Хмельницький національний університет
Факультет програмування
та комп`ютерних і телекомунікаційних систем
Кафедра телекомунікацій, медійних та інтелектуальних технологій

ДИПЛОМНА РОБОТА МАГІСТРА

Розробка комп`ютерної гри в Unreal Engine
Назва теми

Галузь знань 11 - Математика та статистика

Спеціальність 113 - Прикладна математика

Шифр ДРПМ.2019/111.01.21.00

Виконала: студентка 2 курсу, група ПМм-19-1 _____ Локазюк В.Ю.
Підпис Ініціали, прізвище

Керівник _____ Медзатий Д.М.
Підпис, дата Ініціали, прізвище

До захисту допускаю:
Зав. кафедри ТМІТ _____ Підченко С.К.
Підпис, дата Ініціали, прізвище
11 12 2020 р.

Хмельницький, 2020

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет ПРОГРАМУВАННЯ ТА КОМП'ЮТЕРНИХ І ТЕЛЕКОМУНІКАЦІЙНИХ СИСТЕМ

Кафедра ТЕЛЕКОМУНІКАЦІЙ, МЕДІЙНИХ ТА ІНТЕЛЕКТУАЛЬНИХ ТЕХНОЛОГІЙ

Освітній рівень МАГІСТР

Галузь знань 11 МАТЕМАТИКА ТА СТАТИСТИКА

Спеціальність 113 ПРИКЛАДНА МАТЕМАТИКА

Освітня програма ОСВІТНЬО-ПРОФЕСІЙНА ПРОГРАМА ПІДГОТОВКИ МАГІСТРА

ЗАТВЕРДЖУЮ

Зав. кафедри д.т.н. доцент Підченко

Сергій Костянтинович

" 01 " 09 2020 р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ)

Локазюка Владислава Юрійовича

Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Розробка комп'ютерної гри в Unreal Engine

Керівник проекту (роботи) Медзатий Дмитро Миколайович

Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

к.т.н., доцент, Кандидат технічних наук

Затверджена наказом ректора університету від 01.09.2020 р. № 118

2. Строк подання студентом проекту (роботи) на кафедру 01.12.2020 р.

3. Вихідні дані до проекту (роботи) Проведення порівняльного аналізу підходів до вирішення та реалізації розробки комп'ютерної гри. Створити систему відкритого світу та удосконалити рпг-систему на базі Unreal Engine 4. Розробити комп'ютерну гру.

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити) Аналіз предметної області та постановка задач в розробці комп'ютерної гри. Дослідження різних ігровиї рушіїв та варіантів розробки комп'ютерної гри. Розробка системи відкритого світу та удосконалення рпг-системи.Розробка комп'ютерної гри.

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень) Аналіз результатів порівняння ігрових рушіїв. Архітектура комп'терної гри. Реалізація системи відкритого світу так удосконаленої рпг-системи.Реалізація комп'ютерної гри.

6. Консультанти розділів дипломного проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «_1_»_____вересня_____2020р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітка
1	Вибір напряму дослідження та узгодження тематики ДРМ з керівником	1.09.2020	
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	3.09.2020	
3	Робота над розділом 1 – аналіз відомих моделей, методів за темою; постановка задачі	7.09.2020	
4	Робота над розділом 2 – розробка моделей і методів для вирішення поставленої задачі	14.09.2020	
5	Робота над науковою статтею	21.09.2020	
6	Робота над розділом 3 – розробка алгоритмів та технологій, їх аналіз	5.10.2020	
7	Робота над розділом 4 – проектування ПЗ для вирішення поставленої задачі	28.10.2020	
8	Узгодження отриманих; оформлення пояснювальної записки згідно вимог	8.11.2020	
10	Попередній захист ДРМ	10.11.2020	
11	Захист ДРМ на засіданні ЕК	12.12.2020	

Студент _____ Підпис ____Локазюк В.Ю.____ Ініціали, прізвище

Керівник проекту (роботи) _____ Підпис ____Медзатий Д.М.____ Ініціали, прізвище

АНОТАЦІЯ

Тема дипломної роботи:«Розробка комп'ютерної гри в Unreal Engine».

Автор проекту: Локазюк Владислав Юрійович

Керівник проекту: Медзатий Дмитро Миколайович

Загальний обсяг роботи: 124 с., 97 рис., 4 таб., 21 джерела.

Об'єктом дослідження: є розробка комп'ютерної гри в Unreal Engine 4.

Предметом дослідження: є метод розробки рпг-системи і системи відкритого світу.

Метою роботи є створення системи відкритого світу а також удосконалення рпг-система для гри на базі Unreal Engine 4.

У дипломномній роботі досліджена предметна область, спроектована архітектура веб-додатку із урахуванням основних принципів об'єктно-орієнтованого програмування та з використанням шаблонів проектування і в результаті розробка програмної реалізації даної системи із використанням виначеної архітектури.

Для програмної реалізації данної дипломної роботи використовувався ігровий рушій Unreal Engine.

Під час розробки комп`ютерної гри було пройдено увесь життєвий цикл ПЗ. В результаті чого була реалізована Гра в жанрі ММОРПГ, яка служить для розваги та урізномантітнення жанру RPG.

ANNOTATION

Theme of the diploma project:«Development of a computer game in Unreal Engine».

The author of the project: Lokazyuk Vladislav Yurievich

Project Manager: Dmitry Medzatiy Mykolayovych

Total volume of work: 124 pages, 97 figures, 4 tables, 21 sources.

The object of research: is the development of a computer game in Unreal Engine 4.

The subject of research: is a method of developing an RPG system and an open world system.

The aim of the work is to create an open world system and improve the RPG system for playing on the basis of Unreal Engine 4.

In the thesis the subject area is studied, the architecture of the web application is designed taking into account the basic principles of object-oriented programming and using design templates and as a result the development of software implementation of this system using the developed architecture.

For the software implementation of this thesis was used game engine Unreal Engine.

During the development of the computer game, the entire software life cycle was completed. As a result, the Game in the MMORPG genre was implemented, which serves to entertain and diversify the RPG genre.

11.12.2020
Дата/Date

Підпис студента/Signature

ЗМІСТ

ВСТУП

Разом з появою комп'ютерів з'явилися комп'ютерні ігри, які відразу знайшли масу шанувальників. Комп'ютерний гравець адаптується до переміщень з одного віртуального світу в інший, швидко сприймаючи незнайомі ситуації. Варто відзначити, що в комп'ютерних іграх моделюються найрізноманітніші ситуації: від повсякденних прогулянок з собакою, до фантастичних космічних війн. У бурхливо мінливому суспільстві XXI століття розвинена інтелектуальна гнучкість сприяє пристосуванню до нових, несподіваних ситуацій.

Розвиток і вдосконалення ігор тісно взаємопов'язано з розвитком комп'ютерних технологій. На даний момент безліч комплектуючих комп'ютерів розробляється, спеціально для сучасних ігор. Наприклад, високопродуктивні відеокарти, вартість яких сягає вартості двох комп'ютерів для роботи в офісі. Всі ігри розробляються з урахуванням останніх інновацій комп'ютерної індустрії, реагуючи на всі розробки і все більше посилюючи реалістичність зображення і звукового супроводу. На сьогоднішній день існують вражаючі своєю правдоподібністю гри, з хорошим графічним і звуковим оформленням, практично повністю повторюючи реальне життя. Існує і постійно виникає величезна кількість фірм, які розробляють все нові і нові ігри.

Поточні темпи комп'ютеризації перевищують темпи розвитку всіх інших галузей. Без комп'ютерів і комп'ютерних мереж не обходиться сьогодні жодна фірма, не кажучи про великих корпораціях. Сучасна людина взаємодіє з комп'ютером постійно - на роботі, вдома, в машині і навіть в літаку. Комп'ютери стрімко входять в людське життя, займаючи своє місце в нашій свідомості, а ми часто не усвідомлюємо того, що багато в чому залежимо від працездатності цих дорогих металевих коробок.

Словосполучення «комп'ютерні ігри» міцно увійшло в наше життя, кожен, хто має комп'ютер напевно зміг відчути їх привабливість, мабуть, гра закладена в саму природу людини з найдавніших часів - вистежити здобич, заманити її в пастку - це теж свого роду гра. Але тепер ми позбавлені цього в повсякденному житті, а

інстинкти залишилися і вони знаходять вихід на просторах віртуальних світів. Ви не замислювалися, чому відсоток чоловіків серед захоплюються комп'ютерними іграми в десятки разів більше ніж жінок? Можливо, саме через те, що чоловік завжди був мисливцем-добувачем-воїном і нікуди всім нам не піти від 30000 років протистоянь з хижаками і між собою записаних в наших генах.

З удосконаленням комп'ютерних технологій удосконалювалися і гри, привертаючи до себе все більше уваги. На поточний момент комп'ютерна техніка досягла такого рівня розвитку, що дозволяє програмістам розробляти дуже реалістичні ігри з гарним графічним і звуковим оформленням. З кожним стрибком в області комп'ютерних технологій зростає кількість людей, яких в народі називають «ігровими фанатами» або «геймерами» (від англійського «game» - гра).

Основною діяльністю таких людей є гра на комп'ютері, коло соціальних контактів у них практично відсутній, вся інша діяльність спрямована лише існування, на задоволення потреб, а головне - на задоволення потреби в грі на комп'ютері.

Зрозуміло, в повній мірі це визначення підходить лише людям, насправді фанатично захоплюються комп'ютерними іграми, коли захопленість близька до хвороби. Але, так чи інакше, незаперечним є той факт, що феномен залежності людини від комп'ютерних ігор має місце.

Комп'ютерні ігри так міцно увійшли в наше життя, що тепер практично неможливо уявити комп'ютер, на якому не було б хоч однієї з них. Ні для кого не секрет, що комп'ютери вже давно досягли рівня швидкодії достатнього для роботи в основних офісних 6 додатках. Подальше збільшення продуктивності мотивує тільки загальна любов до комп'ютерних ігор, виробники яких стають все більш вимогливими до ресурсів комп'ютерів.

Відеогра - це спеціальна комп'ютерна програма або електронний пристрій, що реалізують ігровий процес (геймплей, від англ. Gameplay).

Синонімами поняття «відеогра» служать терміни «електронна гра», «комп'ютерна гра», «ігровий пристрій», «відеогра для ПК» і так далі. В

англомовному ігровому просторі для позначення відеоігор використовується словосполучення «video game».

Обов'язковими елементами будь відеоігри є візуальний інтерфейс, через який гравець (геймер, від англ. Gamer) управляє ігровим процесом, ігрове середовище - сеттинг (від англ. Setting), що включає в себе конкретний час, місце дії, зовнішні атрибути гри.

Нерідко в основі таких ігор лежать сюжети з кінофільмів, літературних творів, політичних подій. Можна спостерігати і зворотний процес, коли за мотивами тих чи інших відеоігор знімаються повнометражні фільми.

Одним з найбільш уживаних критеріїв класифікації відеоігор виступає їх жанр. Нижче ми представимо окремий список жанрів відеоігор, який отримав визнання більшості геймерів і оглядачів відеоігор.

Жанри відеоігор.

Класифікація за жанрами передбачає віднесення тієї чи іншої відеоігри до наступних категорій:

навчальні ігри (в їх основі лежить необхідність здійснення тих чи інших дій, спрямованих на тренування заздалегідь певного досвіду);

стратегії (сюжетні ігри, що вимагають від гравця аналізу ситуації і прийняття неочевидних рішень; щось більше, ніж просто логічні ігри на зразок шашок або шахів);

аркади (гри на спритність і уважність);

пригодницькі ігри («дії», як правило. спираються на історичні сюжети і сценарії);

електронні симулятори (віртуальні прообрази реальних об'єктів і процесів, пов'язаних з управлінням автомобілем або іншим пристроєм тощо);

екшени (від англ. action - дія; гри передбачають постійне клацання по кнопках і обертання мишею; гри з захоплюючим сюжетом, який передбачає активні і безперервні дії з боку гравця);

головоломки і логічно гри (всілякі настільні ігри, включаючи згадані шахи і шашки);

змішані (будь-яка комбінація, що припускає віднесення відеоігри більш ніж до одного жанру, наприклад, екшн і стратегія, навчальна гра і симулятор).

Основні тенденції ігрової індустрії останніх років - ухил в бік змішання жанрів, реалізації якісної тривимірної графіки, наповнення ігрових сценаріїв змістом і динамікою.

Залежно від числа використовуваних платформ відеоігри діляться на ті, які встановлюються на персональні комп'ютери, ті, під які розроблені спеціальні приставки (консолі), і, нарешті, ті, які встановлюються на мобільні пристрої.

Кілька застарілої виглядає класифікація відеоігор на мультиплатформенні і одноплатформенние. Останні зустрічаються вкрай рідко і в більшій мірі свідчать про непрофесіоналізм їхніх творців.

Число задіяних гравців може служити самостійним критерієм розподілу відеоігор на однопользовательские (один гравець) і розраховані на багато користувачів (два і більше гравців).

Наявність / відсутність мережевих версій - допоміжний критерій класифікації відеоігор на ті, які дозволяють грати за допомогою використання мережі Інтернет, і ті, які такої можливості гравцям не пропонують.

Залежно від використовуваних засобів оформлення відеоігри можуть бути текстовими або графічними (останні, в свою чергу, можуть бути дво- або тривимірними).

Більшість відеоігор створені для людей, хоча окремі їх різновиди розраховані на інших живих істот, наприклад, кішок і котів.

Це спеціальні програми або пристрою, що змушують домашнього вихованця стежити за зображенням мишки на екрані чи іншим чином проявляти активність.

В ході представленої роботи буде розібрано існуючі ігрові рушії, взявши за основу один буде розроблено комп’ютерну гру з системою відкритого світу та удосконаленою рпг-системою.

1 Дослідження предметної області та постановка задачі

1.1 Змістовний аналіз та опис предметної області, її структурних і функціональних особливостей

В основі сучасних розподілів відеоігор на жанри лежить вид активності, який найчастіше здійснює гравець в іграх даного жанру. Так відеоігри в загальному можуть поділятися на ігри руху, планування і сюжету або спілкування, дії та контролю. В багатьох класифікаціях визначення жанру відбувається за кількома осями. Наприклад, за двома осями сюжет — свобода дії, або трьома абстракція — симуляція — свобода. Проте найчастіше використовуваною класифікацією, хоч і не прийнятою усіма, жанри з якої зустрічаються в більшості існуючих. Жанр - сукупність ознак гри, яка визначає особливості власне ігрового процесу. Жанри рольових ігор багато в чому аналогічні літературним жанрам, жанрами кінематографа і так далі. Як і у випадку з останніми, чітке розподіл ігор на жанрові категорії багато в чому умовно, і нерідко гри знаходяться на стику жанрів. Жанр задає очікувану манеру дій ігрових персонажів, стилістику персонажів і їх оточення, типові прийоми ведення гри, набір на увазі для прискорення ігрового процесу умовностей (законів жанру). Чітке визначення жанру гри дозволяє гравцям розуміти, які деталі дій їх персонажів заслуговують докладного опису (або опрацювання на стадії створення персонажа), а які не настільки істотні.

На відміну від антуражу, жанр вказує на власне деталі процесу, а не на «декорації» в яких розгортається конкретна гра. Точно так само не слід плутати жанр з сюжетом конкретної гри - жанр є поняттям, об'єднуючим групи типових сюжетів, причому сюжет конкретної гри може включати перехід з одного жанру в інший (скажімо, після періоду розслідування персонажами діяльності якогось таємного товариства, довгий час грали в детективному жанрі, персонажі стикаються з наслідками його діяльності, і наступні кілька сесій представляють гру в жанрі хоррора або бойовика). В нашому випадку нас цікавить саме жанр RPG. Гравець асоціюється з конкретним персонажем або лідером команди, які

діють відповідно до правил своїх ролей. Наприклад, лицар не може того, що чарівник, кожна роль має свої особливості, а часом від неї залежить і розвиток сюжету. Мета ігрового процесу полягає у виконанні різноманітних завдань (квестів) для розвитку одного персонажа або групи. Можливі варіанти дій залежать від обраного образу персонажа, попередньо визначеного чи формованого самим гравцем.

Особливість жанра RPG в тому що гравці керують одними або кількома героями, описаними набором характеристик. У список характеристик можуть входити здоров'я, сила, спритність і різні здібності персонажа. В ході гри, як правило, показники характеристик героя ростуть, збільшується число здібностей, пасивних умінь. За рахунок цього герой здатний більш успішно справлятися з поставленими завданнями. В Action-RPG, на відміну від покрокових RPG, успішний результат всієї справи залежить не тільки від ступеня розвитку характеристик персонажа, а й швидкості реакції самого гравця.

Іноді рольові відеоігри поділяються за дизайном і побудовою сюжету. Так існує умовний поділ на рольові ігри західного зразка та східного.

Самим важливим рішенням при створенні гри є вибір движка.На данний момент вибирати нам доведеться з самих популярних і якісних движків у вільному доступі,таких движка є 2,Unreal engine 4 і Unity.

Unity — багатоплатформовий інструмент для розробки дво- та тривимірних додатків та ігор, що працює на операційних системах Windows і OS X. Створені за допомогою Unity застосування працюють під системами Windows, OS X, Android, Apple iOS, Linux, а також на гральних консолях Wii, PlayStation 3 і XBox 360.

Є можливість створювати інтернет-додатки за допомогою спеціального під'єднуваного модуля для браузера Unity, а також за допомогою експериментальної реалізації в межах модуля Adobe Flash Player. Застосування, створені за допомогою Unity, підтримують DirectX та OpenGL.

Технічні характеристики

Сценарії на C#, JavaScript та Boo;

Ігровий рушій повністю пов'язаний із середовищем розробки. Це дозволяє випробовувати гру прямо в редакторі;

- Робота з ресурсами можлива через звичайний Drag&Drop.
- Система успадкування об'єктів;
- Підтримка імпортування великої кількості форматів файлів;
- Вбудований генератор ландшафтів;
- Вбудована підтримка мережі;

Існує рішення для спільної розробки — Asset Server. Також можна використовувати зручний для користувача спосіб контролю версій. Наприклад, SVN або Source Gear.

Редактор Unity має простий Drag & Drop інтерфейс, який легко налаштовувати, що складається з різних вікон, завдяки чому можна проводити налагодження гри прямо в редакторі. Рушій підтримує три сценарних мови: C #, JavaScript (модифікація). Проект в Unity ділиться на сцени (рівні) — окремі файли, що містять свої ігрові світи зі своїм набором об'єктів, сценаріїв, і налаштувань. Сцени можуть містити в собі як, об'єкти (моделі), так і порожні ігрові об'єкти — тобто ті, які не мають моделі. Об'єкти, в свою чергу містять набори компонентів, з якими і взаємодіють скрипти. Також у них є назва (в Unity допускається наявність двох і більше об'єктів з однаковими назвами), може бути тег (мітка) і шар, на якому він повинен відображатися. Так, у будь-якого предмета на сцені обов'язково присутній компонент Transform — він зберігає в собі координати місця розташування, повороту і розмірів по всіх трьох осях. У об'єктів з видимою геометрією також за замовчуванням присутній компонент Mesh Renderer, що робить модель видимою.

Також Unity підтримує фізику твердих тіл і тканини, фізику типу Ragdoll (ганчіркова лялька). У редакторі є система успадкування об'єктів; дочірні об'єкти будуть повторювати всі зміни позиції, повороту і масштабу батьківського об'єкта. Скрипти в редакторі прикріплюються до об'єктів у вигляді окремих компонентів.

При імпорті текстури в рушій можна згенерувати alpha-канал, mip-рівні, normal-map, light-map, карту відображень, проте безпосередньо на модель

текстуру прикріпити не можна — буде створено матеріал, з яким буде призначений шейдер, і потім матеріал прикріпиться до моделі. Редактор Unity підтримує написання і редагування шейдерів. Крім того він містить компонент для створення анімації, яку також можна створити попередньо в 3D-редакторі та імпортувати разом з моделлю, а потім розбити на файли.

Unreal Engine — це ігровий рушій, розроблюваний і підтримуваний компанією Epic Games.

Перша гра, створена на цьому рушію — Unreal — з'явилася 1998року. З тих пір різні версії цього ігрового рушія використали в більш ніж сотні ігор, серед яких Deus Ex, Lineage II, Thief: Deadly Shadows, Postal 2, серіях ігор Brothers in Arms, серія ігор Splinter Cell, Tom Clancy's Rainbow Six, а також у відомих ігрових серіях Unreal і Unreal Tournament від самих Epic Games. Пристосований у першу чергу для шутерів від першої особи, рушій використовувався й при створенні ігор інших жанрів.

Написаний мовою C++, рушій дозволяє створювати ігри для більшості операційних систем і платформ: Microsoft Windows, Linux, Mac OS і Mac OS X, консолей Xbox, Xbox 360, PlayStation 2, PlayStation Portable, PlayStation 3, Wii, Dreamcast і Nintendo GameCube. У грудні Марк Рейн продемонстрував роботу рушія Unreal Engine 3 на iPod Touch і iPhone 3GS. У березні 2010 робота рушія була продемонстрована на комунікаторі Palm Pre, що базується на мобільній платформі webOS.

Для спрощення портування рушій використовує модульну систему залежних компонентів: підтримує різні системи рендерингу (Direct3D, OpenGL, Pixomatic; раніше підтримувалися Glide API, S3 Metal, PowerVR SGL), відтворення звуку (EAX, OpenAL, DirectSound3D; раніше підтримувалися A3D), засоби голосового відтворення тексту, розпізнавання мовлення (тільки для Xbox360, PlayStation 3, Nintendo Wii і Microsoft Windows, також планувалося для Linux і Mac), модулі для роботи з мережею й підтримка різних пристроїв вводу.

Для гри у мережі підтримуються технології Windows Live, Xbox Live, і GameSpy, включаючи до 64 гравців (клієнтів) одночасно. Попри те, що офіційно

засоби розробки не містять у собі підтримки великої кількості клієнтів на одному сервері, рушій використовувався для створення MMORPG-ігор. Один з найвідоміших представників жанру, Lineage II, використовує рушій Unreal Engine.

Усі елементи ігрового рушія представлені у вигляді об'єктів, що мають набір характеристик, і клас, який визначає доступні характеристики. У свою чергу будь-29 який клас є “дочірнім” класом object. Серед основних класів і об'єктів можна виділити наступні:

Актор (actor) — базовий клас, що містить усі об'єкти, які мають відношення до ігрового процесу й мають просторові координати.

Павн, пішак (pawn) — фізична модель гравця або об'єкта, керованого штучним інтелектом. Назва походить від англ. pawn — той, ким маніпулюють (pawn можна перевести також як пішак, тому такий об'єкт без якої-небудь моделі виглядає як пішак). Метод керування описаний спеціальним об'єктом, такий об'єкт називається контролером. Контролер штучного інтелекту описує лише загальну поведінку пішака під час ігрового процесу, а такі параметри як “здоров'я” (кількість пошкоджень, після яких пішак перестає функціонувати) або, наприклад, відстань, на якій пішак звертає увагу на звуки, задаються для кожного об'єкта окремо.

Світ, рівень (world, game level) — об'єкт, що характеризує загальні властивості “простору”, наприклад, силу тяжіння й туман, у якому розташовуються всі актори. Також може містити в собі параметри ігрового процесу, як, наприклад, ігровий режим, для якого призначений рівень.

