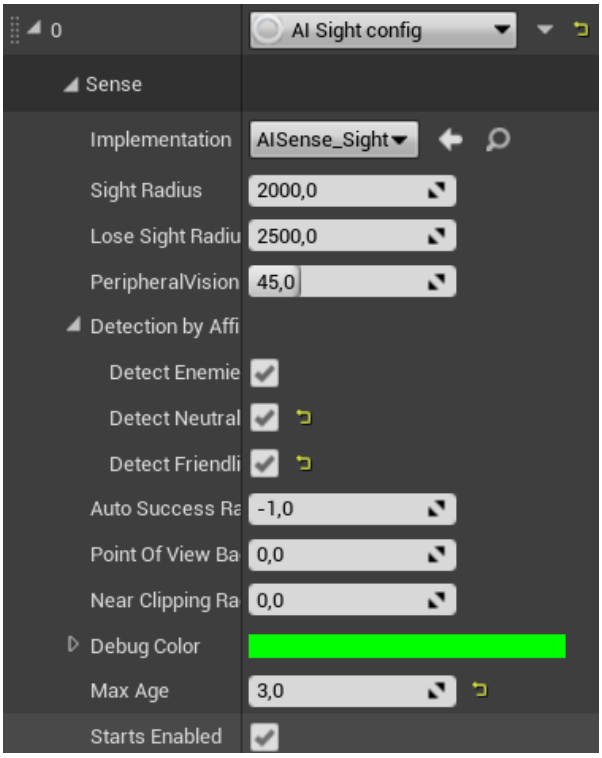



рис 8.3

Тепер боти будуть помічати ігрового персонажа, коли він зайде в їх радіус бачення. Проте вони все ще не спроможні його почути. Хоч зброя і видає звуки влучення, це не підійде для того, щоб бот почув гравця. Слух ботів не реагує на звук у прямому сенсі, він реагує лише на системний шум, так званий noise, що є просто апаратним “інструментом” спеціально для слуху у контроллера. Отже потрібно повернутися до минулого пункту, коли створювалася стрільба. Щоб бот реагував на звук пострілу треба просто додати цей самий шум у місці, де знаходиться зброя. Рисунок 8.4.