Для роботи із простими й, як правило, нерухомими елементами ігрового простору (наприклад, стіни) використовується бінарна розбивка простору — увесь простір ділиться на “заповнене” і “порожнє”. В “порожній” частині простору розташовуються всі об'єкти а також тільки в ній може перебувати “точка спостереження” при відмальовці сцени. Можливість повного або часткового поміщення об'єктів в “заповнену” частину простору не виключається, однак може привести до неправильної обробки таких об'єктів (наприклад, розрахунки фізичної взаємодії) або неправильної відмальовки у випадку поміщення туди

“точки спостереження” (наприклад, ефект “залу дзеркал”). Усі пішаки, що потрапляють в “заповнену” частину простору, відразу “гинуть”.

Поверхня (surface) є основним елементом бінарного дерева простору. Ці елементи створюються на грані перетинання між “заповненою” і “порожньою” частинами простору. Група елементів бінарного дерева простору називається нодом (node, укр. вузол). Цей термін, як правило, уживається в контексті node count — кількість нодів на екрані або в ігровому просторі взагалі. Кількість нодів, одночасно видимих на екрані впливає на продуктивність при промальовуванні сцени. Якщо якийсь нод не потрапляє на екран або перекривається цілком іншими нодами, він не обраховуються — це служить для підвищення продуктивності, особливо в закритих просторах. Розбивка всього простору на групи нодів називається зонуванням. Для цього іноді використовуються портали — невидимі поверхні, які служать для того щоб вручну розділити великий нод на два менші. Крім порталів використовуються антипортали, які обмежують області відмальовки.

Опис “заповнених” і “порожніх” частин простору виконується за допомогою набору замкнених тривимірних об'єктів, складених з не пересічних поверхонь — брашей (brush, укр.пе́ нзель). Цей принцип побудови простору називається конструктивною суцільною геометрією. Геометрія може бути “аддитивною” (увесь простір початково “порожній”) і “віднімальною” (початково заповнений матерією простір).

Браши діляться на п'ять типів:

- Суцільні (solid) — повноцінно беруть участь у бінарній розбивці простору.
- Аддитивні (additive) — “заповнюють” бінарний простір.
- Віднімальні (substractive) — “вирізують” об'єми у просторі.
- Напів-Суцільні (semi-solid) — не впливають напряму на бінарне дерево простору, однак впливають на її фізичну модель. Можуть тільки “заповнювати” простір. Слугують для створення “невидимих” перешкод, а також зниження числа полігонів і нодів.

- Порожні (non-solid) — тільки створюють поверхні, не впливають на бінарне дерево простору. Використовуються переважно для створення об'ємів (volume) — частина простору, яка має властивості, відмінні від властивостей ігрового світу. Об'єми мають пріоритет, властивості об'єму з великим пріоритетом застосовуються до акторів, що 31 перебувають у ньому. Ігровий світ завжди має мінімальний пріоритет. За допомогою об'ємів можна змінити гравітацію, в'язкість, туман і таке інше. Об'єми, починаючи з версії рушія Unreal Engine 2, використовуються для створення води (але не водної поверхні).

У Unreal Engine 4 вбудований пост-процесінг. До сцени можна застосовувати bloom-ефект, тонування і антиалиасинг як глобально, так і до окремих її

Sequencer в Unreal Engine4 можна використовувати для створення сінематіков. Це потужний інструмент, що працює за принципом додавання об'єктів на тимчасову шкалу.

Жанр RPG в основному поділяють на 4 категорії.

1) Рольова система.Описує способи створення, зміни і розвитку персонажів, що підвищують їх ефективність в грі необхідні елементи.

Гравець керує одним (втілення, аватар) або декількома (група, партія) унікальними персонажами. Це необхідна умова для всіх рольових ігор. На відміну від чистих стратегій, його персонажі унікальні і мають ім'я. Гравець поступово покращує характеристики і / або навички (за допомогою внутрішньоігрових значень, найчастіше очок досвіду, одержуваних за виконання завдань, дослідження світу, діалоги, битви і так далі).

В процесі проходження відбуваються перевірки характеристик і / або здібностей і навичок. Персонажі можуть покращувати свої характеристики та / або здібності та навички за допомогою елементів спорядження. Теоретично можливо створити комп'ютерну рольову гру без спорядження. Відсутність елементів спорядження може служити розпізнавальним знаком пригодницьких ігор, найчастіше використовують інвентар виключно для предметів, які

застосовуються при вирішенні головоломок.важливі елемент.Гравець може створювати своїх персонажів.Гравець повинен планувати розвиток персонажа (-жей). Цей елемент відображає стратегічний аспект створення та розвитку персонажа, а також поєднання навичок в групі.

Основним способом вирішення проблем, взаємодії з ігровим світом і подолання перешкод є тактичне застосування навичок / здібностей персонажа / групи персонажів (навички самого гравця вторинні). Якщо ця умова не виконується, то швидше за все, перед нами один з видів бойовика, в якому вирішальне значення мають навички гравця, а не персонажа.

2) Дослідження.Описує способи переміщення персонажа по ігровому світу, все, що він може знайти, побачити, з чим він може взаємодіяти. Наприклад, ігрові області, предмети та інші об'єкти необхідні елементи.Персонаж гравця може взаємодіяти з ігровим світом і знаходити нові ігрові області.Персонаж гравця може знаходити предмети і зберігати їх в інвентарі. Теоретично можливо створити комп'ютерну рольову гру без інвентарю.Персонаж гравця може знаходити джерела інформації. Комп'ютерна рольова гра без пошуку інформації неможлива в принципі.важливі елементи.У грі є персонажі. Цей елемент віднесений до важливих тому, що перші дії часто не мали інших персонажів, крім того, яким керував гравець.Ви можете вибрати свій шлях (хоча б з декількох варіантів). Персонаж може впливати на ігровий світ (опускати важелі, натискати кнопки, відкривати скрині).

Ігровий світ впливає на персонажа (-жей) (погода, пастки, отруєні місця). Існують спочатку недоступні області ігрового світу, в які можна потрапити, лише поліпшивши навички персонажа, виконавши завдання або вирішивши головоломку (відкрити замок, подолати перешкоду, починають міст, розвіяти магію і так далі). Дослідження ігрового світу також має залежати від навичок.

3)Сюжет.Включає в себе всі елементи розповіді, такі як ігровий світ, його історію, персонажів, діалоги, завдання, опису, сюжетні лінії і способи взаємодії цих компонентів.Необхідні елементи персонаж гравця може отримувати інформацію з інформаційних джерел (натяки, цілі, завдання, навички, заклинання,

навчання). Цей елемент тісно пов'язаний з третім необхідним елементом категорії Дослідження. Після того, як джерело інформації знайдений, її необхідно отримати.

Персонаж гравця може виконувати завдання (існує принаймні одне сюжетне завдання).Персонаж гравця просувається по ланцюжку пов'язаних подій і грає в них свою роль.Важливі елементи сюжета залежать від рішень гравця, дій персонажів і характеристик / здібностей / навичок. Персонаж може взаємодіяти з джерелами інформації (наприклад, розмовляти з персонажами).В процесі взаємодії персонаж може зробити вибір.Хоча б деякі з цих виборів повинні мати наслідки.Просування по сюжету вимагає від гравця обмірковування (моральні дилеми, головоломки).

4)Бойова система.Пояснює вплив рольової системи, досліджень і сюжету на результати битви (або, більш узагальнено, на результат вирішення конфлікту).важливі елементи Ефективність в битві залежить від характеристик і навичок персонажа (кількість пошкоджень, шанс потрапляння, можливість використання певних видів зброї і так далі). Бойова система, не пов'язана з характеристиками і навичками персонажа, є вірною ознакою бойовика, належного виключно на навички гравця.

У боях присутній елемент випадковості (кидки внутрішньоігрових кубиків). Практично у всіх комп'ютерних рольових іграх присутні кидки віртуальних кубиків і імовірнісні функції.

1.2 Аналіз наявного програмно-технічного забезпечення предметної області.

На данний момент є багато хороших RPG,однак серед них можна виділити кращих,наприклад The Witcher 3:Wild hunt.Протягом гри під управлінням гравця знаходиться єдиний, заздалегідь заданий герой - відьмак Геральт з Рівії, який подорожує по ігровому світу, спілкується з неігровими персонажами і виконує різні завдання, зокрема, пов'язані з пошуком скарбів і полюванням на

чудовиськ.Геральт використовує в боях два меча: сталевий завдає більшої шкоди людям, срібний - чудовиськам. У багатьох боях необхідне знання слабостей противника: гравцеві пропонується внутрішньо довідник, який повідомляє про те, до яких здібностям та пошкодження речей ворог вразливий. Додатковим спорядженням Геральта є різноманітні бомби і арбалет - вони дозволяють атакувати ворогів з відстані, боротися з літаючими або підводними противниками.

Гра включає в себе складну систему «алхімії», що дозволяє складати різноманітні зілля з зірваних рослин, частин тіл чудовиськ і інших предметів. Ці зілля дозволяють Геральту на час отримати особливі здібності, збільшити його характеристики або шкоди проти певних типів ворогів. Протягом гри перед гравцем ставляться різні завдання. «Основні» завдання дозволяють гравцеві просунутися далі по сюжету гри; в необов'язкових побічних завданнях Геральт може допомогти місцевим жителям. При виконанні так званих «Відьмачий контрактів» Геральт повинен вистежувати і вбивати особливо сильних чудовиськ; ще одним заняттям є полювання за скарбами і кресленнями Відьмачий спорядження.

Гравець може отримувати завдання різними способами, говорячи з неігровими персонажами, оглядаючи дошки оголошень в селах або, наприклад, обшукуючи трупи. Виконання завдань часто включає в себе детективні елементи: збираючи інформацію, гравець повинен розпитувати персонажів гри, знаходити докази і рухатися по слідах. «Відьмаче чуття» являє собою особливий режим гри, в якому підсвічуються предмети, що представляють для Геральта інтерес, такі, як трупи і скрині з цінними речами, різні докази і сліди.Розробляла гру студія розробників CD Projekt Red з Польші, і також самостійно її випустивши,на своєму власному движку REDengine 3. Движок був офіційно представлений публіці 1 лютого 2013 року.

Попередні версії движка використовувалися при створенні гри «Відьмак 2: Вбивці королів»; REDengine 3 дозволяє потоковим чином довантажувати масштабні відкриті світи, що дозволило значно збільшити розміри ігрового світу. Для управління окклюзівном обрізанням в движок вбудували технологію Umbra

3 Visibility Solution. Інженери з Umbra і CDP продемонстрували роботу технології на Game Developers Conference 2014.

Ще одним прикладом є The Elder Scrols V:Skyrim розроблений Bethesda Game Studios, і випущена Bethesda Softworks. Гравець може вільно мандрувати по всій території провінції Скайрім, що включає в себе дев'ять великих міст, безліч дрібних селищ, а також великі простори диких земель і високих гір. У містах гравець може вдаватися до таких занять, як приготування їжі і зілля, Зачарований предметів, фермерство, робота з рудою або ковальська справа. При цьому розробники відзначали, що ігровий процес може зайняти близько 500 годин. Рівень персонажа підвищується в міру поліпшення навичок, за певну кількість таких поліпшень (залежить від рівня гравця і рівня навику) персонаж отримує один рівень.

У грі є система «автолевелінга» - чим вище рівень ігрового персонажа, тим більше цінні предмети він знаходить в грі і тим більше небезпечні противники йому протистоять. Розробники вирішили повністю відмовитися від системи класів, представленої в Oblivion, попередній грі серії. У грі присутні таланти - особливі здібності, пов'язані з певними навичками гравця. Таланти організовані в розгалужену систему під назвою «древо навичок». Кожне підвищення рівня дозволяє взяти черговий талант. Всього в грі 280 талантів. HUD-інтерфейс виникає на екрані тільки тоді, коли здоров'я, запас сил, або магія гравця знаходяться нижче максимуму, встановленого рівнем. Компас - єдиний елемент інтерфейсу, який присутній на екрані весь час.

Предмети можна зберігати в «вибране» меню, а меню інвентарю (при відкритті якого гра ставиться на паузу) за своїм візуальному стилю нагадує компас. Перебуваючи в режимі інвентарю, кожен ігровий предмет можна обертати, наближати і розглядати в повному. Зброю можна призначати як в праву, так і в ліву руку, що дозволяє нести по одному виду зброї в кожній руці.

Спеціальне меню дозволить гравцеві швидко перемикатися між різними типами зброї та обладунків. Щитом можна атакувати, а на те щоб вибудувати їм блок потрібен певний час. Кожен з типів зброї (одноручне, двуручное, цибуля)

має своє особливе перевагу і призначення. При стрільбі з лука потрібно більше часу на те, щоб натягнути тятиву, ніж це було в попередніх іграх серії Elder Scrolls, проте постріли при цьому наносять противнику більших збитків. З цієї причини стріли стоять вельми дорого.

Гравець, озброєний луком, може використовувати його для захисту в ближньому бою, точно так же, як може використовувати для цієї мети двуручное зброю або щит (хоча щит в даній ситуації найбільш ефективний). Різні типи заклинань мають різні якості - так, заклинання холоду уповільнює і при цьому виснажує запас сил, в той час як заклинання вогню завдає тривалий шкоди на всьому протязі горіння, і навіть може запалити навколишні об'єкти, а заклинання електрики спалює ворожу ману (магію), а також може дезінтегрувати противника. Заклинання можна вимовляти відразу з двох рук. Поєднання заклинань теж можливо, наприклад, вогонь і блискавка. Інші доступні в грі заклинання можуть, наприклад, відновлювати втрачені очки здоров'я, на час оживляти мертвих, створювати джерела світла в темних приміщеннях або перетворювати залізну руду в срібну.

Обидві ігри є дивовижними, але Witcher 3 дає нам більш характерні символи, більш цікаву боротьбу, і світ, який занурює гравців через прекрасні параметри і механіку.

1.3 Визначення вимог до програмного забезпечення та розробка технічного завдання.

Комп’ютерні ігри на данний момент являють собою ще одну прибуткову сферу розваг,як наприклад Кіноіндустрія,тому в першу чергу розробники хочуть отримати більший прибуток чим вони вклали.

У рольових іграх користувачі можуть вибрати собі відповідну роль, хорошого або поганого героя, в такій грі можна спробувати себе в різних ролях.Також часто дається можливість подоружувати по карті в пошуках різних деталей,об’єктів,завдань, цінних засобів які стануть у пригоді гравцеві.Також

дається можливість розвивати свого персонажа і вдосконалювати його навики, та отримувати нові.

Наша Rpg буде конкретно надавати гравцеві можливість:Керувати заздалегіть підготовленим персонажом від третього лиця,Подорожувати по відкритій для вивчення карті,брати завдання в заздалегіть підготовленних персонжаів(НПС) по типу вбий-принеси,керувати вибором завданнь черех журнал який доступний гравцеві або перключати їх на міні-списку, вступати в бій з ворогами,можливість підіймати свій рівень персонажа через нагороду за знащення ворогів та завдання у вигляді балів досвіду.

Системні вимоги — стале поняття, яке використовується для опису характеристик, яким повинен відповідати цифровий пристрій (ПК, гральна консоль, мобільний телефон тощо) для коректної роботи певного програмного забезпечення,в нашому випадку гри. Ці вимоги можуть описувати, як апаратне забезпечення, так і програмне забезпечення (необхідні драйвери, операційна система тощо). Розрізняють мінімальні та рекомендовані системні вимоги. Якщо мінімальні системні вимоги показують, яка конфігурація системи цілковито необхідна для запуску ПЗ, то рекомендовані системні вимоги показують, яка конфігурація здатна забезпечити максимально комфортні умови роботи ПЗ. Також інколи виділяють максимальні системні вимоги, при яких забезпечується повна функціональність і можливість користуватись усіма послугами потрібної програми. Оскільки серед поширених програмних продуктів найбільш технічно вимогливими є відеоігри, то для них відомості про системні вимоги є дуже бажаними.

Виходячи з предметної галузі сформолюємо постановку задачі:Розробити ПЗ (Комп’ютерна гра на Unreal engine.),яке задовільняє наступним вимогам: проводити вільний час користувачу в грі по типу жанра Action-RPG де можна:Керувати персонажем,брати завдання,відкривати журнал,керувати списком завдань,вступати в бій з вороже налаштованими НПС,подорожувати по відкритій карті,підіймати свій рівень персонажа,використовуючи ПК платформу для запуска з мінімальними системними вимогами.

Таблиця 1.1 Опис корисутвачів

Актор	Короткий опис
Гравець	Можливість купити,запускати та використовувати гру(Продукт).Керування персонажем,можливість входити в бій з супротивниками,брати квести та подорожувати по відкритій карті.
Представники онлайн сервісу(площадки де буде публікація)	Надання можливості продавати в себе продукт(гру)з стягненням комісії з продажу.

Таблиця 1.2 Опис варіантів використання

Актор	Найменування ВВ	Опис ВВ
Гравець	Реєстрація покупки та використання	Дозволяє купити продукт(гру),використовувати його.Керувати заздалегіть підготовленим персонажем,входити в бій з супротивниками,брати завдання в НПС та подорожувати по відкритій карті.
...	...	...
Представники онлайн сервісу(площадки де буде публікація).	Продаж продукту(гри)	Розміщує в себе на онлайн сервісі продукт (гру) з можливістю для користувача(відвідувача сервісу)купити його та стягувати процент з комісії за продаж.

1.4 Вибір ігрового рушія для створення гри.

Unity - це інструмент для розробки двох-і тривимірних додатків та ігор, що працює під операційними системами Windows, Linux і OS X. Створені за допомогою Unity програми працюють під операційними системами Windows, OS X, Windows Phone, Android, Apple iOS, Linux, а також на ігрових приставках Wii, PlayStation 3, PlayStation 4,Xbox 360, Xbox One і MotionParallax3D.

Архітектура проекту Unity3D заснована на шаблоні EntityComponent-System. Згідно з цим шаблоном, додаток складається з базових сутностей, функціональність яких розширюється за допомогою спеціалізованих компонентів.

Проект Unity3D складається з декількох сцен (Scene), на яких розташовані ігрові об'єкти (GameObject) з прикріпленими до них компонентами (Component). У кожного ігрового об'єкта є обов'язковий компонент Transform, який відповідає за розташування об'єкта на сцені. Крім цього, можуть бути підключені як готові компоненти (наприклад, Rigidbody, що відповідає за фізичну симуляцію), так і призначені для користувача компоненти.Безпосередньо програмування в Unity3D полягає в першу чергу в розробці користувальницьких класів, які підключаються до ігрових об'єктів як компоненти. Всі такі класи повинні успадковуватися від класу MonoBehaviour. Вказівка цього відносини значне перевантаження б діаграму, тому для позначення классовкомпонентов до їх імен доданий суфікс «Script».

Unreal Engine - ігровий рушій, розроблюваний і підтримуваний компанією Epic Games.

Перша гра, створена на цьому рушії — Unreal, яка з'явилася 1998 року. З тих пір різні версії цього ігрового рушія використали в більш ніж сотні ігор, серед яких Deus Ex, Lineage II, Thief: Deadly Shadows, Postal 2, серіях ігор Brothers in Arms, серія ігор Splinter Cell, Tom Clancy's Rainbow Six, а також у відомих ігрових серіях Unreal і Unreal Tournament від самих Epic Games. Пристосований у першу

чергу для шутерів від першої особи, рушій використовувався й при створенні ігор інших жанрів.

Написаний мовою C++, рушій дозволяє створювати ігри для більшості операційних систем і платформ: Microsoft Windows, Linux, Mac OS і Mac OS X, консолей Xbox, Xbox 360, PlayStation 2, PlayStation Portable, PlayStation 3, Wii, Dreamcast і Nintendo GameCube. У грудні Марк Рейн продемонстрував роботу рушія Unreal Engine 3 на iPod Touch і iPhone 3GS. У березні 2010 робота рушія була продемонстрована на комунікаторі Palm Pre, що базується на мобільній платформі webOS.

Для спрощення портування рушій використовує модульну систему залежних компонентів: підтримує різні системи рендерингу, відтворення звуку, засоби голосового відтворення тексту, розпізнавання мовлення, модулі для роботи з мережею й підтримка різних пристроїв вводу.

Для гри у мережі підтримуються технології Windows Live, Xbox Live, і GameSpy, що дає можливість підключити до 64 гравців (клієнтів) одночасно. Попри те, що офіційно засоби розробки не містять у собі підтримки великої кількості клієнтів на одному сервері, рушій використовувався для створення MMORPG-ігор. Один з найвідоміших представників жанру, Lineage II, використовує рушій Unreal Engine.

Ціновий пункт був безкоштовним, доки гра не нараховувала певну суму грошей, тоді розробник повинен був оплатити ліцензію. Нова модель - це підписка, починаючи від 19 доларів на місяць. Існує також велика крива навчання, тому не підходить для нових починаючих розробників ігор.

Загалом це очевидно, що це найпотужніший двигун для великих студій, але, можливо, не ідеально підходить для невеликих команд Indie з низьким бюджетом.

Розібравши всі сильні та слабкі сторони таких двигунів, як:

- Unity
- Unreal Engine 4

, а також зробивши аналіз їх можливостей і вивівши рейтинг кожного з них,можна зробити висновок, що Unreal Engine 4 майже ідеально підходить для розробки гри.

2. Проектування програманого забезпечення

2.1 Проектування архітектури програмного забезпечення.

Ми будемо вікорістовуаті Абстратній тип данніх для архітектурі нашого проекту, де тип даніх візначається поведінкою (семантикою) з точки зору корістувача даніх, а саме в термінах можлівіх значень, можлівіх операцій над данімі цього типу и поведінкі ціх операцій. Весь внутрішня структура такого тіпу захована від розробніка - в цьому і полягає суть абстракції. Абстрактній тип Даних візначає набір функцій, незалежних від конкретної реализации типу, для оперування его значення. Конкретні реализации АТД назіваються структурами Даних.

Відеогра передбачає створення графікі, звуків та внутрішньоігровіх текстів. Концепт-арти віконуються на папері або комп'ютері, зазвічай в кількох варіантах. На основі концепт-артів художнікамі затверджуються і створюються двовімірні або трівімірні моделі персонажів, предметі та декорації. Для цього художнікі працюють в програмах, прізначеніх для роботі із графікою.

Щоб моделі рухалися, їх потрібно анімувати в спеціально розроблених програмах для даного типу роботи, або самому движку. Створюються різні набори рухів гравця, які будуть виконуватись залежно від конкретних дій гравця за допомогою програмного коду. У випадку двовимірної графіки це набори спеціальних спрайтів, де кожна окрема картинка є окремим кадром.

Візуальні ефекти розроблені для того щоб зробити гру видовищнішою і задати стиль гри. Серед них деякі додають реалістичності для гри, такі як відкидання тіней, заломлення світла, постріли і вибухи. Інші позначають стани і дії персонажа, які визначають стиль виконання гри.За реалізацію картинки і звуку відповідають графічний і звуковий рушії.

Для звукового оформлення гри записується музика і відбувається озвучування персонажів. Крім того для повноцінного звукового оформлення потрібні ефекти, як кроки, звуки пострілів. Вони можуть обиратися з вільних бібліотек чи записуватися окремо. Декотрі композитори спеціалізуються на

створенні музики до ігор. Музика може виконуватися цілими професійними оркестрами, мати пісенний супровід. Діалоги персонажів часто озвучуються спеціальними акторами на студіях озвучування.

Ігрова механіка визначає насиченість ігрового процесу, правила, за якими створюють умови для (абстрактні рамки) для гравця. Основою механіки є ігрові об'єкти, такі як персонажі, об'єкти, якими вони можуть маніпулювати. Частиною ігровової механіки є управління персонажем, яким гравець керує персонажем та ігровим світом. Наприклад, як задається напрям руху персонажа, як працює взаємодія з віртуальними предметами. Крім того на етапі розробки механіки створюється користувацький інтерфейс, який інформує гравця і дозволяє взаємодіяти із світом гри.

Ігри зазвичай поділяються на рівні, щоб комп'ютер не навантажувався обчисленням всього ігрового світу. В нових іграх світ часто створюється так, щоб не мати чітких поділів на локації. Дизайнер рівнів розміщує готові об'єкти в ігровому світі та продумує їх рухи. Компонування рівнів визначає наскільки цікавою буде гра, які можливості будуть у гравця вирішити конкретну ситуацію.

За взаємодію об'єктів, яка відбувається без контролю гравця, відповідає фізичний рушій. До прикладу, він реалізує закони інерції, гравітацію, поведінку рідин, властивості предметів. Штучний інтелект (ШІ) відповідає за поведінку персонажів, як вони реагуватимуть на дії гравця. Багато подій в грі відбуваються за скриптами. Самі події придумуються сценаристами, а скрипти реалізуються програмістами.

Одною з головних частин моєї робити є розробка моделей та керування персонажами. До моделей ми віднесемо також анімації та ефекти які можуть використовуватись наприклад при рухах персонажа,іншi частини як наприклад звук до нашої робити не входять.Що до керування персонажами то тут потрібно розуміти що сюди(окрім розробки моделей) також входить розробка завдань для гравця,поведінка різних типів самих персонажів(нейтральні,ворожі т.д.),розробка бойової системи.Після переліку всіх аспектів наших задач можна виділити системи які ми будем розроблювати незалежно,тобто:Персонажі(НПС), система

завдань,бойова система,анімації-ефекти,система керування заданим для гравця персонажем.

Взаємодія наші система буде йти за таким принципом: Ми будемо Керувати заздалегіть створеним персонажем яким зможемо подоружваті по карті та брати завдання.Завадння будут відаватісь через НПС, Самі НПС будут поділенні на нейтральні (что и будут нам відаваті завдання) и ворожіх.Завданяя будут поділятісь на поиск (Пошук об'єкта з Яким можна взаємодіяті, підібраті), та знищення ворожок НПС, Ворожі НПС якіх ми будемо зустрічаті на карті будут нас атакуваті, з можлівістю підконтрольнім нам персонажу атакуваті їх такоже до полного обнулення їхньої шкали здоров'я, после чого буде Виконувати очищення (відалення) з карти НПС.Самі ж НПС будут мати такоже можлівість появлятісь (респавнітісь) после умовної перемоги над ними.

2.2 Детальне проектування програмного забезпечення.

Статічні карті або локації - ігровій рівень заздалегідь намальованій художніком однією великий картінкою, а пізніше по ходу розробкі грі на ньому розміщуються ігрові об'єкті і персонажі з Якими може взаємодіяті гравець.В нашому віпадку ми будем вікорістовуваті готові паки моделей, які можемо взяти в онлайн -магазіні розробніків ігрового рушія Unreal Engine 4, встановівші до нашого проекту гри.

Масштаб для об'єктів як правило візначається віходячі із загального ігрового простору і залежіть від того в якому жанрі гра і чи буде карта віходити за межі екрану з можлівістю прокруткі або переходу від екрану до екрану. Практічно будь-яка дінамічна карта будується на основі сіткі - це те ж саме що і зошіт в клітінку. Сітка спрощує побудову світу і його життя в майбутньому.

У моїй грі ігровий екран запланований в 1980x1024 пікселів. Щоб визначити масштаб для такого простору я просто створив полотно з такими розмірами і розкреслив його в клітинку, розміри моєї клітини складають 64x64 пікселя. Після чого став малювати схематичні начерки героїв і об'єктів підбираючи оптимальний

розмір сітки, так щоб герой не здавався занадто маленький і коло героя був достатній огляд на всі боки. Щоб світ було простіше створювати, бажано робити так щоб кожен об'єкт міг займати тільки одну клітинку. Але існує необхідність коли повинні бути об'єкти менше однієї клітини або більше однієї клітини.

Безшовний світ - ігрове поле, для переміщення між локаціями якого не доводиться чекати дозавантаження. Для розробників реалізація безшовного світу складніше, але для гравця такі ігри більш привабливі. Завдяки безшовні досягається безперервне занурення в ігровий процес, від якого не відволікають завантаження між локаціями в світах зі швами.

Наприклад, в безшовному світі перехід з вулиці в будинок не вимагає очікування - гравець продовжує грати так, як ніби він зайшов в приміщення в реальному житті. Це позбавляє відчуття фальша і додає ігрового азарту.

Для Геймдизайнер створення безшовного світу - завдання, до якої доводиться підходити відразу з декількох сторін. У число прийомів, використовуваних для створення повноцінного безшовного світу, входять:

Відкритий світ - місцевість без входу і виходу. Зазвичай це місто, острів або поселення, при спробах вибратися з яких персонаж наштовхується на стіну, пробку або щось подібне, що заважає перетнути кордон світу. В кінці-кінців, гравцеві це набридає, а при спробах повністю пройти карту він раз у раз знаходить нові цікаві мікролокації або квести, тому бажання знайти кінець світу відходить на другий план.

Завдання геймдизайнера при створенні безшовного світу - розробити локацію так, щоб вона повертала гравця до гри, а не супроводжувала його пошуків кінця карти. Тому міста можуть бути закільцьовані - всі вулиці будуть вести один на одного, а основна гра відбуватиметься десь в центрі карти.

У той же час безшовний світ відкриває нові можливості для досліджень місцевості. Наприклад, вентиляційні ходи і шахти в багатьох іграх з відкритим світом знаходяться в постійній досяжності. Дослідження їх захоплює людину і дає йому шлях безпечного переміщення між ділянками ігрової карти, що особливо важливо, якщо в грі часто ведуться поєдинки.

Незважаючи на те, що в реальному житті вентиляція є далеко не скрізь, а людина середньої комплекції навряд чи зміг би вільно по ній переміщатися, така опція в грі не здається дивною, а навпаки, додає ефект занурення. Це стосується і інших способів переміщення у відкритому безшовному світі.

Використання реалістичних локацій також йде на користь занурення. Театр, торговий центр, концертний зал - місця, в яких людина неодноразово бував в життя. Зберігаючи логіку будови локації, ви позбавляєте гравця від втрати часу на вивчення карти: опинившись в знайомої місцевості, простіше діяти інтуїтивно.

При спробах зробити гру достовірної, розробникам важливо пам'ятати про збереження почуття безпеки у гравця. Тому в іграх повинні залишатися місця, за якими персонаж може сховатися, навіть якщо в реальному житті вбрання укриття не забезпечила б безпеку.

Так. наприклад, в іграх з перестрілками персонаж ховається за меблями, поручнями сходів, огородженнями та іншими сумнівними предметами. Але, на відміну від реального життя, в грі такі укриття дають відчуття безпеки. При цьому, незалежно від габаритів укриття, персонажу завжди вдається за ним сховатися.

До цього відноситься і створення місій, які неможливо провалити, хіба що спеціально постаратися. Гра, що дає гравцеві кошти для досягнення цілей, завжди у виграші, тому що не викликає негативних емоцій.

Ігрові інтер'єри на порядок більше, ніж інтер'єри квартир в реальному житті. Це обумовлено особливостями управління та камери - переходячи між ракурсами від першої особи і від третьої особи, камера вимагає простору. Тому ігрові квартири більше реальних майже в два рази - щоб гравцеві було достатньо місця для проходу.

В умовах реалістичних будівель розробникам доводиться шукати компромісні рішення. Наприклад, в старовинних вузьких будівлях зазвичай розташовується тільки одна квартира, потрапити в яку можна тільки через вікно або вентиляцію - для сходів місця не залишається.

При створенні безшовного світу з великими будинками простіше - там досить простору, щоб розмістити кілька внутрішніх локацій, між якими зможе

переміщатися гравець. Додаючи ультрасучасний дизайн творець гри також виграє, відволікаючи гравця від масштабів споруди і акцентуючи увагу на її внутрішньому наповненні. Цим прийомом часто користувалися творці ігор 10-15 років тому, коли рівень графіки був не таким високим, щоб реалістично передати, наприклад, старовинне місто, але достатнім для відтворення космічних станцій без дрібних деталей.

Візуальна цілісність безшовного світу досягається за рахунок використання модульних деталей. Наприклад, використання подібних елементів фасадів старовинних будівель сприяє підвищенню цілісного сприйняття локації. Модульні будівлі типові і для реального світу, тому в процесі гри мозку не доводиться відволікатися на розгляд деталей - вони все однотипні.

Модульність безшовного світу зручна і для розробників. Опрацьовуючи дизайн рівнів, у них є можливість скористатися таким конструктором з модулів, в яких потрібно змінити тільки деякі деталі. Це дозволяє зосередитися на дизайні переходів між локаціями і їх внутрішньому оснащенні і не витрачати час на химерні несхожі один на одного будинку.

Модулі, які використовуються для створення безшовного світу, впливають також на сталість ігрової локації. Важливо дотримуватися однотипність дій, які гравець може зробити з однаковими предметами в різних локаціях.

Наприклад, якщо в одній будівлі персонаж може призвести взаємодія з плитою, шафою або холодильником, то і в іншій будівлі необхідно залишити для нього цю можливість. Недотримання принципу сталість може викликати дисонанс і відволікати від ігрового процесу. До того ж, взаємодія з предметами може бути частиною геймплея, а відсуваючи речі, персонаж може знаходити приємні бонуси, які знадобляться йому в грі.

Дотримання однакових правил гри для об'єктів поглиблює занурення людини в гру. Ці правила визначають безшовний світ і роблять його схожим з реальністю, так як залишаються незмінними для всіх однотипних об'єктів - точно так, як і в реальному житті. Проводячи аналогію з реальністю, уявіть собі, що в

одній квартирі вас обпікає включена плита, а в іншій ви спокійно до неї доторкається. Це викликало б масу питань.

Ці основні принципи дозволяють створювати безшовні реалістичні ігрові світи, в яких гравець відчуває себе частиною ігрового процесу. Незважаючи на обмежені масштаби ігрової карти, поетапно продумані в зв'язці сюжет, геймплей і світ дозволяють утримувати увагу гравця на прописаних завданнях і при цьому підтримувати його інтерес до гри.

Портал - це вікно, яке виходить не назовні, а в інше місце, то є ми локально задаємо певну точку огляду щодо об'єкта і репліціруем цю точку огляду десь ще. Користуючись цим принципом, ми можемо поєднати два простору, навіть якщо вони знаходяться дуже далеко один від одного. Вікно нагадує маску, яка дозволяє нам дізнатися, де і коли відображати інший простір, замість справжнього. Так як вихідна точка огляду реплицируется в інше місце, це дає нам ілюзію безперервності.

Я побудував свою систему на підставі двох основних класів, керованих PlayerController і Character. Клас Portal - це справжня точка входу в портал, точкою огляду / виходу якого є актор Target. Також тут є Portal Manager, який породжується PlayerController і оновлюється Character для управління кожним порталом на рівні і їх оновленням, а також для маніпулювання об'єктом SceneCapture (який є загальним для всіх порталів).

Я почав з актора порталу, який буде використовуватися для завдання «вікон», через які я буду дивитися на рівень. Завдання актора - надання інформації щодо гравця для обчислення різних позицій і поворотів.

Також він буде займатися розпізнаванням того, перетинає чи гравець портал, і його телепортацией.

Перш ніж почати детально розглядати актора, дозвольте пояснити кілька концепцій, які я створив для управління системою порталів:

Для зручного відмови від обчислень у порталу є стан «активний-неактивний». Цей стан оновлюється Portal Manager.

Портал має передню і задню боку, визначаються його позицією і напрямком (вектором forward).

Щоб дізнатися, перетинає чи гравець портал, він зберігає попереднє положення гравця і порівнює його з поточним. Якщо в попередньому такті гравець перебував перед порталом, а в поточному - за ним, то ми вважаємо, що гравець його перетнув. Зворотне поведінку ігнорується.

У порталу є обмежує обсяг, щоб не виконувати обчислень і перевірок, поки гравець не знаходиться в цьому обсязі. Приклад: ігнорувати перетин, якщо гравець насправді не стосується порталу.

Місцезнаходження гравця обчислюється з місця розташування камери, щоб забезпечити правильну поведінку в разі, коли точка огляду перетинає портал, але не тіло гравця.

Портал отримує Render Target, який відображає в кожному такті іншу точку огляду на випадок, якщо текстура в наступний раз буде невірною і зажадає заміни.

Портал зберігає посилання на інший актор, який має назву Target, щоб знати, де знаходиться інший простір, з яким потрібно зв'язатися.

Скориставшись цими правилами, я створив в якості початкової точки новий клас ExedrePortal, успадковані від AActor.

Конструктор тут займається підготовкою кореневих компонентів. Я вирішив створювати два кореневих компонента, тому що актор порталу буде поєднувати в собі і графічні ефекти, і колізії / розпізнавання. Тому мені потрібен був простий спосіб для визначення того, де знаходиться площину вікна / порталу, без необхідності використання функцій блюпрінтов або інших трюків. PortalRootComponent буде в подальшому основою для всіх обчислень, пов'язаних з порталом. Root порталу заданий як динамічний, на випадок, якщо клас Blueprint анімує його (наприклад, використовує анімацію відкриття / закриття).

Остання частина актора порталу, яку ми розглянемо - це функція TeleportActor ().

При телепортувані актора з точки А в точку В потрібно реплицировать його рух і позицію. Наприклад, якщо в портал йде гравець, то в поєднанні з

відповідними візуальними ефектами йому буде здаватися, що він пройшов в звичайні двері.

Перетин порталу відчувається як рух по прямій лінії, але в реальності відбувається зовсім інше. При виході з порталу гравець може з'явитись в іншій локації.

Анімації персонажа важлива частина тому приділити увагу їм також дуже важливо.Для того щоб анімації змогли працювати нам потрібно для початку в Блюпрінті самого персонажа прописати блоки коду в яких буде прописна можливість рухатись нашому персонажу,далі потрібно створити анімаційний Блюпринт для нашого персонажа де будуть прописані виклики потрібних нам команд для руху та всіх можливих і достпних анімацій взагалом.

В сам кінець для того щоб ми мали змогу грати нашим персонажем, який у нас є, ми повинні створити блюпринт його класу та запустити в движку саме його в опціях вибора классів,так як в іншому випадку ми будем керувати модель'ю андроїда що надається самим движком в якості тестового персонажа гравця, з базовою можливістю бігу та стрибків.Тобто ми повинні надати движку зрозуміти який клас персонажа ми хочем мати в якості основного,і який слід запускати замість того що був раніше.

Тепер наша задача створити Нейтральних та ворожих НПС,для початку зробимо нейтральних.Потрібно створити Блюпринт типу Character після чого припсати можливості взаємодіяти нашому персонажу з ним на видачу завдання коли ми вохдимо в радіус для розмови.Щоб зробити радіус для розмови ми просто на нашого персонажа повішаємо капсулу(невидиму) без колізії яку витягнемо вперед і просто зробимо її тим самим радіусом в якому ми можемо взяємодіяти,зкапусла буде на 7 метрів в перед.Тепер потрібно прописати можливість брати завдання в наших НПС у блюпрінті нашого персонажа,де поставимо виклик на клавішу – E.Тепер залишилось розробити тільки самі завдання.Створюємо блюпринт в якому ми прописуємо виконання наших завдань.Перший блок скріпта буде по знаходженю об'єкта розміщеного на карті який гравець може підняти якщо увійде в радіус взаємодії через клавішу - E яку

ми припсали раніше в блюпрінті нашого песронажа.Далі нам потрібно створити блюпринт в якому буде шаблон який ми будемо виставляти нашим НПС,в цьому шаблоні будуть можливості по пропису потрібного нам завдання для нашого персонажа з логікою виконання чи провалу також всебічному налаштуванню різних функцій по інтерефейсу,як наприклад описання завдання,проте на цьому ми небуло зациклювати увагу так як це не наша частина роботи.Також ми можемо створити можливість і відміни завдань та ,однак це теж в нашу частину роботи не входить.

Тепер ми повинні створити ворожих НПС.Для початку ми свторюємо блюпринт з модель'ю наших супротивників і назвемо його як Севарог. В ньому ми створюємо блоки скриптів по атакуваню гравця з скриптом знаходження через умовний конус який ми можемо виставити перед севарогом,входячи в зону конуса скрипт буде вираховувати де знаходиться гравець і буде задавати функцію переміщення до нас Севарога, також ми будем задавати кількість числа здоров'я,його рівень,урон атаки та логіку смерті,де буде знищення моделі після обнулення кількості числа здоров'я.Крім блюпринта севарога нам також треба створити для ворога і блпринт знаходження точки для стнадартного переміщення(патрулювання), блюпринт дестанції атакування, блюпринт респавна,а також створити Blackboard і Behavior tree.

В Behavior tree ми задаємо функці поведінки ворогу,в нашому випадку це знаходження та переміщення в довільну точку в рамках радіусу патрулювання,час який на точці він проведе після чого обновиться процес переміщення в іншу точку.І також логіка провірки перед собою через конус гравця,якщо гравець є то спрацює логіка перміщення за гравцем на більшій швидкості яку ми задаємо в Анімаційному блюпрінті який ми редагуємо як і на нашого героя, з помаркою на те що ми робимо додатковий анімаційний блюпринт на швидкість переміщення, де вказуємо стадії переміщення(коефіцієнтом швидкості) з анімаціями.Blackboard потрібен для того щоб створити там перемінні по нашому Behavior Tree,які ми будемо використовувати в інших блюпринтах описаних вище.

Також створимо для завдань на знаходження хочаб один об’єкт. В ньому вкажимо медель з материалом а також зробимо блюпринт скрипт по взємодії та окремий блюпринт по видимості з нашим об’єктом при вході в доступний радіус.Далі данний об’єкт можемо включати в завдання по пошуку.

2.3 Вибір засобів розробки гри та планування.

При аналізі індустрії розробкі комп'ютерніх ігор ми обралі компанію Epic Games і її ігровий рушій Unreal Engine 4. Перша його версія з'явилася в 1998 році, але навіть через 20 з гаком років свого існування UE займає одну з лідируючих позицій серед ігрових движків. Спочатку розробники використовували його для випуску шутера Unreal, але потім вирішили заробляти і на ліцензування самого движка. З моменту появи на ньому було створено понад сотні проектів.

Unreal Engine розвивався все більше і більше з кожним роком: міняв версії, наповнювався сучасними технологіями. Кожна з версій радувала новими графічними ефектами. Завдяки доступності придбання і простоті використання, цей движок вибирають і низькобюджетні інді-студії, і найбільші компанії, що створюють світові AAA-блокбастери.З тих пір завдякі йому з'явілась не одна Сотни ігор а найвідомішімі є: Deux Ex, Lineage II, Postal 2, серія ігор Splinter Cell і Unreal Tournament від самих Epic Games. Сам ігровій двіжок був направленій в першу чергу на жанрі «Шутер» від першого лиця, але згодом і з'явілісь функції для інших жанрів вчасності і для нашого «RPG». А завдякі тому що він підтрімує virtual reality (Віртуальну реальність) піднімається до топу найновішіх розробок. Особліво найновішій з них (Unreal Engine 4) з Яким ми і будем працюваті.

Станом на 2020 рік Unreal Engine (вже в четвертій номерний версії) залишається надзвичайно затребуваним ігровим рушієм, дуже гнучким, зручним у використанні і абсолютно різноплановим - поставте поруч гранично похмуру RTS Battlefleet Gothic: Armada і веселу «Королівську битву» Fortnite, і ви ні за не здогадаєтеся, що обидві гри виконані на одному двигуні. Epic Games продовжує

регулярно оновлювати його, використовуючи, в тому числі, і для своїх проектів, включаючи вищезгадану мегапопулярну Fortnite.

Ще однією особлівістю цього ігрового рушія є те що він напісаній на мові C ++. Завдякі цьому можна створюваті ігри на більшість операційніх систем таких як Microsoft Windows, Linux, Mac OS и MAC OS X. Це ж стосується консолей: Xbox, Xbox 360, PlayStation 2, PlayStation Portable.Якщо ж ви бажаєте зіграті з друзями це не є проблемою . Для цього вікорістовується технології WindowsLive, XboxLive и GameSpy.Тепер зупінімось на ігровому рушію Unreal Engine 4. Це ігровий рушій для Створення, запуску, налаштування та управлінням вісокоякіснімі іграмі. Віпустілі його 19 березня 2014 року. Спочатку щоб отріматі повній доступ до його головного кодування потрібно було б заплатіті 19 доларів на місяць за підпіску. Але це продовжувалось не довго. І через рік цей двіжок ставши безплатнім, але стягував 5% від продажу всіх матеріалів яку булі розроблені на ньом. На мою думку це є чудовім рішенням адже тепер це вігідно не тільки для великий компаній але й для меншіх стартапів або для безкоштовна ігор. На даній момент він спеціалізується на створені 2D і 3D ігор, архітекурніх візуальзацій та демок.Особливості Unreal Engine 4:

– Повний вихідний код на C++.

– Надійна розрахована на багато користувачів платформа.

– Гнучкий редактор матеріалів.

– Редактор VR

– Високий рівень Штучного Інтелекту

Його системні вимоги це:

Ноутбук, ПК або Mac.

Windows 7 64-bit або Mac OS X 10.9.2 і версії які йдуть далі.

Чотирьохядерний процесор Intel або AMD, 2.5 GHz або щось краще.

Nvidia GeForse 470 GTX або AMD Radeon 6870 HD або щось краще.

4 Гб озу або більше.

Це налаштування для мінімальних вимог до системи. При створенні проекту(шо б це не було в віртуаьного чату до гри) вам допоможе гнучка система

створення та редагування. Також ти можеш використовувати такі модулі як системи візуалізації :(Microsoft Diekt X, Open GL та Pixomaic), відтворення звуку:(EAX,DirectSound3D) а також засоби голосового відтворення тексту і розпізнавання голосу(мовлення).

Є два види реалізація коду це - Blueprint і C++.

В мому випадку я використовував Blueprint, тому що це швидкий і гнучкий спосіб розробки коду для нашого проекту. Це дозволяє в мінімальний обсяг часу запустити робочий процес і приступити до розробки коду. Ціль Blueprint - дати розробникам можливість оптимізувати процес налаштування розробки коду, щоб вони могли витрачати більше часу на втілення своїх ідей в запланованих проектах. Blueprint є інструментом для запуску і налаштування контенту в проекті.

Blueprint - це система візуального скриптинга. Вона дозволяє реалізувати свої задумки в спрощеному режимі. Наприклад, без знань програмування можна створити цикл дня і ночі або змусити оточення реагувати на переміщення гравця.

Велика частина Fortnite (відомої гри Epic Games) реалізована через Blueprint. І це не якийсь рекламний хід для просування візуального програмування. Просто це оптимізує роботу.

Коли у креативної команди виникає ідея, її реалізацію потрібно побачити якомога раніше без залучення програмістів. Це не означає, що при роботі з UE4 відпадає потреба в VFX-художників або програмістів. Просто тепер можливості звичайного художника розширилися, і можна відразу наочно представити свою ідею. Попередня візуалізація за допомогою движка значно прискорює роботу команди в цілому.

Цінна можливість Unreal Engine - швидке створення прототипу гри. У прототипі особливо важливо відразу передати атмосферность. Для цієї мети UE пропонує безліч інструментів - від симуляції природних явищ через систему частинок до постпроцесингу.

Так, сніг можна швидко додати через Particle System. Для цього потрібно просто створити еммітер (компонент системи, що генерує частинки) і

налаштувати його під ваші потреби: пустити білі частинки і підігнати параметри їх прольоту.