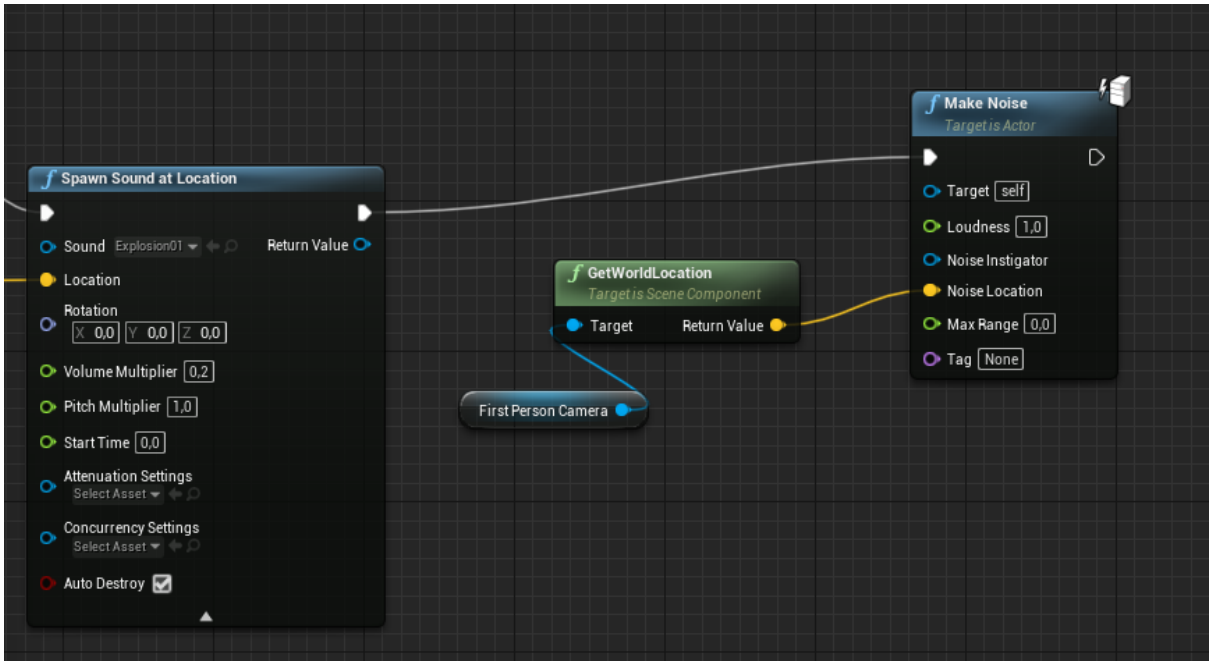


рис. 8.4

Тепер бот може не тільки побачити гравця, а й почути. Далі створимо Blackboard - об’єкт, що зберігає отримані контроллером змінні. І одразу створимо BehaviorTree, тобто дерево поведінки бота, та прив’яжемо до нього щойно створений blackboard. Далі налаштовується саме дерево поведінки. Рисунок 8.5.