А інструменти постобробки допоможуть надати зображенні цікавий вид. Потрібно просто покрутити значення відповідних параметрів: контрасту, колірного тону або аберацій.

У більшості випадків навіть не доведеться шукати відеоуроків, так як всі параметри очевидно названі в інтерфейсі, і потрібний ефект можна підібрати інтуїтивно.

Epic Games надають свій движок на безкоштовній основі. Тільки якщо ви почнете поширювати свій продукт, і прибуток перевищить 1 мільйон доларів, потрібно буде заплатити 5% роялті.

Як кажуть самі Epic Game: «Ваш успіх - це наш успіх». Крім постійного розвитку свого безкоштовного движка, компанія вкладається в побудову ком'юніті. Це включає додаткові матеріали, лекції, конференції, а також - видачу грантів.

Якщо у вас є концепція гри, ви можете подати свій прототип на конкурс і отримати грант на розробку. Творці UE надають початківцям як матеріальну допомогу, так і нематеріальну - у вигляді консультацій, знижок і пільг. Все це робиться з метою заохочення креативних рішень.

Також, близько року тому Epic Games у співпраці з Quixel розширили бібліотеку Megascan і зробили її безкоштовною при використанні в зв'язці з Unreal Engine 4. Це ціла бібліотека безкоштовних високодеталізованих сканів з дозволом від 2 до 8К, інтегрованих в движок.

У художників з'явилися готові базові елементи оточення на кшталт каменів, повалених дерев, землі, трави або різного роду малих пропсов. Це дозволяє прискорити і спростити презентацію свого Ассет, а також процеси створення ігрового рівня або архітектурної візуалізації.

Якщо ж знадобиться щось специфічне - на торгових майданчиках завжди можна знайти елементи коду, блупрінти і окремі Ассет-паки.

Перевагами Unreal Engine останньої версії є неймовірна гнучкість і універсальність. На відміну від свого конкурента Unity, що вимагає установки цілого ряду плагінів, він вже оснащений всім необхідним. Движок написаний на C ++, тому користувачі, знайомі з цією мовою програмування, освоять його швидше. Втім, навіть без знання C ++ розібратися з UE4 не складе особливих труднощів. Така доступність досягається завдяки редактору Blueprints, який допомагає створювати скрипти і розміщувати об'єкти без написання коду.

В Unreal Engine є всі інструменти, необхідні дизайнерам, програмістам і артистам. Багато з них полегшують роботу фахівців. Наприклад, підтримують різні формати текстур, неймовірно повно відтворюють фізичні властивості матеріалів, дають можливість змінювати об'єкти в режимі реального часу, дозволяють вибирати джерела світла, додавати ефекти і ін. Багату колекцію Ассет надалі можна використовувати для розробки інших проектів, а відкритий вихідний код дозволяє вносити в движок необхідні зміни. Він з легкістю портується на будь-яку платформу, адаптуючи гри під ПК, консолі і мобільні гаджети.

Ще один очевидний плюс дітища Epic Games - мультиплеерная гра «з коробки». Після запуску редактора всього декількома кліками можна створити проект з уже готовою мережевою грою. Якщо враховувати зростання багатокористувацьких ігор і націленість великих компаній на онлайн-режими з декількома гравцями, то це досить вагома перевага, якого, наприклад, у того ж Unity немає.

UE надає розробникам вільний доступ до всього функціоналу ігрового рушіява. Є, правда, ряд умов і роялті, але все інструменти можна використовувати відразу в повному обсязі.

За допомогою UE4 можна робити відеоігри практично для всіх ОС, мобільних платформ і консолей. Завдяки підтримці різноманітних систем відтворення графіки, відтворення звуку і можливості для мережевої гри, движок відмінно підходить для створення різних жанрів відеоігор, в тому числі і для MMORPG. Він ідеальний для тривимірних AAA-ігор, перш за все, для екшенів і

шутерів. Список проектів UE різноманітний, але домінують в ньому саме ці жанри.

Epic Games є лідером в області сучасних технологій, тому 4-я версія UE наповнена ними по максимуму. При наявності певного рівня майстерності у розробників вона забезпечує вражаюче візуальне оформлення гри з якісним освітленням, м'якими тінями, реалістичною анімацією і іншими ефектами. Завдяки всьому цьому, движок активно використовується і для комп'ютерної графіки в кіно.

Як вже говорилося, Unreal Engine дуже гнучкий і універсальний. На відміну від Unity, який вимагає установки безлічі плагінів (часто - платних), UE4 вже «з коробки» забезпечений всіма необхідними інструментами розробки. Користувачі, які знають C ++, освоять движок швидше, адже він використовує саме цю мову програмування. Втім, для тих, хто не знайомий з C ++, Unreal Engine теж піддасться без проблем - все завдяки візуальному редактору Blueprints, який дозволяє створювати скрипти і розміщувати об'єкти, не написавши жодного рядка коду.

До того ж, движок містить безліч інструментів, які полегшують роботу з ним. Наприклад, підтримує безліч форматів текстур, точно передає фізичні властивості матеріалів, дозволяє змінювати об'єкти в реальному часі, задавати для них функції і коментарі, автоматично вибрати джерела освітлення, додати туман і інші ефекти, і так далі. Велику колекцію Ассет (платних і безкоштовних) можна використовувати при розробці ігор, а відкритий вихідний код движка дає можливість вносити в нього зміни при необхідності. Движок гнучко підлаштовується під платформу розробки, що дозволяє оптимізувати гри під консолі, мобільні гаджети та ПК.

Epic Games стабільно тримає лідерські позиції в області передових технологій, тому Unreal Engine 4 напханий ними під зав'язку, забезпечуючи, при належному майстерності ігроробів, вражаючу візуальну складову з якісним освітленням (трасування променів в реальному часі, звичайно ж, підтримується), м'якими тінями, чесними відображеннями, достовірної анімацією персонажів і

іншими ефектами. За рахунок цього, до речі, движок також використовується в створенні комп'ютерної графіки в кіноіндустрії - наприклад, його силами було створено дроїд K-2SO для «ізгоя-1» сцени з «У пошуках Дорі» і «Мандалорца».

Здавалося б, у такого багатого можливостями движка не може бути недоліків. Проте знайти їх можна. Так, наприклад, на Unreal Engine 4 проблематично створювати великі безшовні світи, розраховані на безліч гравців, що робить скрутним розробку MMORPG і інших MMO-ігор на движку.

Аналогічні проблеми виникають з II: якщо додати на локацію занадто багато II-істот, спроби движка обробити поведінку всіх одночасно викличуть падіння fps, тому розробникам доведеться придумувати способи обмеження діяльності монстрів, які перебувають за межами взаємодії від гравців.

Зручність використання - не першорядне, але досить важливий параметр движка. І в цьому плані UE4 програє все того ж Unity: якщо другий розрахований на ігроробів-новачків, то перший розроблений для професіоналів, що позначилося на ергономічності інтерфейсу.

Нарешті, Unreal Engine змушує більше працювати над оптимізацією ігор. Деякі проекти, зокрема, PUBG, страждають від слабкої продуктивності навіть на потужних ігрових ПК. Це, в першу чергу, «заслуга» програмістів шутера, проте багато розробників говорять про вимогливості движка. Думки з цього приводу різняться, але факт залишається фактом - необхідно ретельно працювати над іграми на UE, щоб домогтися плавності картинки.

На даний момент Unreal Engine залишається одним з найпопулярніших ігрових рушіїв серед ігроробів, універсальним, гнучким і комфортним у використанні. Він ідеально підходить для команди професіоналів, в руках якої здатний створити воістину неймовірну картинку.

3 Програмна реалізація

3.1 Структура та функціональне призначення модулів програми,їх взаємозв’язок.

Взаємодія системних модулів моєї гри дуже тісно пов’язана одна з одною. Я розположив на карті нейтральних не ігрових персонажів в яких можна брати завдання знаходячи їх при пересувані по карті за допомогою системи управаління персонажем гравця. Так як є завдання як на пошук втрачених предметів так і знищення ворожих не ігрових персонажів то далі вступає в силу механіка бою та взаємодії з сюжетними предметами. Механіка бою передбачає в собі патрулювання не ігрових персонажів територію області в якій вони появляються, після чого якщо гравець попаде в їх поле зору вони почнуть слідувати за персонажем гравця для того щоб його атакувати. Гравець ж можемо атакувати не ігрового персонажа навіть тоді коли він не попав в поле зору ворожого не ігрового персонажу. Після перемоги над ворожими не ігровоми персонажами ми отримаємо деяку кількість досвіду які використовуються щоб отримати настпуний рівень нашого персонажа. Сама система бою також частина керування персонажем тому це входить в його постійну механіку, ми можемо вокористовувати удар навіть просто так, так як система в нас Non-target.Non-target - це тип бойової системи в онлайн-іграх, переважно в mmorpg. Суть - ви не можете «захопити» ціль і поставити атаку на автомат. У мморпг з нон таргет потрібно цілитися, щоб потрапити в рухому мішень, а також пересуватися і ухилятися під час бою. Target - англійською означає "Мета". У мморпг таргет означає вибір персонажа, на якого будуть використовуватися здатності персонажа.Система пошуку діє так що скріпт нашого предмета з яким ми можемо взаємодіяти, розташувавши його там де нам потрібно, і після чого прописавши його в шаблоні завдання, який ми поставимо на наших не ігрових персонажах, ми получим змогу його підняти чим виконаємо завдання.

3.2 Розробка програмних модулів.

Для початку створемо клас Actor. Напевно Actor,це самий часто використовуваний клас в іграх. Actor - це основа для будь-якого об'єкта на рівні, в тому числі для гравців, керованих II ворогів, дверей, стін і геймплейних об'єктів. Актори створюються за допомогою таких ActorComponents (див. Наступний розділ), як StaticMeshComponent, CharacterMovementComponent, ParticleComponent і багатьох інших. Навіть такі класи, як GameMode (див. Нижче) є акторами (хоча у GameMode немає «реального» положення в світі). Давайте обговоримо пару аспектів, які вам потрібно знати про акторів.

Actor - це клас, який можна реплицировать по мережі (для багато режиму). Це легко робиться за допомогою виклику в конструкторі SetReplicates (true). Для створення ефективного мережевого програмування акторів необхідно враховувати безліч аспектів, які я не зможу розглянути в цій статті.

Актори підтримують концепцію отримання шкоди. Втрати може наноситися безпосередньо актору за допомогою MyActor-> TakeDamage (...) або через UGameplayStatics :: ApplyDamage (...). Варто врахувати, що існують варіації: PointDamage (наприклад, для зброї, влучення з якого обчислюється трасуванням променя (hitscan)) і RadialDamage (наприклад, для вибухів). На офіційному сайті Unreal Engine є чудова вступна стаття Damage in UE4.

Створити новий екземпляр актора в коді можна просто за допомогою GetWorld () -> SpawnActor <T> (...); де T - це повертається клас, наприклад, AActor для одного з ваших власних класів - AGadgetActor, AGameplayProp і т.д.

Існує безліч способів отримання доступу до акторам. Зазвичай у вас буде покажчик / посилання на конкретний актор, який вам потрібен. У показаному вище прикладі ми зберігаємо покажчик на актор надівається предмета в змінної і починаємо через неї маніпулювати екземпляром актора.

Дуже корисною функцією, яку можна використовувати при прототіпірованії або освоєнні движка, є UGameplayStatics :: GetAllActorsOfClass (...). Вона дозволяє нам отримає масив з усіх акторів переданого класу (в тому числі і породжених

класів; якщо передати в якості класу Actor, то ми отримаємо ВСЕ об'єкти рівня). Ця функція часто бояться і уникають що не дуже ефективного способу взаємодії з оточенням, але іноді це єдиний доступний інструмент.

Актори не мають власних перенесення, повороту або масштабу. Все це задається і виходить за допомогою RootComponent, тобто компонента верхнього рівня в ієрархії SceneComponents (докладніше про SceneComponents розповідається нижче). Найбільш часто використовувані функції на зразок MyActor-> GetActorLocation () насправді переходять до RootComponent і повертають його розташування в світі.

Ось ще кілька корисних функцій, які використовуються в контексті актора:

BeginPlay // «Перша» функція, що викликається після створення і повної ініціалізації актора. Це зручне місце для завдання базової логіки, таймера і внесення змін до властивості, тому що актор вже повністю инициализирован і може виконувати запити до свого оточення.

Tick // Викликається в кожному кадрі. Для більшості акторів можна відключити її з міркувань продуктивності, але за замовчуванням вона включена. Чудово підходить для швидкого налаштування динамічної логіки і перевірки умов в кожному кадрі. Поступово ви почнете переміщати все більше коду пов'язаної з подіями логіки з таймерів в логіку, яка працює на менших частотах.

EndPlay // Викликається, коли актор видаляється зі світу. Містить «EEndPlayReason», в якому вказується причина виклику.

GetComponentByClass // Знаходить один екземпляр компонента певного класу. Дуже корисно, коли ви не знаєте точного типу актора, але знаєте, що він повинен містити певний тип компонента. Також існує GetComponentsByClass, що повертає всі екземпляри класу, а не тільки перший знайдений.

GetActorLocation // І все його варіації - * Rotation, * Scale, в тому числі і SetActorLocation і т.д.

NotifyActorBeginOverlap // Зручна для перевірки накладень, викликаних будь-яким з його компонентів. Таким способом можна швидко налаштовувати геймплейні тригери.

GetOverlappingActors // Знаходить, які інші актори перетинаються з обраним. Існує також варіант для компонентів: GetOverlappingComponents

Actor містить величезний функціонал і безліч змінних - він є фундаментом основи геймплея в Unreal Engine, тому це не дивно. Для подальшого дослідження цього класу непогано буде відкрити файл заголовка Actor.h в Visual Studio і подивитися, який функціонал в ньому є. У статті ж нам належить ще багато розглянути, тому давайте перейдемо до наступного класу в списку.

ActorComponent

Компоненти розташовуються усередині акторів, стандартними компонентами вважаються StaticMeshComponent, CharacterMovementComponent, CameraComponent і SphereComponent. Кожен з цих компонентів обробляє свою приватну задачу, наприклад, рух, фізична взаємодія (наприклад, обсяг колізії для чіткої перевірки взаємодіючих акторів) або візуально відображає щось в світі, наприклад, меш гравця.

Підкласом цього компонента є SceneComponent - це базовий клас для всього, пов'язаного з Transform (Position, Rotation, Scale), що підтримує прикріплення. Наприклад, ми можемо прикріпити CameraComponent до SpringArmComponent для настройки камери від третьої особи. Для правильного налаштування відносного розташування потрібні і transform, і прикріплення.

Найчастіше компоненти створюються в конструкторі актора, але також можна створювати і знищувати їх в процесі виконання. Для початку давайте розглянемо один з конструкторів мого актора.

USkeletalMeshComponent створюється за допомогою CreateDefaultSubobject <T> (функції актора) і вимагає вказівки імені (це ім'я можна побачити в списку компонентів блюпрінта). Якщо ви пишете код гри на C ++, то часто будете використовувати цю функцію, але ТІЛЬКИ всередині контексту конструктора.

Можна також зауважити, що ми задаємо MeshComp в якості нового RootComponent. Тепер все Scene Component повинні прикріплятися до цього Мішу, що можна легко зробити за допомогою наступного рядка:

WidgetComp = CreateDefaultSubobject <UWidgetComponent> (TEXT ("InteractWidgetComp"));WidgetComp-> SetupAttachment (MeshComp);

SetupAttachment займеться обробкою вихідного прикріплення; очікується, що він буде викликатися в конструкторі для ВСІХ компонентів сцени, крім самого RootComponent. Можна задатися питанням, чому мій ItemComponent не викликає цю функцію SetupAttachment. Так вийшло просто тому, що цей компонент є ActorComponent, але НЕ SceneComponent і не має Transform (позиції, повороту, масштабу), а тому не повинен додаватися в ієрархію. Проте, компонент все одно буде регистировать з Actor. Те, що він відділений від ієрархії, означає, що функції на зразок MyActor-> GetComponentByClass повертатимуть все ActorComponents і SceneComponents.

Поряд з Actor, ці компоненти критично важливі для створення гри як на C ++, так і на блюпрінтах. Саме вони є будівельними цеглинками гри. Ви запросто можете створити власні компоненти, щоб вони обробляли якісь специфічні аспекти гри, наприклад HealthComponent, який зберігає окуляри здоров'я і реагує на втрату, одержуваний його батьківським актором.

За допомогою представленого нижче коду можна створювати в процесі виконання власні компоненти. Це відрізняється від поведінки CreateDefaultSubobject, використовуваного тільки для конструкторів.

Ось частина корисного функціоналу ActorComponents:

TickComponent () // Як і Tick () актора, виконується кожен кадр для обробки високочастотної логіки.

bool bIsActive і пов'язані з нею функції на зразок Activate, Deactivate, ... Використовуються для повного включення / відключення компонента (в тому числі і TickComponent) без знищення компонента і видалення його з актора.

Щоб забезпечити реплікацію ActorComponent, необхідно викликати функцію SetIsReplicated (true), назва якої злегка відрізняється від функції актора. Це необхідно тільки тоді, коли вам потрібно реплицировать конкретну частину логіки компонента, наприклад змінну при викликах функції, тобто реплицировать потрібно не всі компоненти реплицируемой актора.

PlayerController

Це базовий клас для гравця, який отримує введення від користувача. Сам по собі PlayerController не відображається візуально в оточенні, замість цього він керує екземпляром Pawn, визначальним візуальне і фізичне уявлення цього гравця в світі. Під час ігрового процесу гравець може володіти декількома різними Pawn (наприклад, транспортним засобом або свіжої копією Pawn при респауне), а екземпляр PlayerController залишається однаковим протягом усього рівня. Це важливо, тому що в деякі моменти PlayerController може не володіти взагалі ніякими Pawn. Це означає, що такі речі, як відкриття меню повинні додаватися до PlayerController, а не до класу Pawn.

У багатокористувацьких іграх PlayerController існує тільки на володіє їм клієнта і на сервері. Це означає, що в грі на 4 гравця у сервера є 4 контролера гравців, а у кожного клієнта - тільки по одному. Це дуже важливо розуміти, коли необхідно використовувати змінні; якщо для всіх гравців потрібно реплікація змінної гравця, то вона повинна існувати не в PlayerController, а в Pawn або навіть в PlayerState (розглянутому нижче).

Отримання доступу до PlayerControllers

GetWorld () -> GetPlayerControllerIterator () // GetWorld доступний в будь-якому екземплярі Actor

PlayerState-> GetOwner () // власник playerstate має тип PlayerController, і ви повинні передавати його PlayerController самостійно.

Pawn-> GetController () // Задається лише тоді, коли пауні вже володіє (тобто управляє) PlayerController.

Цей клас містить PlayerCameraManager, який обробляє view targets і transforms камери, в тому числі і тряску. Ще одним важливим класом, яким керує PlayerController, є HUD (розглянуто нижче). Він застосовується для рендеринга на Canvas (тепер використовується не так часто, тому що є UMG) і його можна використовувати для управління даними, які необхідно передати в інтерфейс UMG.

Коли до GameMode підключається новий гравець, для цього гравця в класі GameModeBase за допомогою Login () створюється PlayerController.

Pawn - є фізичним і візуальним представленням того, чим керує гравець (або II). Це може бути машина, воїн, вежа або що завгодно, що позначає персонажа гри. Стандартним подклассом Pawn є Character, в якому реалізується SkeletalMesh і, що більш важливо, CharacterMovementComponent з безліччю опцій для точної настройки руху гравця по оточенню за допомогою звичайного шутерний руху.

У багатокористувацьких іграх кожен екземпляр Pawn реплицируется іншим клієнтам. Це означає, що в грі на 4 гравця і на сервері, і на кожному клієнті є 4 примірники pawn. Досить часто екземпляр Pawn «вбивають» при смерті гравця, а при респауне створюється новий екземпляр. Майте це на увазі при зберіганні даних, які потрібно буде відновити після завершення життя гравця (або повністю відмовтеся від цього патерну і постійно залишайте екземпляр pawn живим)

Отримання доступу до Pawn

PlayerController-> GetPawn () // Тільки коли PlayerController володіє Pawn

GetWorld () -> GetPawnIterator () // GetWorld доступний для будь-якого примірника Actor і повертає ВСЕ Pawn, в тому числі і для II.

GameModeBase створює Pawn за допомогою SpawnDefaultPawnAtTransform. Клас GameModeBase також визначає, який клас Pawn потрібно створювати.

GameModeBase

Базовий клас, який визначає використовувані класи (PlayerController, Pawn, HUD, GameState, PlayerState). Часто використовується для завдання правил гри в таких режимах, як «Capture the Flag»; він може обробляти прапори або хвилі ворогів. Обробляє та інші важливі функції, такі як створення гравця.

GameMode - це підклас GameModeBase. Він містить ще кілька функцій, які спочатку використовувалися в Unreal Tournament, такі як MatchState і інші шутерний функції.

У многопользовательском режимі клас GameMode існує тільки на сервері! Це означає, що ні у одного клієнта немає його примірника. В одного користувача іграх він не має ніякого впливу. Для реплицирования функцій і зберігання даних, необхідних для GameMode, можна використовувати GameState, існуючий на всіх клієнтах і створений спеціально для цієї мети.

Отримання доступу до GameMode

GetWorld () -> GetAuthGameMode () // GetWorld доступний для будь-якого примірника Actor.

GetGameState () // повертає gamestate для реплікації функцій і / або змінних

InitGame (...) // ініціалізує деякі з правил гри, в тому числі, зазначені в URL (наприклад, «MyMap? MaxPlayersPerTeam = 2»), які можна передавати при завантаженні рівнів в грі.

HUD - це клас інтерфейсу користувача. У ньому міститься багато коду Canvas, який є кодом відтворення інтерфейсу користувача, написаного до появи UMG. Сьогодні основною роботою по відображенні інтерфейсу користувача займається UMG.

Клас існує тільки в клієнті. Реплікація неможлива. Їм володіє PlayerController.

Отримання доступу до HUD

PlayerController-> GetHUD () // Доступний в локальному PlayerController.

Створюється за допомогою SpawnDefaultHUD (створює звичайний AHUD) всередині PlayerController, який володіє HUD, а потім переопределяется GameModeBase за допомогою InitializeHUDForPlayer класом HUD, зазначеним в GameModeBase.

UWorld - це об'єкт верхнього рівня, що представляє карту, на якій будуть існувати і рендери актори і компоненти. Містить постійний рівень і багато інших об'єктів, такі як gamestate, gamemode, а також списки знаходяться на мапі Pawns і Controllers.

Трасування ліній і все її варіації виконуються через World за допомогою таких функцій, як World-> LineTraceSingleByChannel і багатьох інших подібних варіацій.

Отримання доступу до World

Для отримання доступу досить викликати GetWorld () всередині акторів.

Коли необхідно отримати примірник World в статичних функціях, то потрібно передавати WorldContextObject, по суті є словом для будь-якого актора,

якого можна використовувати для виклику -> GetWorld (). Ось приклад з одного мого файлу заголовка:

static APlayerController * GetFirstLocalPlayerController (UObject * WorldContextObject);

GameInstance

GameInstance має один екземпляр, який продовжує існувати протягом тривалості всієї гри. При переходах між картами і меню буде зберігатися один і той же екземпляр цього класу. Цей клас можна використовувати для створення обробників подій або обробки мережевих помилок, завантаження таких даних користувача, як параметри гри і функцій, які відносяться не тільки до одного рівня гри.

Отримання доступу до GameInstance

GetWorld () -> GetGameInstance <T> (); // де T - тип класу, наприклад GetGameInstance <UGameInstance> () або вас власний породжений тип.

Actor-> GetGameInstance ()

PlayerState - це контейнер для змінних, реплицируемой між клієнтом / сервером для окремого гравця. У багатокористувацьких іграх він не призначений для виконання логіки і є просто контейнером даних, оскільки PlayerController недоступний для всіх клієнтів, а Pawn часто знищується при смерті гравця, тому непридатний для даних, які повинні зберігатися після смерті.

Отримання доступу до PlayerState

Pawn містить його як змінну Pawn-> PlayerState, також доступну в Controller-> PlayerState. PlayerState в Pawn призначений тільки тоді, коли Pawn володіє Controller, в іншому випадку має значення nullptr.

Список всіх наявних примірників PlayerState (наприклад, всіх гравців, що знаходяться в матчі) можна отримати через GameState-> PlayerArray.

Створює клас призначається в GameMode (PlayerStateClass) і створюється в AController :: InitPlayerState ()

GameStateBase - це схожий на PlayerState, але надає клієнтам інформацію про GameMode. Так як екземпляр GameMode існує не в клієнтах, а тільки на сервері,

цей клас є корисним контейнером для реплікації інформації, наприклад, про завершення часу матчу, очок команди і т.д.

Має дві варіації - GameState і GameStateBase. GameState обробляє додаткові змінні, необхідні GameMode (на відміну від GameModeBase)

Отримання доступу до GameStateBase

World-> GetGameState <T> () // де T - викликається клас, наприклад GetGameState <AGameState> ()

MyGameMode-> GetGameState () // зберігається і доступний в екземплярі gamemode (необхідний тільки на сервері, який володіє єдиним екземпляром GameMode); клієнти повинні використовувати вказаний вище виклик.

Використовуйте GameStateBase замість GameState, тільки якщо gamemode не успадковується від GameMode замість GameModeBase.

UObject

Базовий об'єкт практично для всього в движку. З UObject успадковуються актори, а також інші базові класи, такі як GameInstance. Він ніколи не повинен використовуватися для візуалізації, але дуже корисний для зберігання даних і функції, коли під ваші вимоги не підходять struct.

UObjects Не Спаун подібно акторам, а створюються за допомогою NewObject <T> (). наприклад:

TSubclassOf <UObject> ClassToCreate;

UObject * NewDesc = NewObject <UObject> (this, ClassToCreate);

GameplayStatics – це статичні класи використовуються для обробки різного стандартного функціоналу ігор, наприклад, відтворення звуків і створення ефектів частинок, створення акторів, застосування шкоди до акторам, отримання Pawn гравця, PlayerController і т.д. Цей клас дуже корисний для всілякого доступу до геймплейні функцій. Всі функції є статичними, тобто вам не потрібно покажчик на екземпляр цього класу і ви можете викликати функції безпосередньо з будь-якого місця, як це показано в прикладі нижче.

Отримання доступу до GameplayStatics

Так як GameplayStatics є UBlueprintFunctionLibrary, ви можете отримати до нього доступ з будь-якого місця коду (або блюпрінта)

UGameplayStatics :: WhateverFunction (); // static functions are easily accessed anywhere, just include #include "Kismet / GameplayStatics.h"

Далі мною було створенно інтерфейс користувача (user interface, UI).

UI в Unreal Engine 4 створюється за допомогою Unreal Motion Graphics (UMG). UMG дозволяє зручно вибудовувати UI, перетягуючи елементи UI, такі як кнопки і текстові мітки. Щоб його створити, нам необхідні віджети.

Віджет (widget) - це елемент UI, що надає UI візуальні функції. Наприклад, віджет Button надає об'єкт, який користувач може бачити і натискати на нього.

Сам віджет необов'язково повинен бути видимим. Наприклад, віджет Grid Panel рівномірно розділяє свій простір між його вмістом. Користувач не може побачити Grid Panel, але бачить його вплив.

Крім того, віджети можуть містити інші віджети.

Можна навіть створити віджет, який є цілим інтерфейсом, наприклад, екраном меню. Всі елементи UI теж є віджетами і містяться всередині віджета початкового екрана.

Отже, ми дізналися, що таке віджети. Тепер можна створити віджет для HUD.

Створення віджета

Щоб створити віджет перейдіть в Content Browser і знайдіть папку UI. Натисніть на кнопку Add New і виберіть User Interface \ Widget Blueprint. Перейменуйте новий Ассет в WBP_HUD.

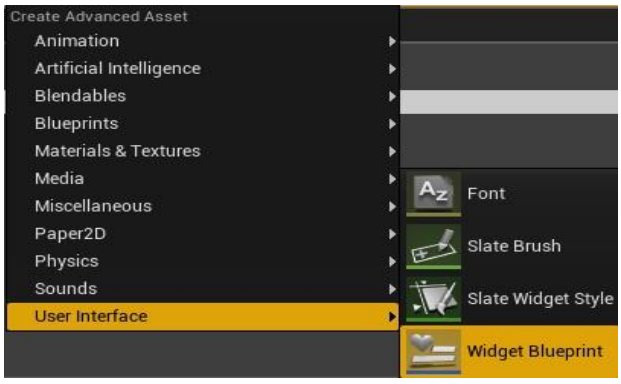

Рисунок 3.1 – Приклад нового Ассета

Двічі натисніть на WBP_HUD, щоб відкрити його в UMG UI Designer.

UMG UI Designer складається з семи основних елементів.

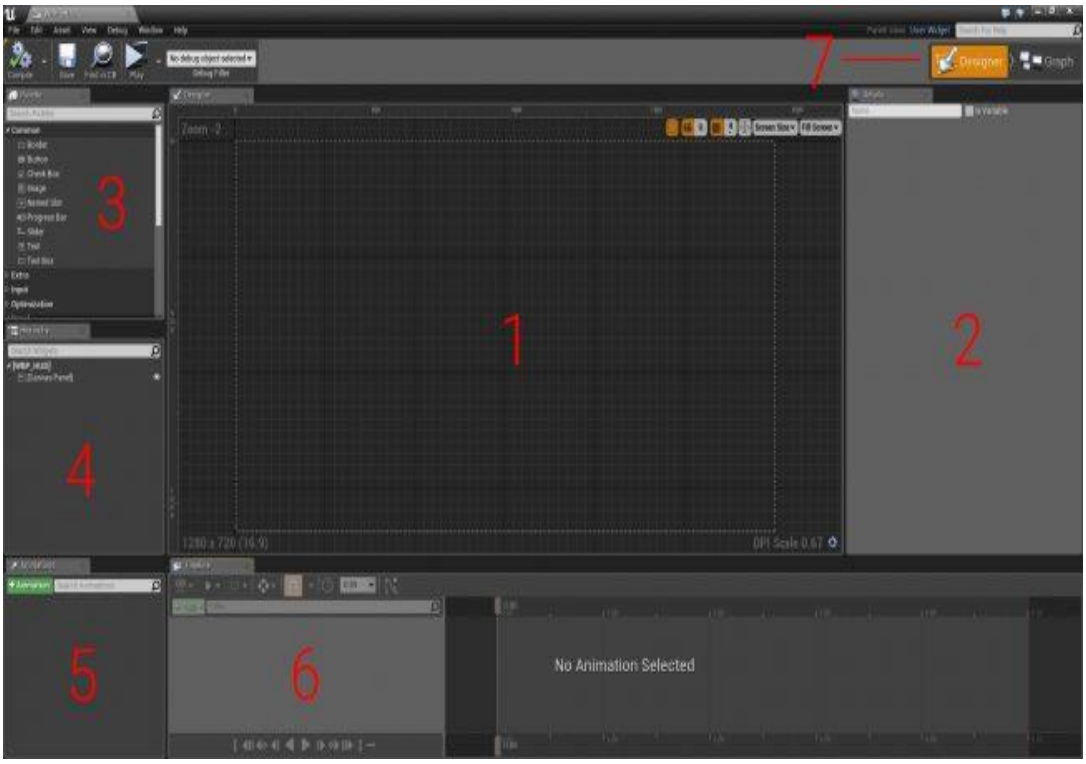

Рисунок 3.2 – Приклад UMG UI Designer

Designer: в цій області представлено візуальне відображення віджета. Переміщатися по ній можна затиснувши праву кнопку миші і рухаючи мишею. Масштабування виконується прокруткою коліщатка миші.

Details: тут відображаються властивості вибраного віджета

Palette: список всіх віджетів, які можна використовувати. Всі створені Вами віджети теж з'являються тут.

Hierarchy: список всіх вже використовуваних віджетів

Animations: деякі властивості віджетів можуть мати анімацію, наприклад, розташування і розмір. У цій панелі перераховані всі анімації.

Timeline: при виборі анімації на цій панелі відображаються анімовані властивості і ключові кадри

Editor Mode: тут можна перемикатися між режимами Designer і Graph. Режим Graph майже аналогічний Event Graph у Blueprint.

Створення віджета Text

Віджети Text відмінно підходять для відображення числової інформації, наприклад, лічильника і таймера.

Перейдіть в панель Palette і знайдіть віджет Text. Додайте віджет, перетягнувши його мишею в панель Designer.

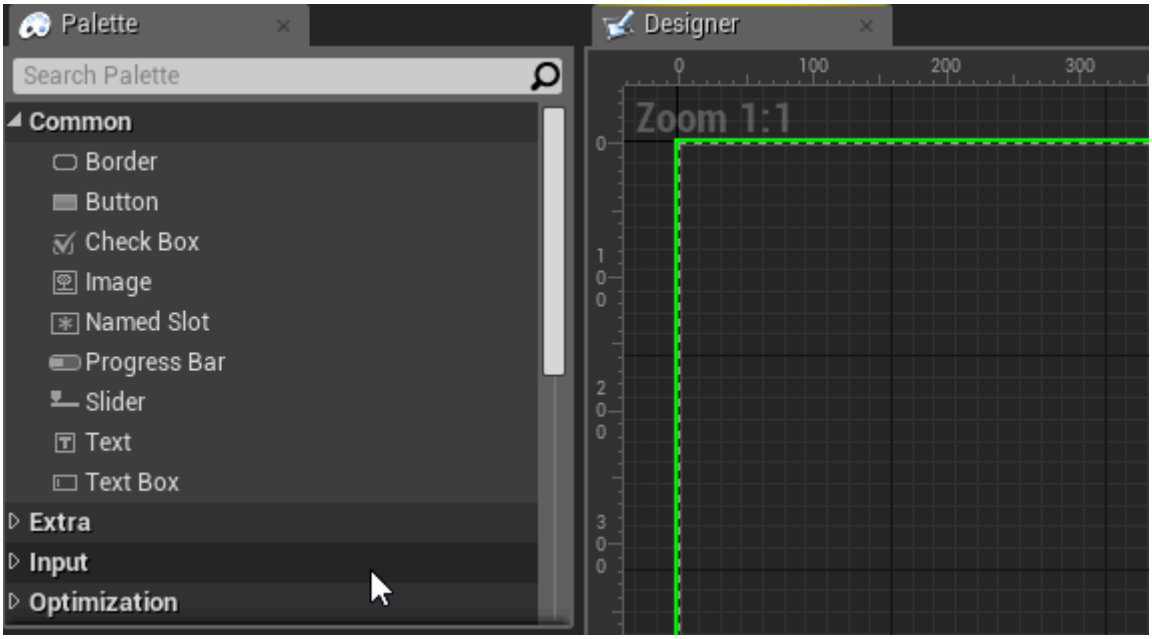

Рисунок 3.3 - Панель Designer

Зміст тексту нас поки не цікавить, ми змінимо його пізніше.

Перейменуйте віджет в CounterText. Це можна зробити, вибравши віджет Text і перейшовши в панель Details. Введіть CounterText в текстове поле у верхній частині.

Рисунок 3.4 - Панель Details

Віджети можна рухати, перетягуючи їх лівою клавішею миші в панелі Designer.

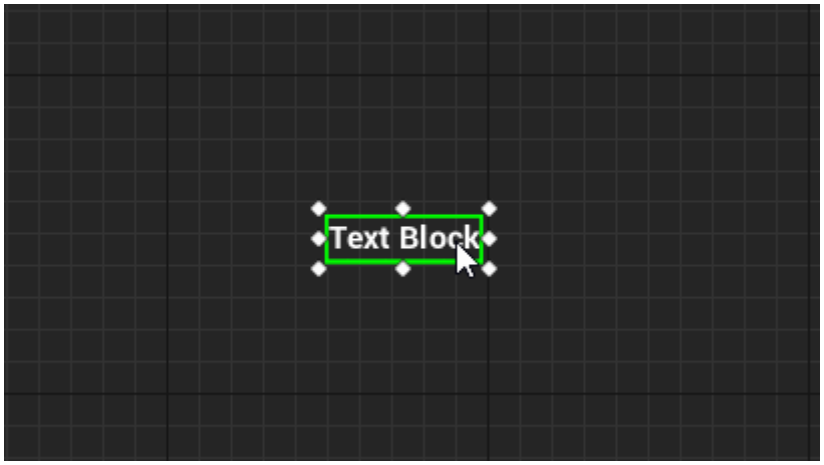

Рисунок 3.5 - Панель Designer

Також можна змінювати розмір віджетів, перетягуючи лівою клавішею миші кордону. Зміна розміру дозволяє задати кордону віджета. Unreal не буде нічого рендерить за межами кордонів.

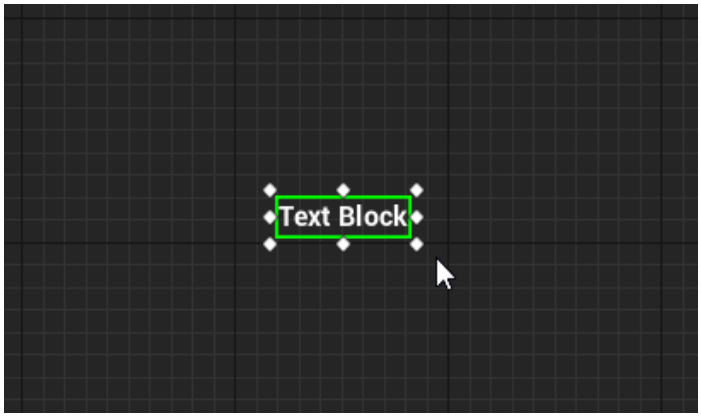

Рисунок 3.6 - Панель Designer з віджетом

Або ж можна задати положення і розмір зміною значень в панелі Details. Задайте наступні властивості і значення для CounterText:

Position X: 200

Position Y: 50

Size X: 500

Size Y: 100

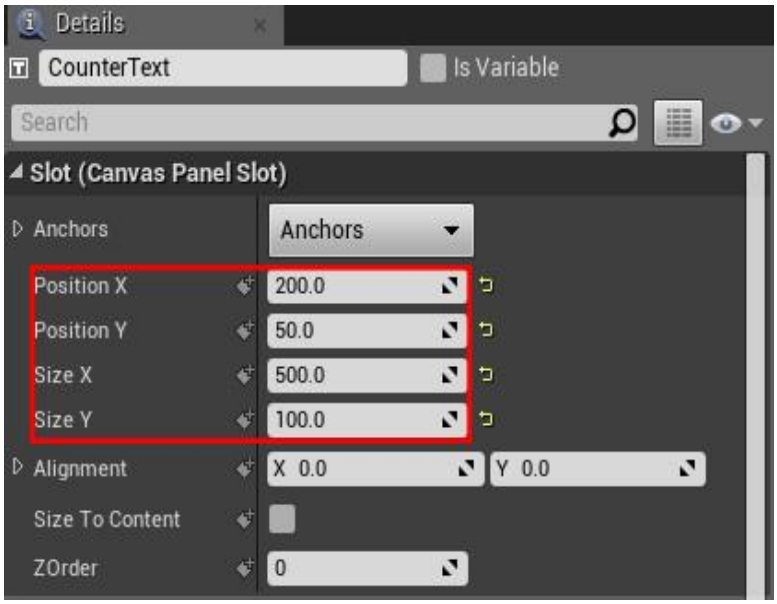

Рисунок 3.7 - Вкладка Details

На даний момент текст займає лише невелику частину поля.

Рисунок 3.8 - Панель Designer

Можна збільшити розмір шрифту, перейшовши в панель Details і до розділу Appearance. Праворуч від властивості Font є текстове поле для завдання розміру шрифту.

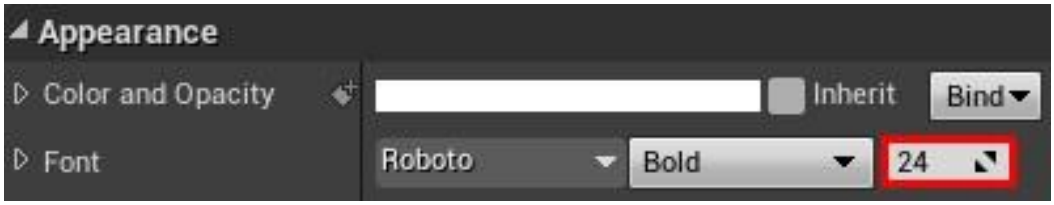

Рисунок 3.9 – Вкладка Appearance

Вводімо розмір 68.

Рисунок 3.10 – Зображення розміру віджета

Давайте зробимо лічильник красивіше, додавши до нього значок.

Створення віджета Image

Віджети Image - це простий спосіб відображення графіки в UI, наприклад, значків.

Створіть віджет Image і назвіть його CounterIcon. Задайте Position X значення 75, а Position Y - значення 50. Віджет розміститься поруч з CounterText.

Рисунок 3.11 - Віджет Image

Щоб задати зображення, перейдіть в панель Details і зайдіть в розділ Appearance. Розгорніть властивість Brush, а потім натисніть на списку поруч з Image. Виберіть T_Counter.

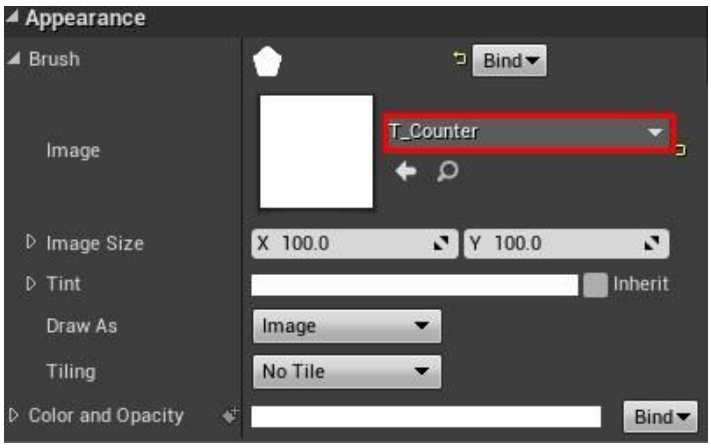

Рисунок 3.12 – Вкладка Appearance

Зображення буде виглядати розтягнутим, тому що розмір віджету відрізняється від розміру зображення.

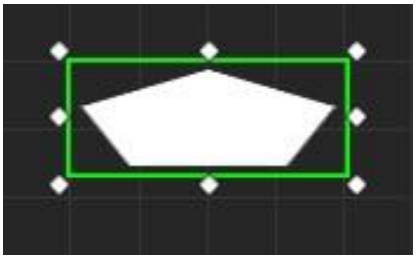

Рисунок 3.13 – Редагування розміру віджета

Замість зміни розміру віджета ми можемо використовувати опцію Size To Content. Ця опція автоматично змінює розмір віджету під розмір його вмісту.

Перебуваючи в панелі Details, перейдіть в розділ Slot (Canvas Panel Slot). Поставте прапорець поруч з Size To Content.

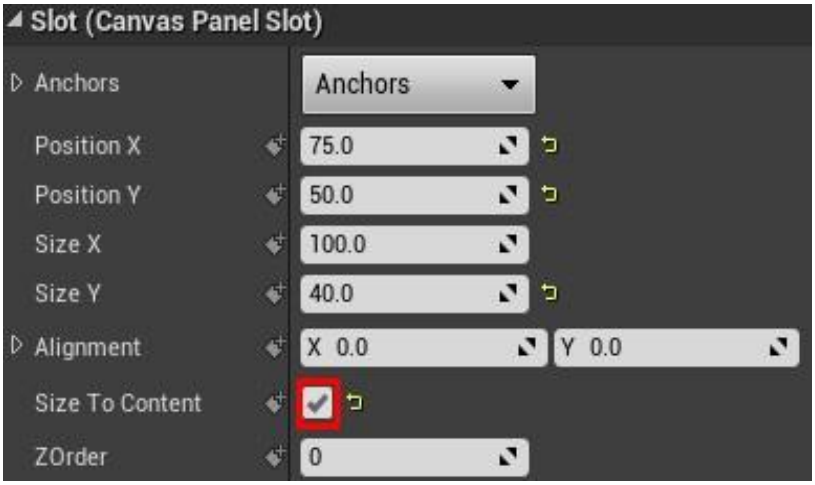

Рисунок 3.14 – Вкладка Slot (Canvas Panel Slot)

Віджет сам піджене свій розмір під зображення.

Рисунок 3.15 - панель Designer

Якщо гра буде запускатися з різними розмірами екрану, то UI повинен відповідним чином переміщати віджети. Щоб зберегти компоновку UI, потрібно використовувати прив'язки.

Прив'язка – це точка що задає місце, щодо якого визначається положення віджету. За замовчуванням віджети прив'язані до верхнього лівого кута свого батьківського елемента. Тому коли ми задаємо положення віджету, ми насправді вказує положення щодо цієї точки прив'язки.

У прикладі нижче кожне зображення прив'язане до однієї точки (до найближчого кутку).

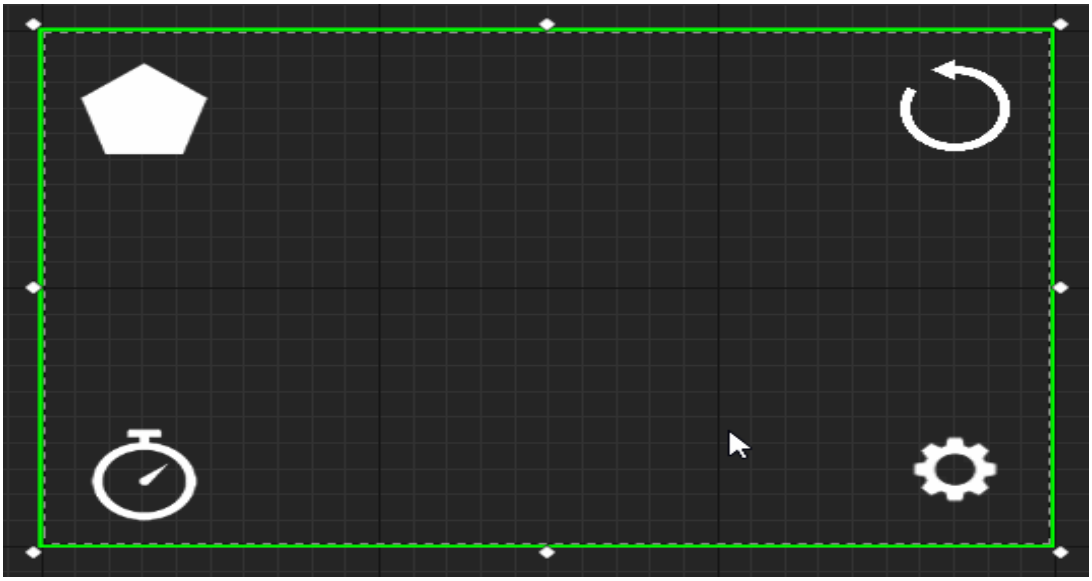

Рисунок 3.16 – Зображення прив'язки

Зауважте, що кожне зображення зберігає положення щодо своєї прив'язки. Завдяки прив'язки UI матиме однакове розташування при різних розмірах екрану.