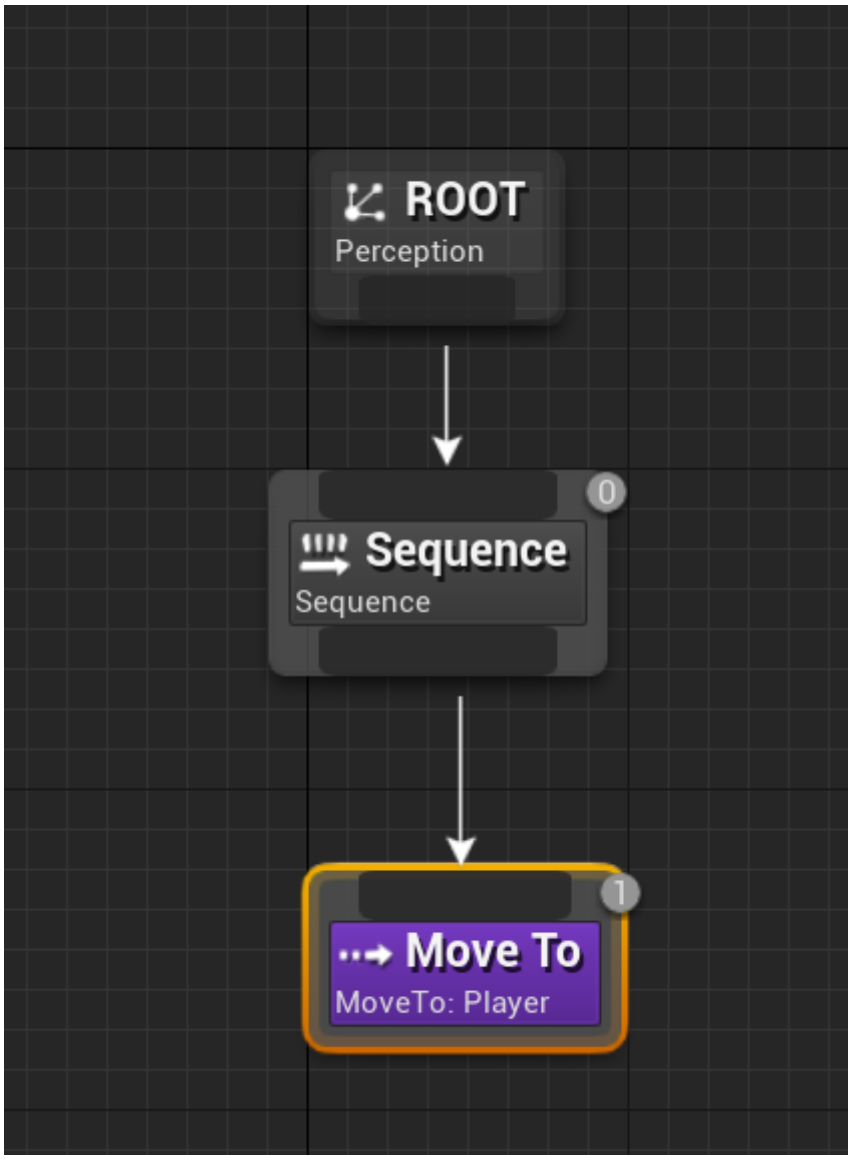



рис. 8.5

У кожного дерева поведінки завжди є корінь. Далі можуть йти лише три оператори. Зараз потрібен лише sequence. Він працює наступним чином: виконується ліва задача, якщо вона змогла виконатися до кінця, то запускається наступна, і так поки хоча б одна задача не зможе виконатися,

або задачі скінчаться або хід подій не буде зупинено спеціальною задачею. На попередньому рисунку бот, помітивши гравця, буде просто йти до нього. Треба додати схему з рисунку 8.6 до контроллеру.

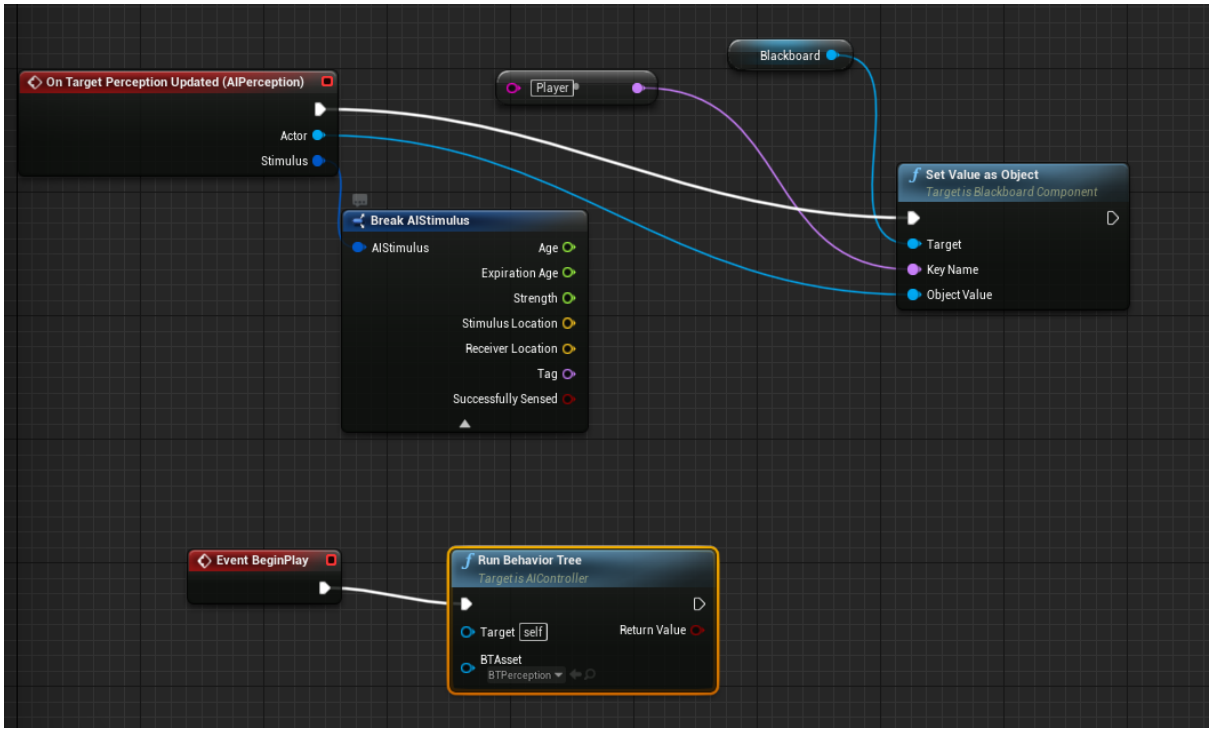



рис. 8.6

За її допомогою, бот з початком гри буде запускати дерево поведінки, а також коли побаче або почує гравця він буде зберігати дані про нього в blackboard.

Тепер бот має зір, слух, може переслідувати гравця. Залишилося прописати можливість самопідриву. Дерево поведінки набуває наступного вигляду. Рисунок 8.7.

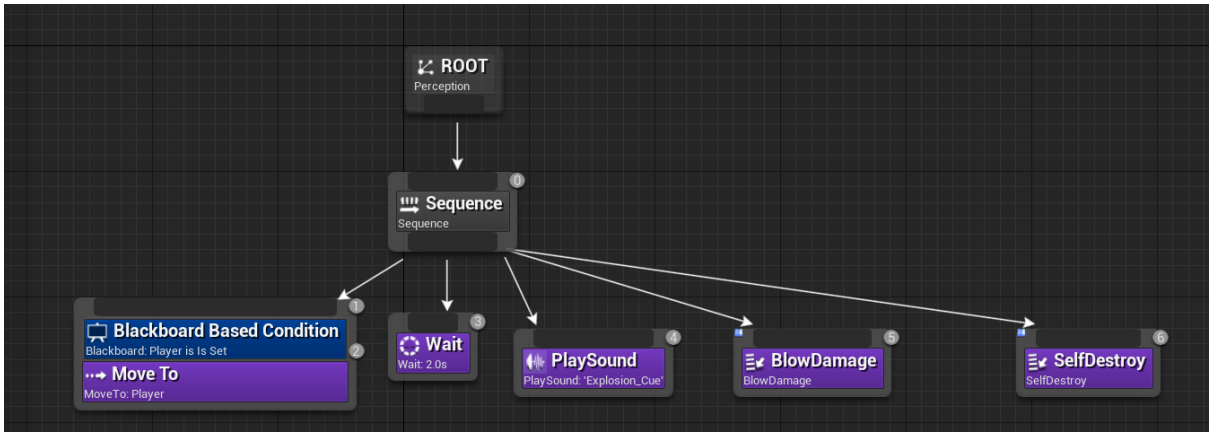



рис. 8.7

Додавання сервісу(синього прямокутника) до першої задачі дає можливість боту краще переслідувати ціль за рахунок того, що її важче втратити. Отже, коли бот помічає гравця, він біжить прямо до нього, швидкість бота спеціально виставлена більшою ніж у персонажа гравця, далі догнавши гравця починається імпровізована детонація, бот встає на одному місці на дві секунди, потім грає звук вибуху. Далі йдуть дві задачі, прописані власноруч. Перша створює візуальний ефект вибуху та викликає функцію, що завдає шкоди в радіусі навкруг бота. Рисунок 8.8.

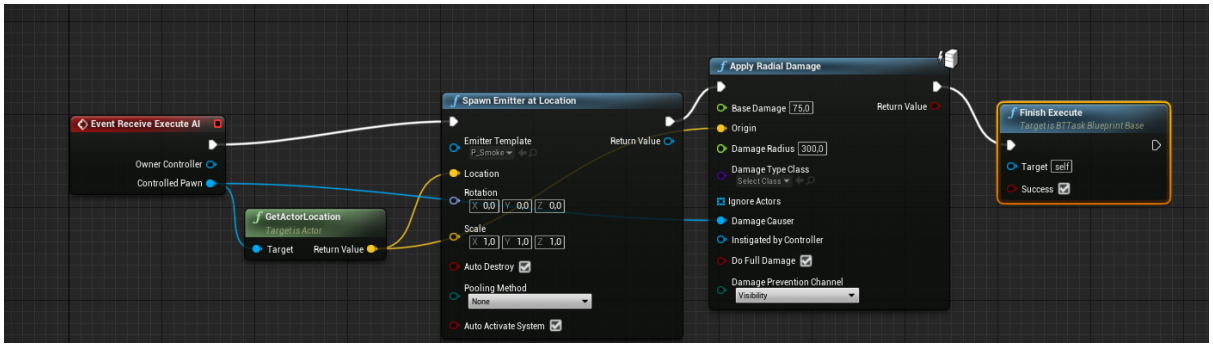



рис. 8.8

Далі йде задача, функціонал котрої просто знищення моделі бота та його контроллеру. Рисунок 8.9.