Також можна використовувати прив'язки для автоматичної зміни розміру віджетів. У разі прив'язки до двох або більше точках віджет буде міняти свій розмір для збереження відносного розміру.

У прикладі нижче індикатор прив'язаний до верхнього лівого і верхнього правого кутах.

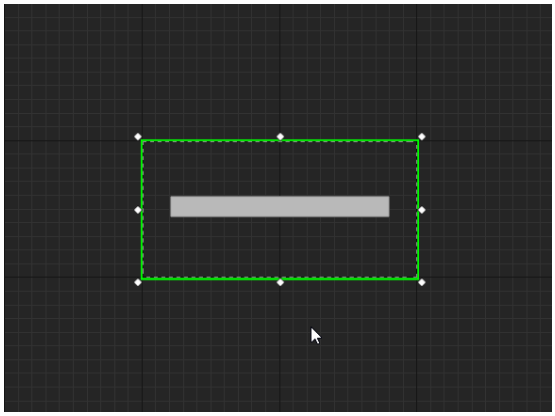

Рисунок 3.17 - панель Designer

Вертикально індикатор переміщається, але не змінює розміру, тому що на осі Y є тільки одна прив'язка (зверху). Однак горизонтально індикатор змінює розмір, тому що має дві точки прив'язки по осі X.

Anchor Medallion відображає розташування прив'язки. Він з'являється при виборі віджета.

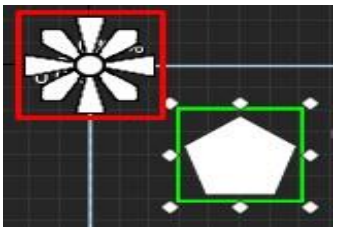

Рисунок 3.18 - Anchor Medallion

Прив'язки CounterText і CounterIcon вже знаходяться в потрібному місці, тому ставити їх не потрібно.

Тепер ми створимо ще по одному віджету Text і Image для таймера. Однак на цей раз ми помістимо їх справа.

Створення таймера

Створіть віджет Text і назвіть його TimerText. Задайте наступні властивості:

Position X: +1225

Position Y: 50

Size X: 500

Size Y: 100

Font Size: 68

Justification: Align Text Right (це вирівняє текст по правій стороні віджета)

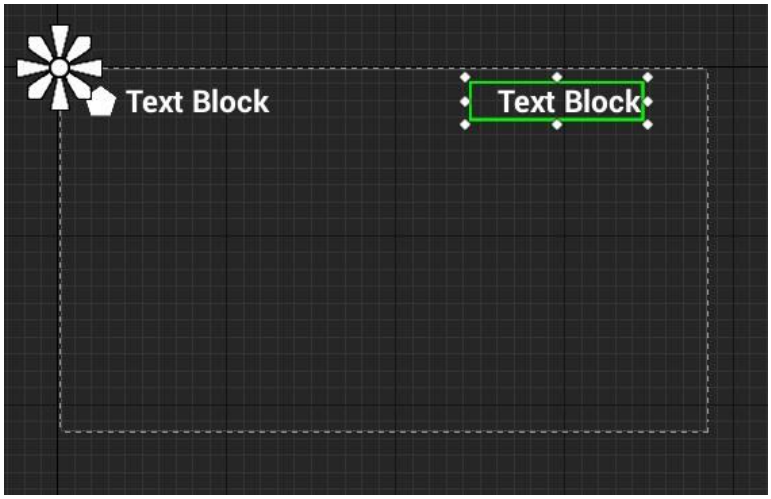

Рисунок 3.19 - Віджет Text

Тепер ми хочемо задати прив'язку в правому верхньому куті. Це можна зробити, перетягнувши мишею коло в Anchor Medallion. Перемістіть Anchor Medallion в правий верхній кут.

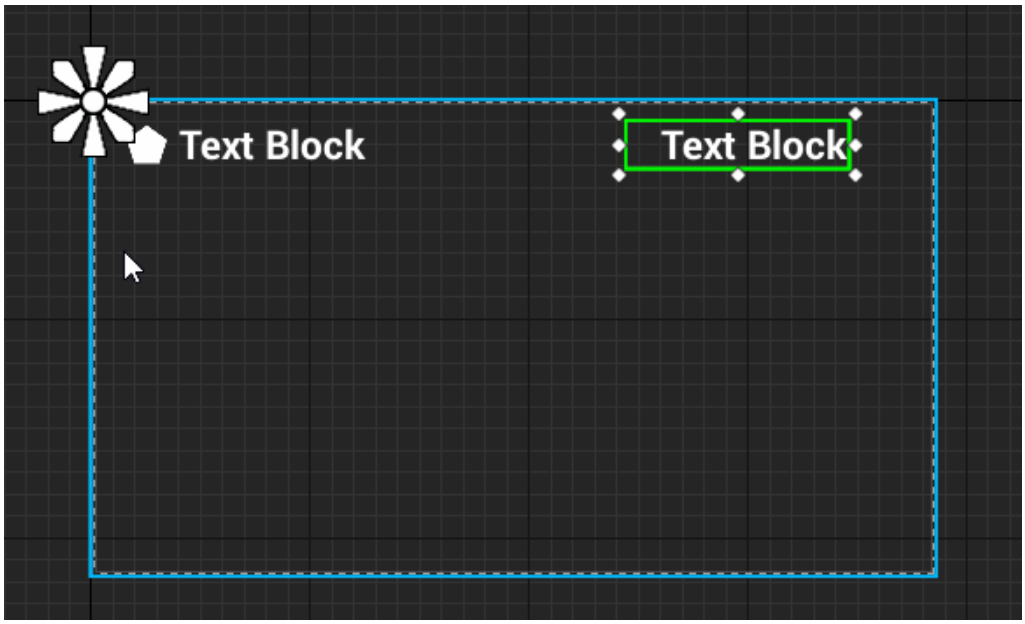

Рисунок 3.20 - Віджет Anchor Medallion

Зауважте, як оновлюється положення щодо прив'язки.

Рисунок 3.21 – Зображення положення щодо прив'язки

Створіть віджет Image і назвіть його TimerIcon. Задайте наступні властивості:

Position X: 1750

Position Y: 50

Size To Content: Checked

Brush \ Image: T_Timer

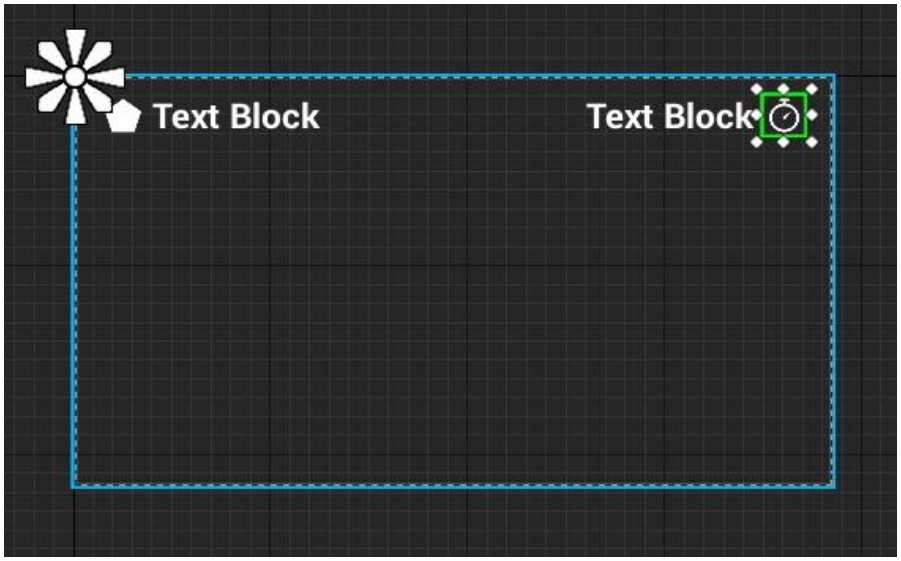

Рисунок 3.22 - Віджет Image

Замість завдання прив'язки за допомогою Anchor Medallion, можна скористатися попередньо встановленими значеннями. Перейдіть в панель Details і натисніть на список, що розкривається поруч з Anchors, щоб відкрити попередньо налаштований. Виберіть третю схему (з квадратом у правому верхньому куті).

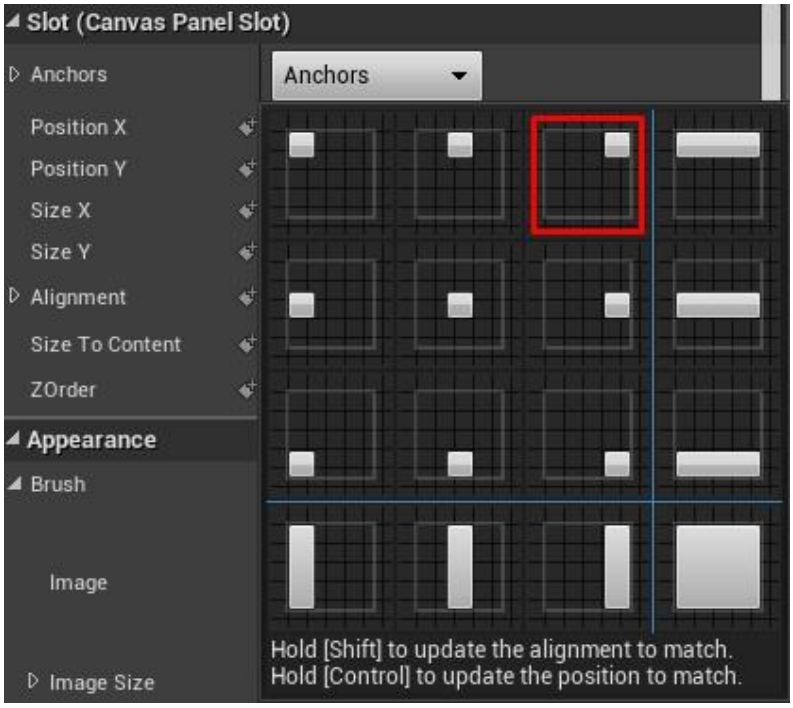

Рисунок 3.23 - Вкладка Slot (Canvas Panel Slot)

Створення схеми UI на цьому завершено. Можна переконатися в тому, що прив'язки працюють, емулюючи різні розміри екрану. Перейдіть в панель Designer і натисніть на список, що розкривається Screen Size.

Рисунок 3.24 - Відображення Screen Size

Вибір опції змінить розмір WBP_HUD для відповідності опції. Нижче показано, як HUD буде виглядати на iPad Air. Зауважте, що віджети стали ближчими один до одного.

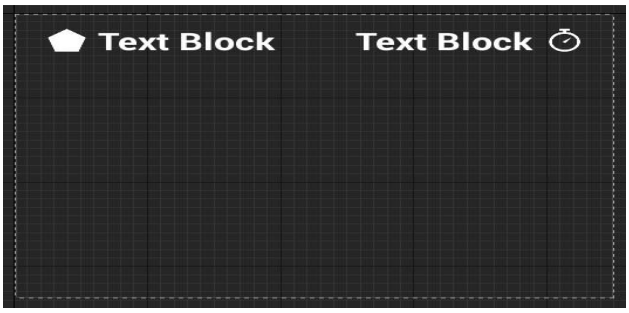

Рисунок 3.25 - Відображення HUD

Натисніть на Compile, а потім поверніться в основний редактор. Перейдіть в папку Blueprints і двічі клацніть на BP_GameManager, щоб відкрити його.

HUD повинен ставати видимим після запуску гри. Для цього можна використовувати нод Event BeginPlay.

Знайдіть нод Event BeginPlay і додайте нод Create Widget в кінець ланцюжка нодов. Цей нод створює екземпляр зазначеного віджета.

Рисунок 3.26 - Ланцюжок нодів

Натисніть на список, що розкривається поруч з Class і виберіть WBP_HUD.

Рисунок 3.27 – Construct NONE

Щоб відобразити HUD, необхідно використовувати нод Add to Viewport. Перетягніть лівою клавішею миші контакт Return Value нода Create Widget. Відпустіть ліву кнопку миші в порожньому просторі, щоб відкрити контекстне меню. Додайте нод Add to Viewport.

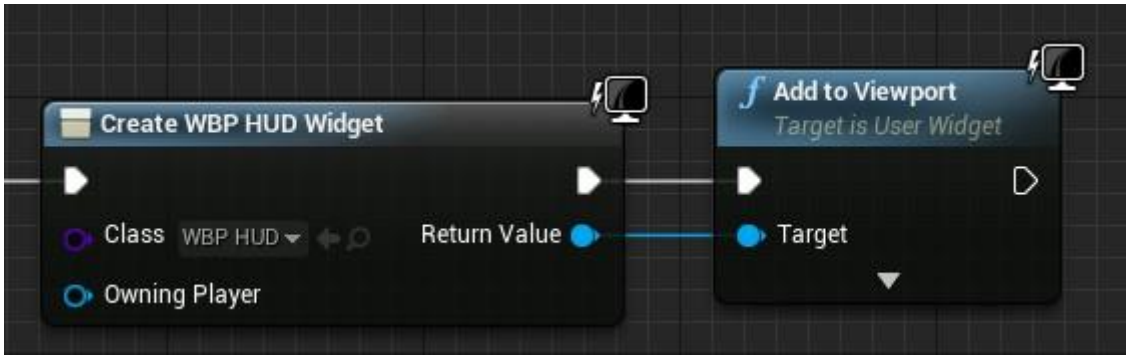

Рисунок 3.28 – Відображення нода Add to Viewport

Давайте розберемося з порядком подій:

Коли Unreal Спауном BP_GameManager, виконуються функції Restart і SetUpCamera. Ці функції налаштовують кілька змінних і камеру. Якщо ви не знаєте, що таке функція, то не хвилюйтеся, скоро ми їх розглянемо.

Нод Create Widget створює екземпляр WBP_HUD

Нод Add to Viewport відображає WBP_HUD

Натисніть на Compile і поверніться в основний редактор. Натисніть на Play, щоб запустити гру з новим HUD.

Щоб відобразити значення лічильника і таймера, нам потрібні змінні для зберігання цієї інформації. Ці змінні можна знайти в BP_GameManager.

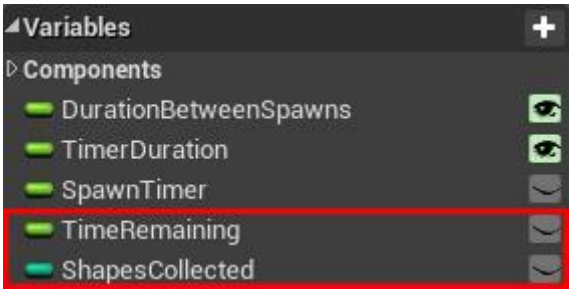

Рисунок 3.29 – Вкладка Variables

Щоб використовувати ці змінні. нам потрібен спосіб отримання доступу до BP_GameManager з WBP_HUD. Для цього можна застосувати змінну-посилання.

Зберігати посилання корисно, тому що вони дозволяють зручно отримувати доступ до екземплярів.

Уявіть, що у вас є одна коробка, в якій лежить м'яч. Якщо нам потрібно знайти і вивчити м'яч, то це буде просто, тому що у нас є всього одна коробка.

Рисунок 3.30 – WBP_HUD

А тепер уявіть, що у нас сотня коробок, але м'яч є тільки в одній. Нам потрібно буде перевіряти кожну коробку, поки ми не знайдемо коробку з м'ячем.

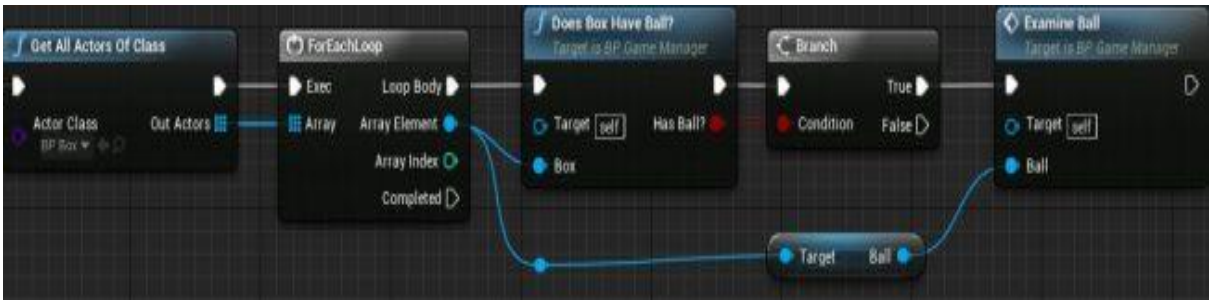

Рисунок 3.31 – Ланцюжок нодів

Кожен раз, коли нам потрібно буде вивчити м'яч, доведеться виконувати цю операцію. Це швидко призведе до проблем з продуктивністю.

Завдяки посиланнях можна відстежувати коробку з м'ячем. Таким чином, нам не доведеться перевіряти кожну коробку.

Рисунок 3.32 – Відображення посилання

Відкрийте WBP_HUD і перейдіть в режим Graph, перейшовши в Editor Mode і вибравши Graph.

Рисунок 3.33 – Режим Graph

Перейдіть на вкладку My Blueprint і створіть нову змінну GameManager.

Перейдіть в панель Details і натисніть на список, що розкривається поруч з Variable Type. Знайдіть BP_GameManager і виберіть BP Game Manager \ Object Reference.

Рисунок 3.33 – BP Game Manager

Натисніть на Compile і відкрийте BP_GameManager.

Знайдіть нод Create Widget і перетягніть лівою клавішею миші контакт Return Value. Відпустіть ліву клавішу на порожньому просторі і виберіть з меню Set Game Manager.

Потім з'єднайте нод Add to Viewport з нодом Set Game Manager.

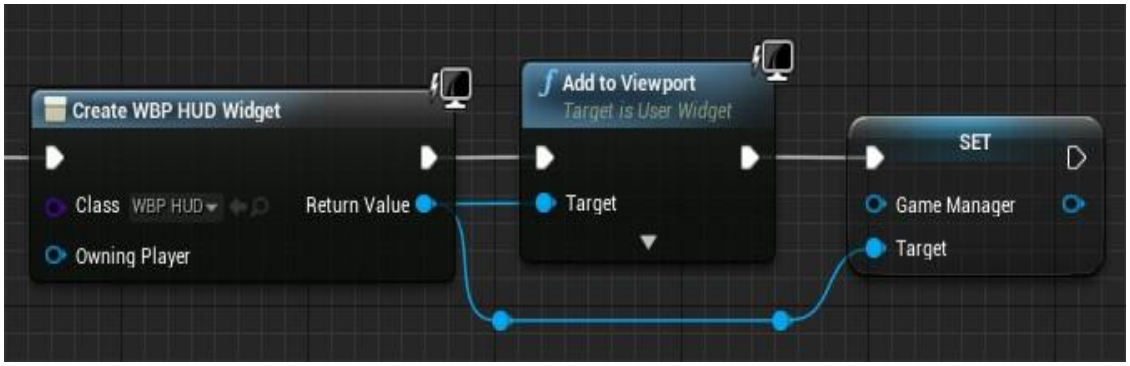

Рисунок 3.35 – Ланцюжок нодів

Потім створіть нод Self і з'єднайте його з лівим контактом нода Set Game Manager. Нод Self може бути перелічений як Get a reference to self.

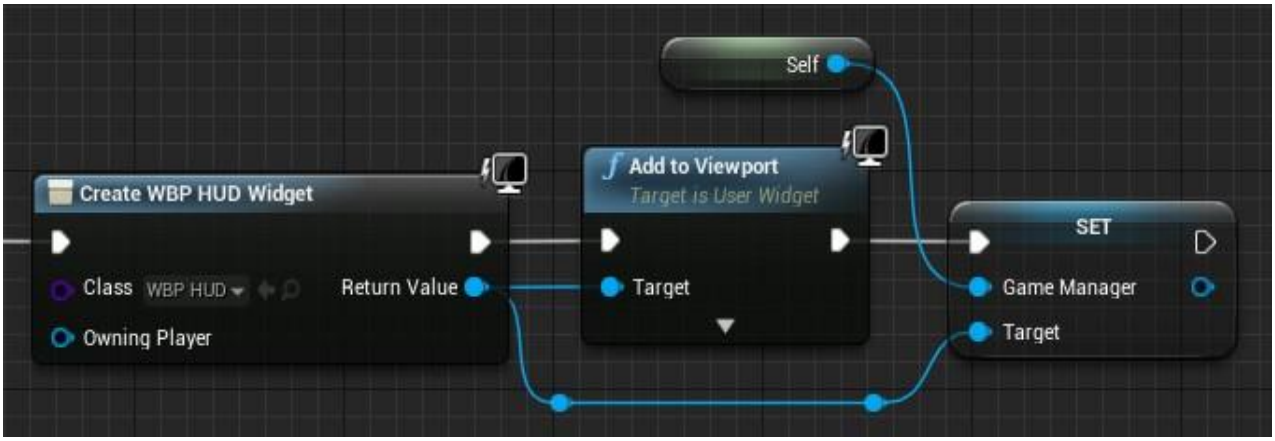

Рисунок 3.36 – Ланцюжок нодів

Тепер, коли WBP_HUD вже створений, у нас буде посилання на BP_GameManager.фТеперь, коли WBP_HUD вже створений, у нас буде посилання на BP_GameManager.

Функції в Blueprints - це графи, схожі на Event Graph. На відміну від Event Graph функції можна викликати за допомогою нодов. Навіщо ж це може знадобитися?

Впорядкованість

Одна з причин для використання функцій - впорядкованість. Завдяки функціям можна з'єднати кілька нодов в один.

Подивіться на розділ Event BeginPlay в BP_GameManager. Тут є дві функції: Restart і SetUpCamera.

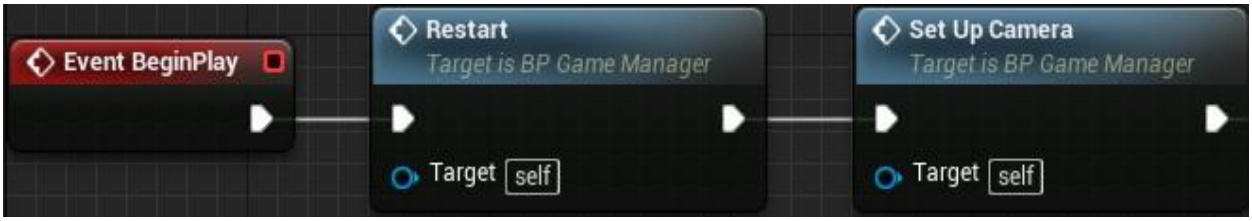

Рисунок 3.37 – Event BeginPlay в BP_GameManager

Ось як буде виглядати цей розділ без функцій:

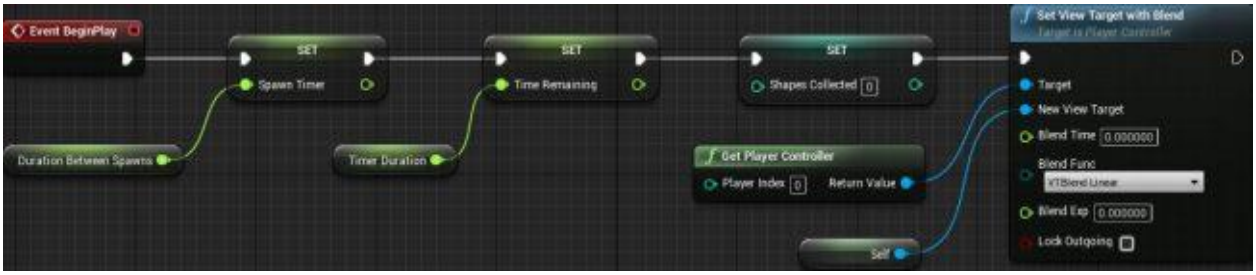

Рисунок 3.38 – Розділ без функцій

Як ви бачите, завдяки функціям він виглядає набагато чистіше.