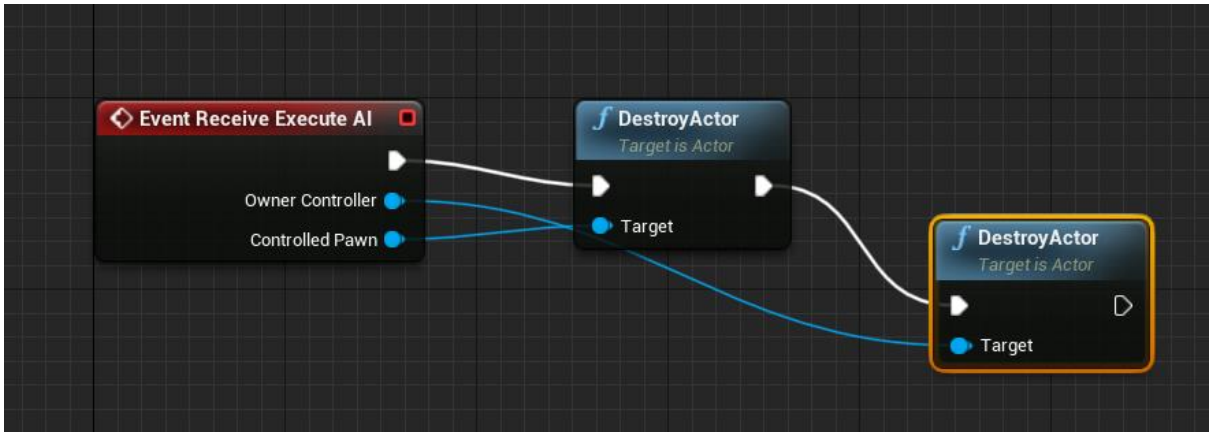



рис 8.9

## 2.9 Показник здоров'я гравця

Тепер у грі є можливість наносити урон ботам, а також боти можуть наносити урон. Проте в логіці персонажа гравця ще досі не прописано отримання урону. Для початку треба створити саму змінну здоров'я у blueprint гравця, а логіку отримання урону просто скопіювати з blueprint бота. Далі треба вивести індикатор, що показував би цей показник в самій грі. Для цього створимо blueprint віджетів і просто перетащимо в нього task

bar, прив’язавши до нього функцію, яка б брала дані про показник здоров’я гравця. Рисунки 9.1 та 9.2.