Повторне використання

Ще одна причина для застосування функцій - повторне використання. Наприклад, якщо вам потрібно скинути лічильник і таймер, то це можна запросто зробити за допомогою функції Restart.

Рисунок 3.39 – Зображення функції Restart

Це дозволяє заощадити час на відтворення нодов кожен раз, коли вам потрібно скинути ці змінні.

Ми познайомилися з функціями і тепер можемо використовувати їх для оновлення віджета CounterText.

Оновлення віджета

При створенні віджету також автоматично створюється і змінна-посилання на цей віджет. Однак за замовчуванням віджети Text не мають змінних-посилань. Це означає, що ми не зможемо ставити їх властивість Text. На щастя, це легко виправити.

Натисніть на Compile і відкрийте WBP_HUD. Перейдіть в режим Designer.

Виберіть CounterText, а потім перейдіть в панель Details. Переконайтеся, що в самому верху є прапорець Is Variable.

Рисунок 3.40 – Панель Details

Тепер ми зможемо оновлювати CounterText. Наступним кроком буде створення функції для поновлення тексту.

Створення функції поновлення

Перейдіть назад в режим Graph і перейдіть у вкладку My Blueprint. Натисніть на значок + праворуч від розділу Functions.

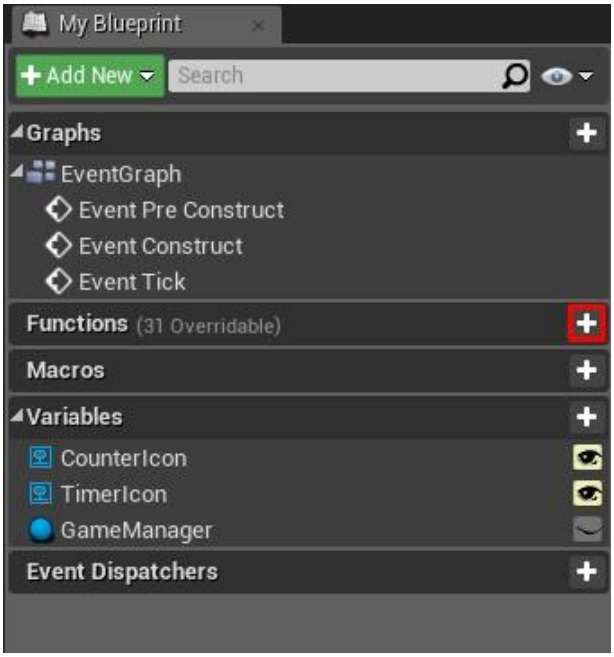

Рисунок 3.41 – Вкладка My Blueprint

При цьому буде створена нова функція і ви перейдете до її графу. Перейменуйте функцію в UpdateCounterText.

За замовчуванням граф буде містити нод Entry. При виконанні функції вона буде починати з цього нода.

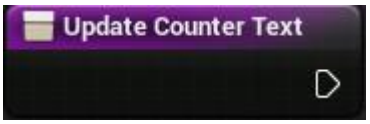

Рисунок 3.42 – функцію в UpdateCounterText

Щоб CounterText відображав змінну ShapesCollected, нам потрібно з'єднати їх.

Перетягніть в граф змінну GameManager. Перетягніть лівою клавішею миші його контакт і відпустіть на порожньому просторі. У меню виберіть Get Shapes Collected.

Рисунок 3.43 – Графа змінної GameManager

Щоб задати текст, потрібно буде використовувати нод SetText (Text). Перетягніть змінну CounterText в граф. Перетягніть лівою клавішею миші її контакт і відпустіть на порожньому просторі. В меню додайте нод SetText (Text).

Рисунок 3.44 – Нод SetText

SetText (Text) може отримувати вхідні дані типу Text. Однак змінна ShapesCollected має тип Integer. На щастя, Unreal автоматично виконує перетворення при підключенні Integer до входу Text.

З'єднайте змінну ShapesCollected з контактом In Text нода Set Text (Text). Unreal автоматично створить нод ToText (int).

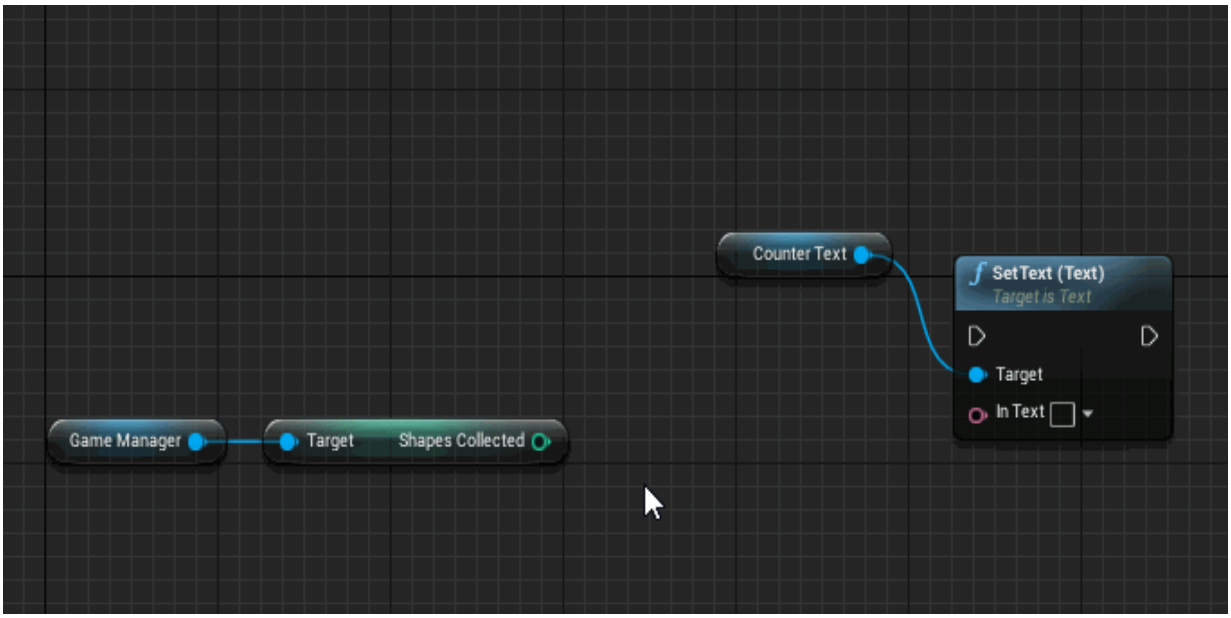

Рисунок 3.45 – Змінна ShapesCollected

Щоб доробити функцію, з'єднайте нод Entry з нодом Set Text (Text).

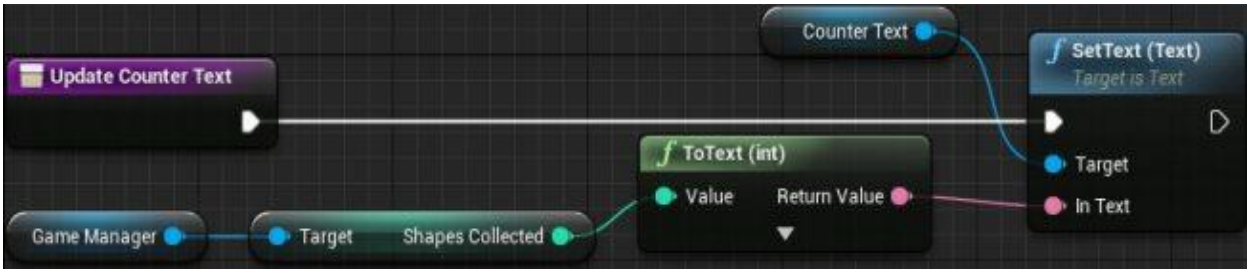

Рисунок 3.46 – З'єднання нода Entry з нодом Set Text

Порядок подій:

При виклику UpdateCounterText функція отримує змінну ShapesCollected від BP_GameManager

Нод ToText (int) перетворює значення ShapesCollected в тип Text

SetText (Text) задає текст CounterText з значення ToText (int)

Тепер нам потрібно навчитися викликати UpdateCounterText, коли гравець ловить фігуру.

Виклик функції поновлення

Найкраще викликати UpdateCounterText відразу після того, як гра збільшує значення ShapesCollected. Я створив функцію IncrementShapesCollected, яка виконує інкремент лічильника. Фігури викликають цю функцію, коли стикаються з гравцем.

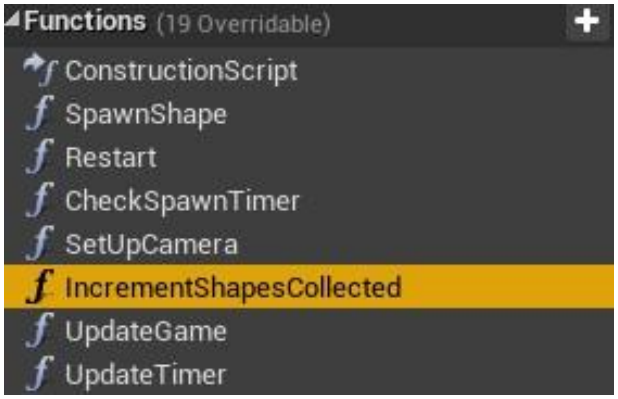

Рисунок 3.47 – Вкладка Functions

Натисніть на Compile і поверніться в BP_GameManager.

Перш ніж викликати UpdateCounterText, нам потрібна посилання на WBP_HUD. Спробуйте створити змінну-посилання самостійно!

Знайдіть розділ, в якому ми створювали і відображали WBP_HUD.

Перетягніть контакт Return Value нода Create Widget.

Відпустіть ліву кнопку миші на порожньому просторі і виберіть в меню Promote to variable.

Додайте новий нод в кінець ланцюжка нодов.

Після створення змінної змініть її ім'я на HUDWidget.

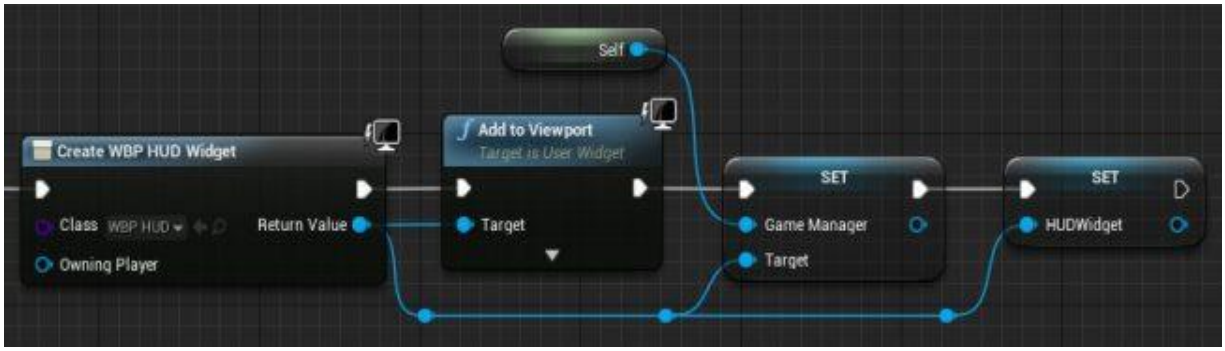

Рисунок 3.48 – Ланцюжок нодів

Потім перетягніть правий контакт нода Set HUDWidget і відпустіть на порожньому просторі. Додайте нод UpdateCounterText. Завдяки цьому CounterText буде відображати значення ShapesCollected при запуску гри.

Рисунок 3.49 – Кінець ланцюжка нодів

Потім перейдіть в панель My Blueprint і зайдіть в розділ Functions. Двічі клацніть на IncrementShapesCollected, щоб відкрити його граф.

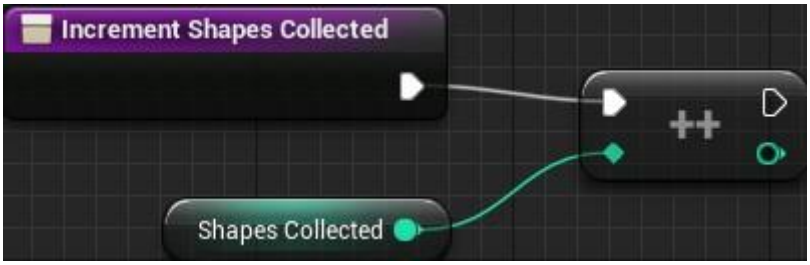

Рисунок 3.50 – Кінець ланцюжка нодів

Перетягніть змінну HUDWidget в граф. Перетягніть його контакт і відпустіть на порожньому просторі. Додайте нод UpdateCounterText і з'єднайте його наступним чином:

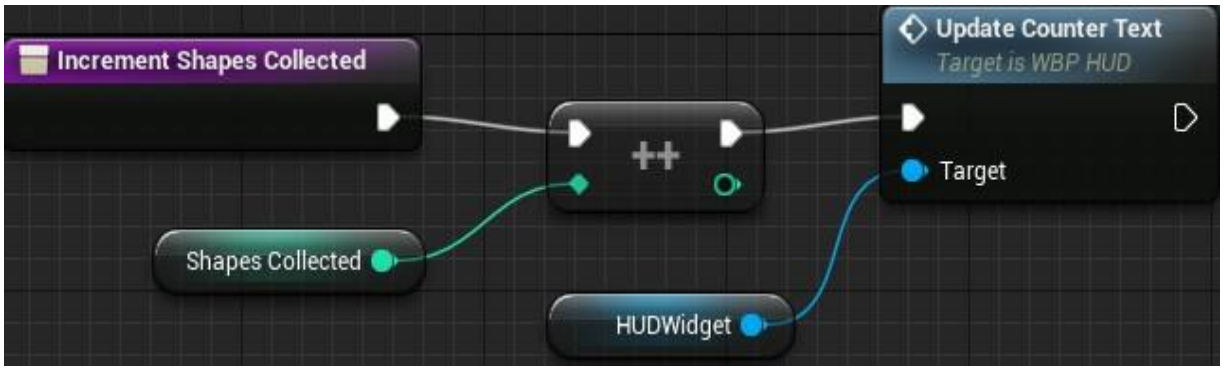

Рисунок 3.51 – Нод UpdateCounterText

Тепер при виконанні IncrementShapesCollected вона буде збільшувати ShapesCollected, а потім викликати UpdateCounterText. Ця функція потім оновить CounterText значенням ShapesCollected.

Натисніть на Compile і закрийте BP_GameManager.

Тепер нам потрібно навчитися оновлювати віджет TimerText, тільки іншим способом, який називається прив'язкою.

Bindings

Прив'язки дозволяють автоматично оновлювати певні властивості віджетів. Щоб мати можливість прив'язки, властивість має володіти списком Bind.

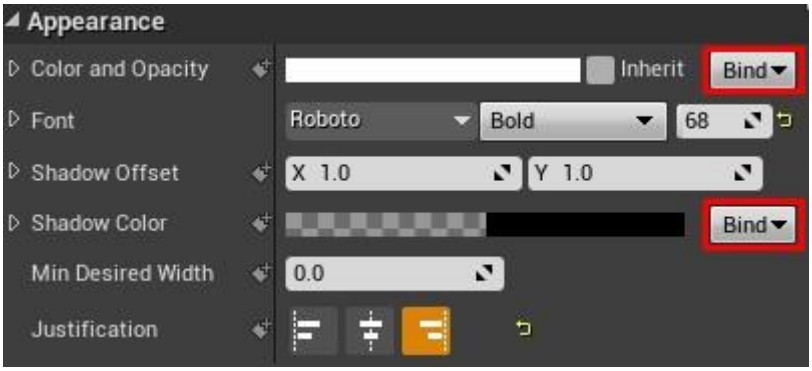

Рисунок 3.52 – Вкладка Appearance

Можна прив'язувати властивості до функції або до змінної, що зберігається всередині віджета. Прив'язка постійно отримує значення від функції або змінної і привласнює прив'язаному властивості це значення.

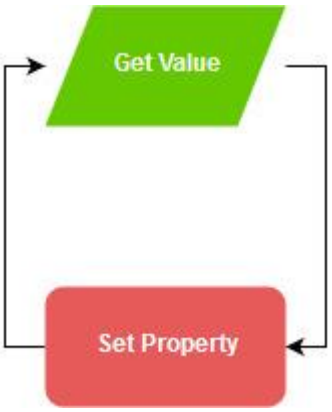

Рисунок 3.53 – Зображення віджета

Ви, напевно, ставите питанням, чому ж не можна постійно використовувати прив'язки. Прив'язки неефективні, тому що постійно оновлюються. Це означає, що гра витрачає час на оновлення властивості, навіть якщо немає ніякої нової інформації. Порівняйте це з попереднім способом, при якому віджет оновлювався тільки при необхідності.

З урахуванням цього прив'язки добре підходять для часто змінюються елементів, таких як таймери. Давайте створимо прив'язку для TimerText.

Створення прив'язки

Відкрийте WBP_HUD і перейдіть в режим Designer.

Виберіть TimerText, а потім перейдіть в розділ Content в панелі Details. Ви побачите, що властивість Text має можливість прив'язки. Натисніть на список, що розкривається Bind і виберіть Create Binding.

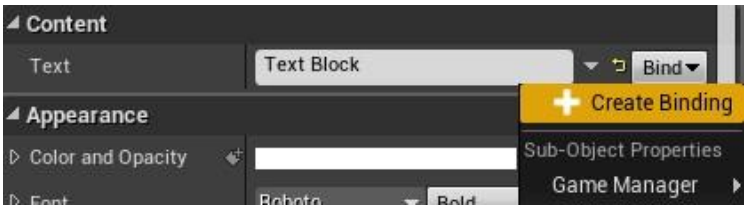

Рисунок 3.54 – Розділ Content

При цьому створиться нова функція і виконається перехід до її графу. Перейменуйте функцію в UpdateTimerText.

Функція буде мати нод Return з контактом Return Value типу Text. TimerText буде відображати будь-який текст, який ви підключите до цього контакту.

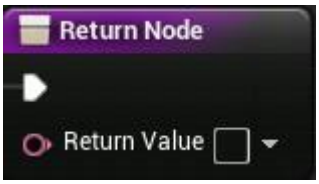

Рисунок 3.55 – Нод Return

Перетягніть GameManager на граф і отримаєте з нього змінну TimeRemaining.

З'єднайте змінну TimeRemaining з Return Value нода Return. Як і минулого разу, Unreal автоматично додасть нод перетворення.

Рисунок 3.56 – Нод перетворення

Підведемо підсумок:

Прив'язка буде постійно викликати функцію UpdateTimerText

Функція отримує змінну TimeRemaining від BP_GameManager

Нод ToText (float) перетворює значення з TimeRemaining в тип Text.

Перетворене значення потім виводиться в нод Return

Наш HUD нарешті готовий. Натисніть на Compile і закрийте WBP_HUD.

Далі я почав роботу з системою частинок.

Системи частинок - найважливіший компонент візуальних ефектів. Вони дозволяють художникам створювати такі ефекти, як вибухи, дим і дощ.

В Unreal Engine 4 є надійна і зручна система під назвою Cascade для створення ефектів частинок. Ця система дозволяє створювати модульні ефекти і легко управляти поведінкою частинок.

Система частинок - це система для створення частинок і управління ними. Частка - це просто точка в просторі. За допомогою систем частинок можна керувати зовнішнім виглядом і поведінкою частинок.

Системи частинок складаються з одного або декількох компонентів, званих емітерами. Вони виконують спаунінг частинок.

У емітерів також є компоненти, звані модулями. Модулі керують окремими властивостями частинок, що створюються емітером, наприклад, матеріалом і початковою швидкістю частинки. У прикладі нижче використані два модулі для додання кожній частинці матеріалу червоного кола і випадкової швидкості.

Також можна змінювати колір частки з плином терміну її існування.

Тепер ми знаємо, що таке система частинок, і ми можемо створити її для двигунів корабля.

Створення системи частинок

Перейдіть в папку ParticleSystems і натисніть на Add New \ Particle System. Перейменуйте систему частинок в PS_Thruster і відкрийте її.

Cascade: редактор систем частинок

Cascade складається з чотирьох основних панелей:

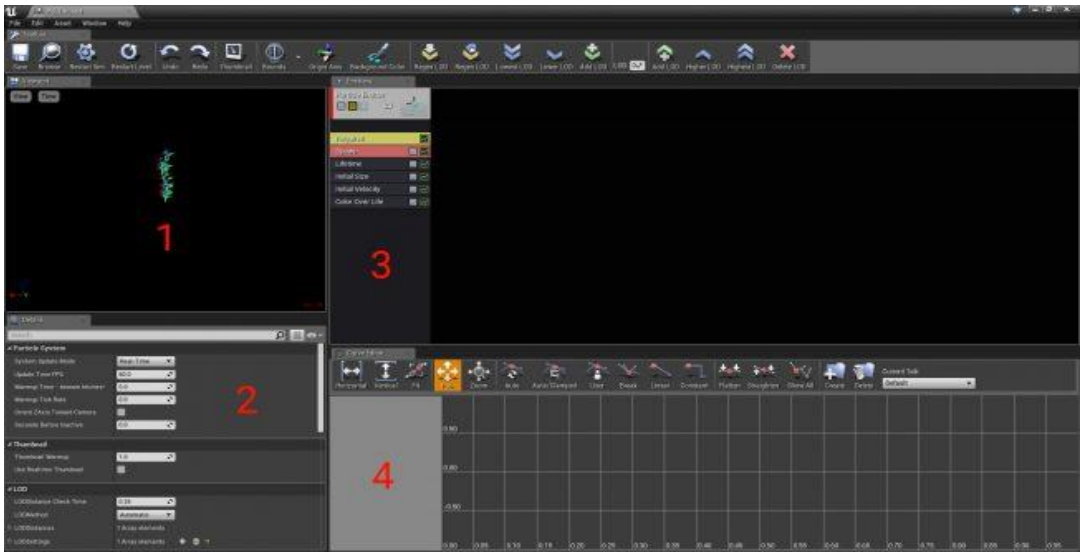

Рисунок 3.56 – Редактор систем частинок

Viewport: в цій панелі відображається зовнішній вигляд системи частинок. Можна повертати її, утримуючи праву кнопку миші і рухаючи нею. Для переміщення утримуйте праву кнопку миші і натискайте клавіші WASD.

Details: тут відображаються всі властивості обраних компонентів (емітерів, модулів і т.д.). Якщо нічого не вибрано, то тут відображаються властивості системи частинок.

Emitters: в цій панелі відображається список емітерів зліва направо. У кожного емітера показаний список його модулів.

Curve Editor: редактор Curve Editor дозволяє візуалізувати і змінювати значення кривих модулів. Не всі властивості модулів підтримують криві.

Поки наша система використовує матеріал частинок за замовчуванням.

Для початку замінимо матеріал частинок на матеріал кола.

Перейдіть в панель Emitters і виберіть модуль Required.

Рисунок 3.57 – Панель Particle Emitters

Модуль Required містить необхідні властивості, такі як матеріал частинок і тривалість роботи емітера. Модуль Required повинен бути у кожного емітера.