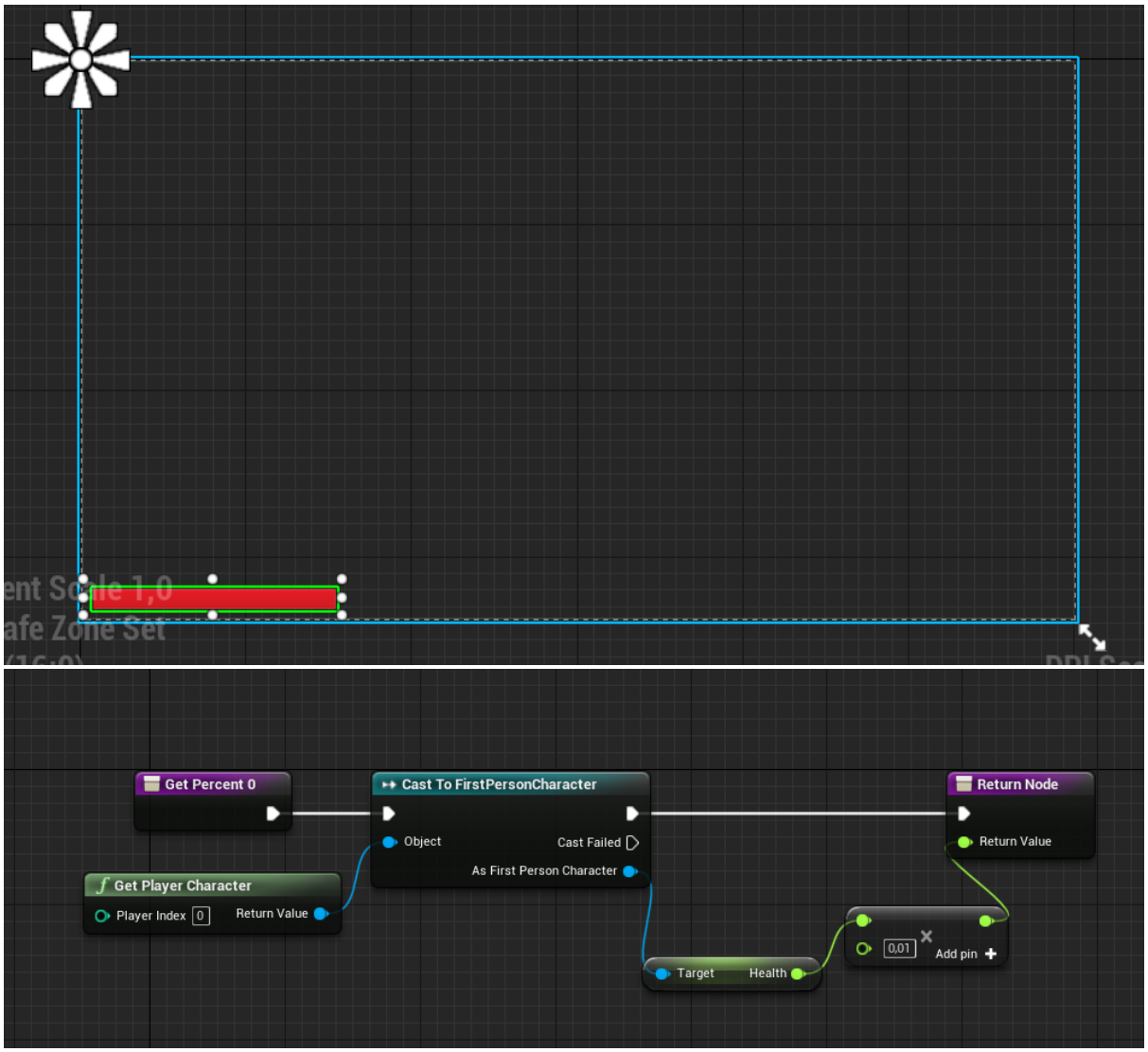



рис 9.1 та 9.2

Показник здоров’я гравця множиться на 0,01 так як task bar приймає значення від 0 до 1.

Далі до блупринту ігрового персонажу додається наступна схема з рисунку 9.3.

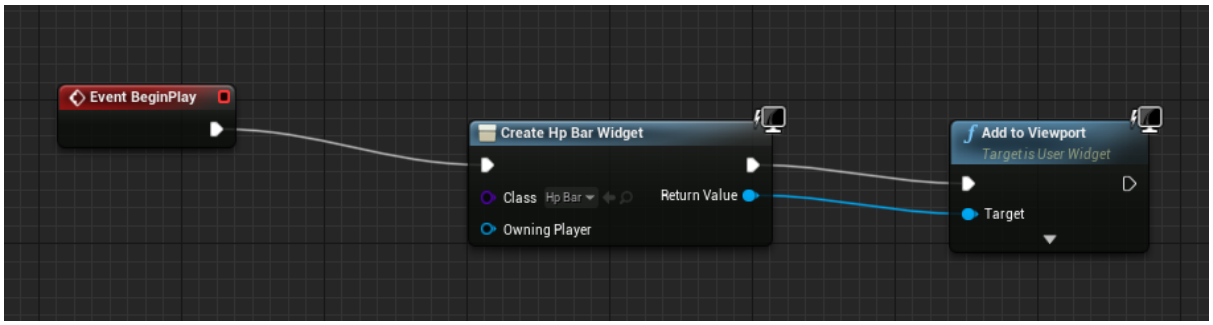


рис 9.3

Ця схема просто виводить віджет на екран після початку гри та обновлює його при зміні даних.

## 2.10 Аудіо-упровадження

Просто до функції, за допомогою якою була реалізована взаємодія з черепом, додається вузол з рисунку 10.1.

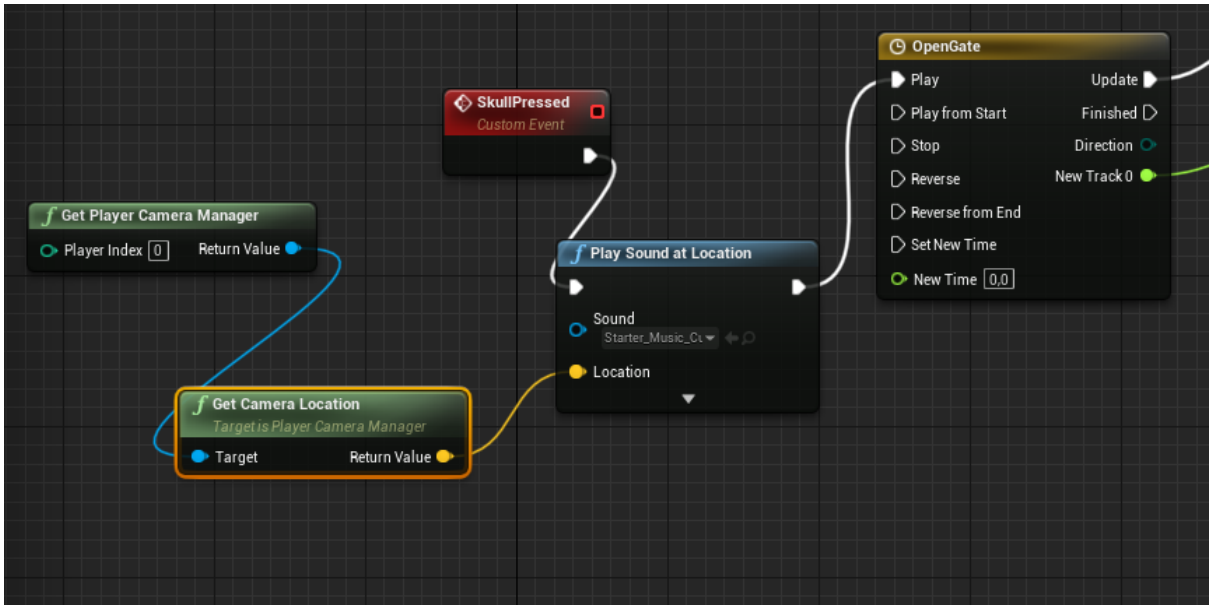



рис 10.1

Тепер при натисканні на череп, перед запуском анімації, буде починати програвати аудіозапис.

# 3 КОНТРОЛЬНИЙ ПРИКЛАД

Після запуску демоверсії можна переконатися в тому, що програмне забезпечення працює правильно. Так вогонь світиться і повертається в напрямку гравця.

При натисканні на череп запускається анімація воріт і аудіо- супровід, що свідчить про правильну взаємодію із цим об’єктом.

Можна переконатися, що механіки подвійного стрибку та ривку працюють справно: відбуваються відповідні пересування, запрограмовані обмеження не порушуються.

Правильно працює механіка стрільби: постріли летять у напрямку погляду гравця, з’являються візуальні та звукові ефекти, шкода завдається лише ворогам.

Також справно працюють боти: коректно виконується дерево поведінки, шкода гравцю завдається. При смерті боти знищуються.

# ВИСНОВКИ

За допомогою движку Unreal Engine 4 та технології візуального програмування Blueprints створено програмне забезпечення з метою побудувати арену для комп'ютерної гри. Воно має перевагу в тому, що являє собою скорочений шаблон для розробки комп'ютерних ігор і заодно це навчальний матеріал для роботи в Unreal Engine 4. Працююча демо-версія гри свідчить, що усі пункти завдання виконані.

Програмне забезпечення може бути основою для створення перспективних комп'ютерних ігор.

Подальші плани по покращенню гри:

- Створення більш якісних ассетів для придання грі більш приємного вигляду;
- Покращення поведінки ботів, додання нових типів ворогів;
- Додання різних типів зброї;
- Впровадження нових механік;
- Створення кооперативного режиму гри по інтернету.

# СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ


1. Руководство - Unity - Manual [Електронний ресурс] // https://docs.unity3d.com – Режим доступу до ресурсу: https://docs.unity3d.com/ru/530/Manual/
2. Unreal Engine 4 Documentation [Електронний ресурс] // https://docs.unrealengine.com – Режим доступу до ресурсу: https://docs.unrealengine.com/4.27/en-US/
3. Введение — Документация Godot ... - Godot Docs [Електронний ресурс] // https://docs.godotengine.org – Режим доступу до ресурсу: https://docs.godotengine.org/ru/stable/about/introduction.html
4. GameMaker Manual [Електронний ресурс] // https://manual-ru.yoyogames.com – Режим доступу до ресурсу: https://manual-ru.yoyogames.com/#t=Content.htm
5. Global Anarchy 3 Gameplay [Електронний ресурс] // https://www.youtube.com – Режим доступу до ресурсу: https://www.youtube.com/watch?v=UTeLTRPYbys
6. Список игр на движке Unreal Engine [Електронний ресурс] // https://ru.wikipedia.org – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_%D0%B8%D0%B3%D1%80_%D0%BD%D0%B0_%D0%B4%D0%B2%D0%B8%D0%B6%D0%BA%D0%B5_Unreal_Engine
7. Категория:Игры на движке Unity [Електронний ресурс] // https://ru.wikipedia.org – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/%D0%9A%D0%B0%D1%82%D0%B5%D0%B3%D0%BE%D1%80%D0%B8%D1%8F:%D0%98%D0%B3%D1%80%D1%8B_%D0%BD%D0%B0_%D0%B4%D0%B2%D0%B8%D0%B6%D0%BA%D0%B5_Unity
8. Официальный сайт Escape from Tarkov [Електронний ресурс] // https://www.escapefromtarkov.com – Режим доступу до ресурсу: https://www.escapefromtarkov.com/?lang=ru

9. Atomic Heart on Steam [Електронний ресурс] // https://store.steampowered.com – Режим доступу до ресурсу: https://store.steampowered.com/app/668580/Atomic_Heart/
10. Туториал по Unreal Engine. Часть 1: знакомство с движком [Електронний ресурс] // https://habr.com/ – Режим доступу до ресурсу: https://habr.com/ru/post/344394/
11. UE4 Tutorial: Fire Effect [Електронний ресурс] // https://www.youtube.com/ – Режим доступу до ресурсу: https://www.youtube.com/watch?v=B8Gya38KkJY&t=0s
12. EffectTextureMaker [Електронний ресурс] // http://mebiusbox.github.io – Режим доступу до ресурсу: http://mebiusbox.github.io/contents/EffectTextureMaker/
13. [Глава 1] Как создать игру на Unreal Engine 4/5. Основы программирования в Blueprint [в одном уроке] [Електронний ресурс] // https://www.youtube.com/ – Режим доступу до ресурсу:https://www.youtube.com/watch?v=vrEwa5Pdqxc&list=PL2suyruNHd0hW0dsCRH9ryb8vq7pC-uXR&index=2
14. [Глава 2] Как создать игру на Unreal Engine 4/5. Основы программирования в Blueprint [в одном уроке] [Електронний ресурс] // https://www.youtube.com/ – Режим доступу до ресурсу:https://www.youtube.com/watch?v=bid81A0hor8&list=PL2suyruNHd0hW0dsCRH9ryb8vq7pC-uXR&index=3
15. Unreal Engine 4 Искусственный Интеллект [Електронний ресурс] // https://www.youtube.com/ – Режим доступу до ресурсу: https://www.youtube.com/playlist?list=PL2suyruNHd0jgUqk01YFHjvVKlrx-bcCd
16. Creating A First Person Shooter - Unreal Engine 4 Course [Електронний ресурс] // https://www.youtube.com/ – Режим доступу до ресурсу: https://www.youtube.com/playlist?list=PLL0cLF8gjBprG6487lxqSq-aEo6ZXLDLg