Для зміни матеріалу перейдіть в панель Details і задайте для Material значення M_Particle. При цьому частинки стануть помаранчевими колами.

Приєднання системи частинок

Поверніться в основний редактор і зайдіть в папку Blueprints. Відкрийте BP_Player і перейдіть до панелі Components.

Для використання системи частинок можна застосувати компонент Particle System. Створіть його і перейменуйте в ThrusterParticles. З'єднайте його з компонентом Collision.

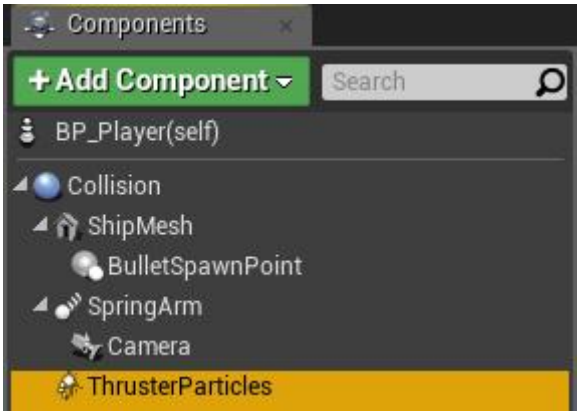

Рисунок 3.58 – ThrusterParticles в панелі Components

Для Задати системи частинок перейдіть в панель Details і знайдіть розділ Particles. Задайте Template значення PS_Thruster.

Потім задайте для для Location компонента ThrusterParticles значення (-80, 0, 0).

Нарешті, задайте Rotation значення (0, 90, 0). Це направить систему частинок таким чином, що частки будуть віддалятися.

Натисніть на Compile і поверніться в основний редактор.

Система частинок працює, але частинки рухаються занадто повільно і вони дуже малі. Це можна виправити, задавши початкову швидкість і розмір часток.

Завдання швидкості і розміру часток

Спочатку ми поставимо початкову швидкість частинок. Відкрийте PS_Thruster і виберіть модуль Initial Velocity. Потім розгорніть Start Velocity \ Distribution.

За замовчуванням початкова швидкість часток знаходиться в інтервалі від (-10, -10, 50) до (10, 10, 100).

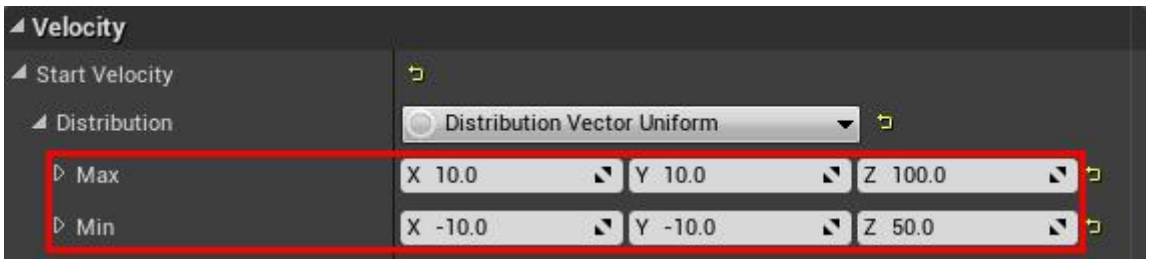

Рисунок 3.59 – Модуль Velocity

Щоб частки віддалялися швидше, нам потрібно збільшити швидкість по Z. Дайте Min Z значення 300, а Max Z - значення 400.

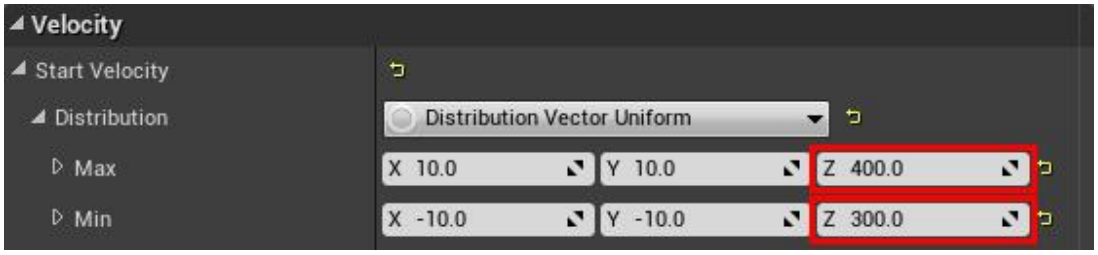

Рисунок 3.60 – Модуль Velocity

Далі нам потрібно задати початковий розмір часток.

Завдання розміру часток

Виберіть модуль Initial Size і перейдіть в панель Details. Потім розгорніть Start Size \ Distribution.

Як і в модулі Initial Velocity, в Initial Size теж є інтервал мінімальних і максимальних значень. Однак в цьому туторіали ми поставимо постійний розмір часток. Для цього виберіть для Distribution значення Distribution Vector Constant.

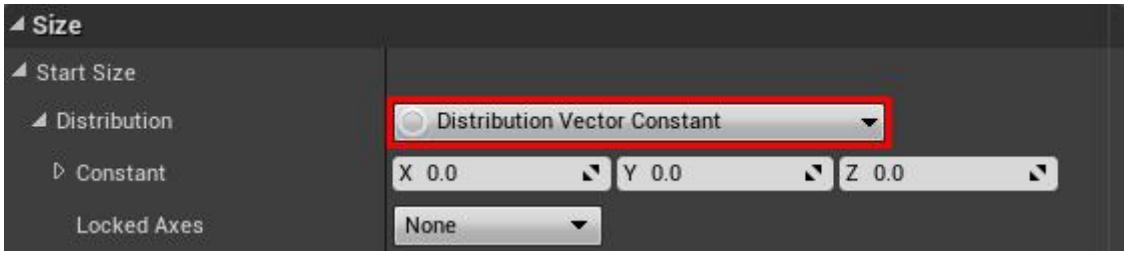

Рисунок 3.61 – Модуль Velocity вкладка Size

Частинки виглядають краще, але відстань між ними як і раніше занадто велика. Так вийшло через занадто тривалого інтервалу між Спауном частинок. Щоб виправити це, ми можемо збільшити швидкість Спауна.

Збільшення швидкості Спаун частинок

Для збільшення швидкості Спаун нам потрібно скористатися модулем Spawn. Цей модуль управляє швидкістю Спаун частинок емітером. Разом з Required кожен емітер повинен мати модуль Spawn.

Відкрийте PS_Thruster і виберіть Spawn. Перейдіть в панель Details і розгорніть розділ Spawn \ Rate.

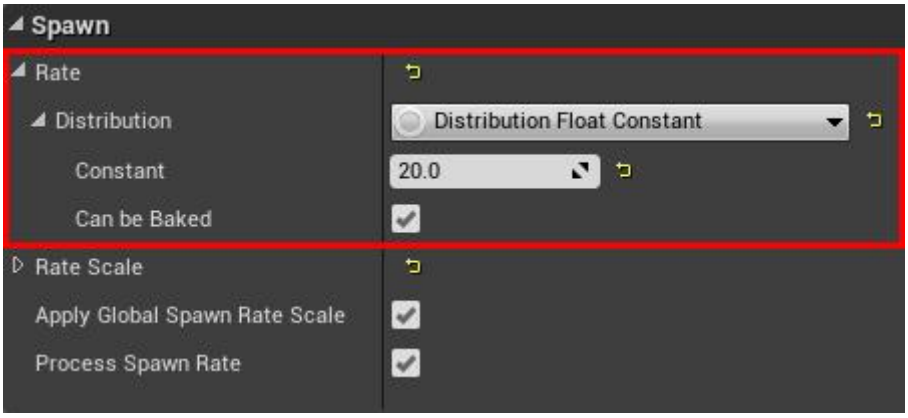

Рисунок 3.62 – Панель Details вкладка Spawn

Задайте Constant значення 50. Це збільшить швидкість Спаун до 50 частинок в секунду.

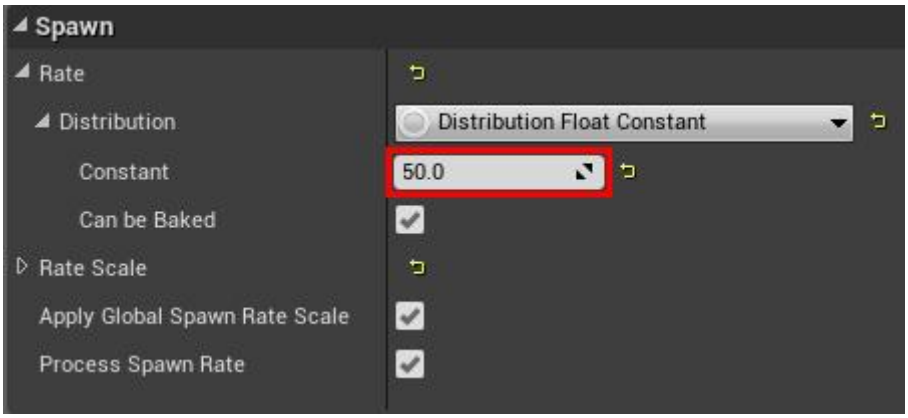

Рисунок 3.63 – Панель Details вкладка Spawn

Як ви бачите, тепер частинки більше нагадують слід. Щоб зробити частинки більше схожими на полум'я двигуна, можна зменшити термін їх життя.

Зменшення терміну існування частинок

Відкрийте PS_Thruster і Перей в панель Emitters.

Щоб зменшити час існування частинок, потрібно скористатися модулем Size By Life. Цей модуль застосовує множник розміру частки протягом терміну її існування. Створіть його, натиснувши правою клавішею миші на порожньому просторі в емітер і вибравши Size \ Size By Life.

Рисунок 3.64 – Панель Emitters модуль Size By Life

За замовчуванням це не вплине візуально на розмір часток, тому що за замовчуванням множник завжди дорівнює 1. Щоб зменшити частку, нам потрібно змінити криву модуля, щоб множник розміру згодом зменшувався. Але спочатку потрібно розібратися, що ж таке крива?

Крива (curve) - це безліч точок. Кожна точка володіє двома властивостями: становищем і значенням.

Коли у нас є дві або більше точок, то вони створюють лінію. Нижче представлений приклад простої лінійної кривої. Точка A має положення і значення, рівні 0. Точка B має положення 2 і значення 1.

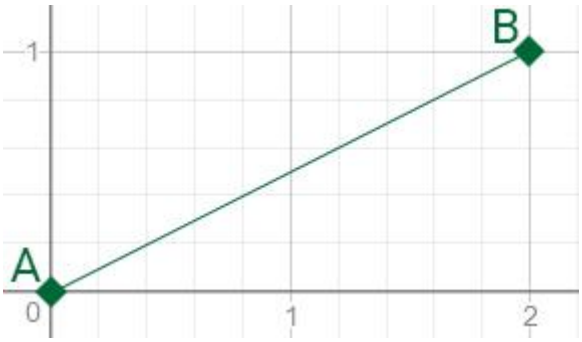

Рисунок 3.65 – Крива

Якщо семпліровалісь лінійну криву в будь-якій точці, то це працює як лінійна інтерполяція. Наприклад, якщо семпліровалісь криву в точці 1, то ми отримаємо значення 0.5.

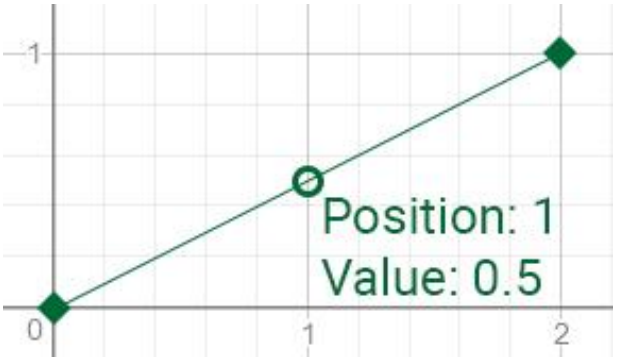

Рисунок 3.66 – Лінійна інтерполяція

Якщо створити спадну криву, то одержуване значення поступово зменшуватиметься. Саме таку криву ми хочемо використовувати для модуля Size By Life.

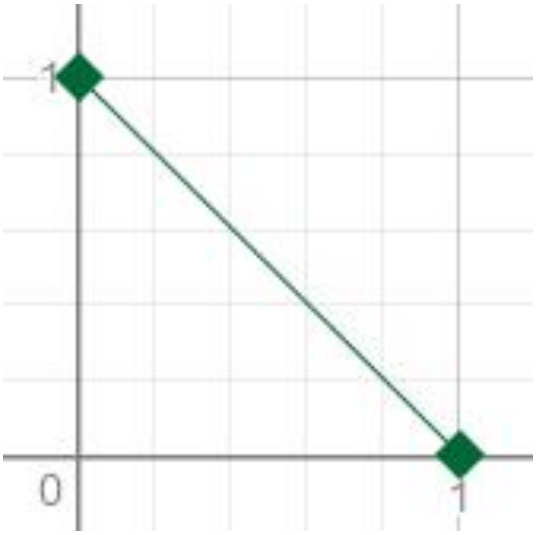

Рисунок 3.67 – Спадна крива

Тепер ми створимо показану вище криву в Cascade.

Зміна кривої модуля

Виберіть Size By Life і перейдіть в панель Details. Розгорніть Life Multiplier \ Distribution \ Constant Curve \ Points. Тут представлений список точок кривої Life Multiplier.

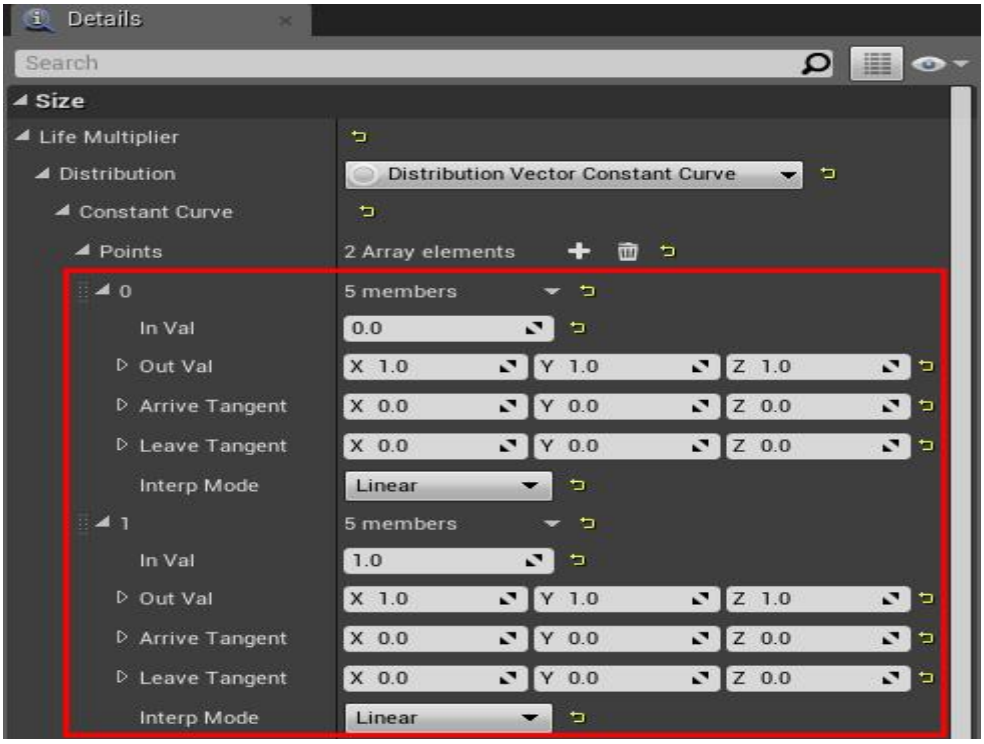

Рисунок 3.68 – Панель Details

In Val - це положення точки на кривій. Для Size By Life значення 0 означає початок терміну життя частинки. Значення 1 означає кінець терміну життя частинки.

Для зменшення множника розміру згодом нам потрібно зменшити Out Val другий точки. Задайте для Out Val точки 1 значення (0, 0, 0). Це поступово знизить розмір частки до 0.

Рисунок 3.69 – Панель Details вкладка Points

Наочно побачити криву Life Multiplier можна в Curve Editor. Для цього натисніть на значок графа модуля Size By Life.

Рисунок 3.70 – Модуль Size By Life

Це додасть Life Multiplier в Curve Editor. Щоб крива містилася в вікно, натисніть на Fit в Curve Editor.

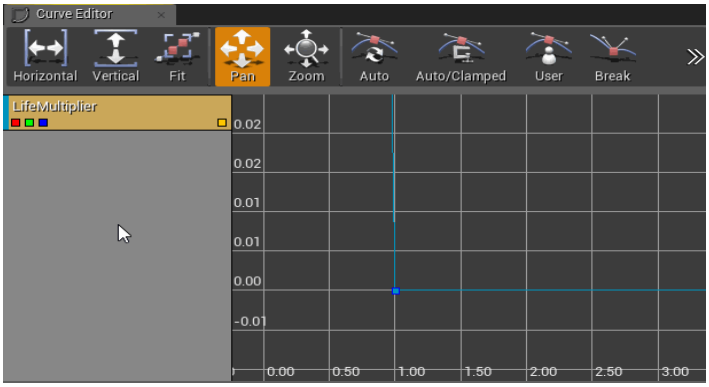

Рисунок 3.71 – Curve Editor

Як ви бачите, множник розміру зменшується за термін життя частинки з 1 до 0.

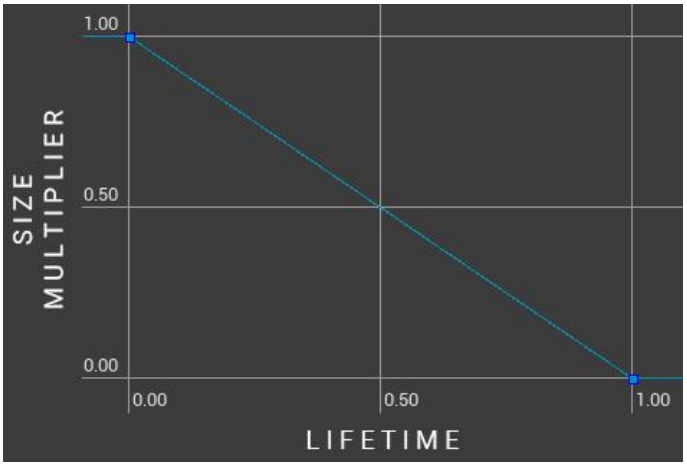

Рисунок 3.72 – Вікно в Curve Editor

Тепер частинки більше схожі на полум'я! Останнє, що ми додамо до цієї системи частинок - колірні варіації.

Щоб задати колір частки за допомогою Cascade, нам потрібно правильно вибрати матеріал частинок. Перейдіть в папку Materials і відкрийте M_Particle.

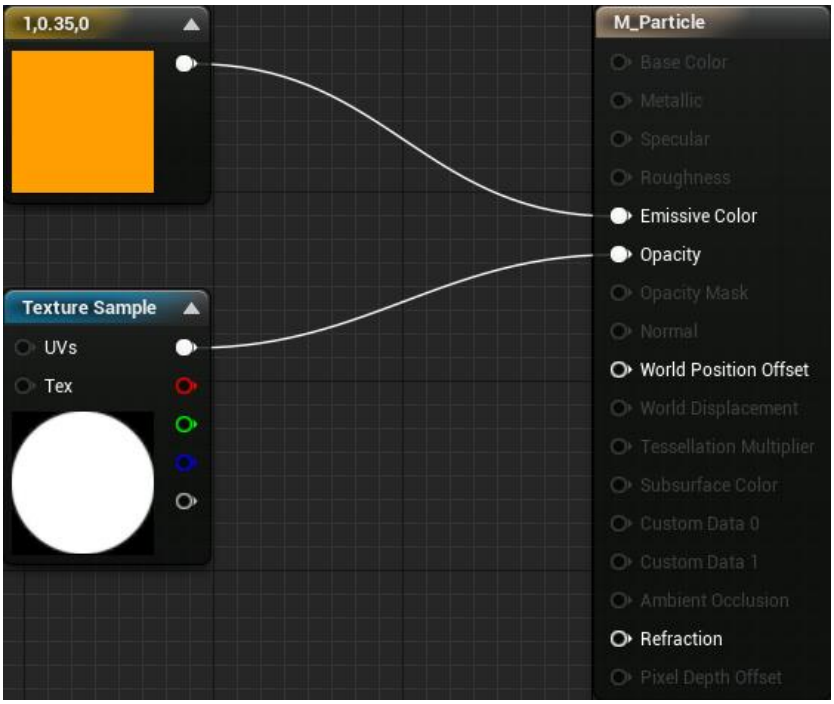

Рисунок 3.73 – Папка Materials

Зараз колір заданий в матеріалі. Щоб використовувати колір з системи частинок, нам потрібно застосувати нод ParticleColor.

По-перше, видаліть нод, з'єднаний з Emissive Color. Потім додайте нод ParticleColor і з'єднайте його наступним чином:

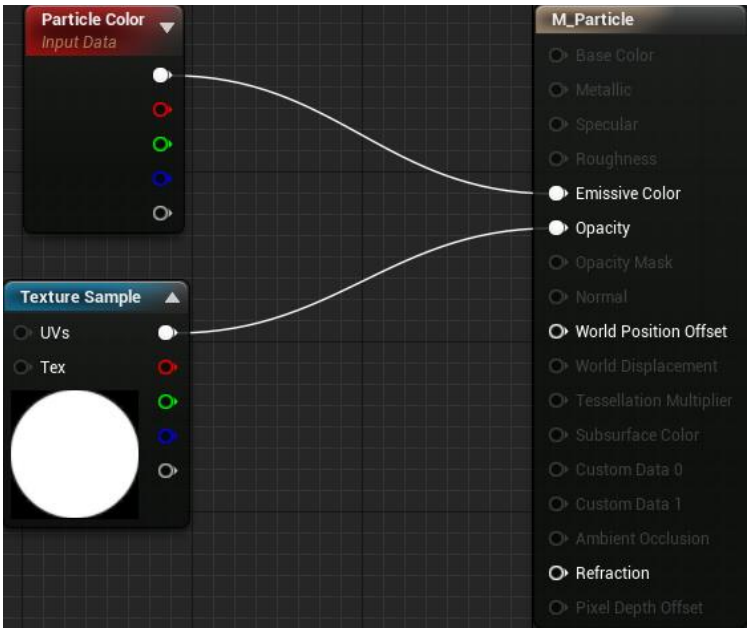

Рисунок 3.74 – Нод ParticleColor

Натисніть на Apply і закрийте M_Particle.

Щоб задати колір частки, можна використовувати модуль Initial Color.

Відкрийте PS_Thruster і додайте модуль Initial Color. Його можна знайти в категорії Color.

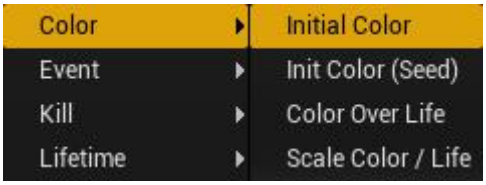

Рисунок 3.75 – Модуль Initial Color

Щоб додати колірні варіації, нам потрібно задати інтервал, в якому може перебувати колір. Для цього можна скористатися розподілами.

Виберіть Initial Color і перейдіть в панель Details. Розгорніть розділ Start Color і змініть Distribution на Distribution Vector Uniform. Це дозволить нам з певними інтервалами для кожного колірного каналу.

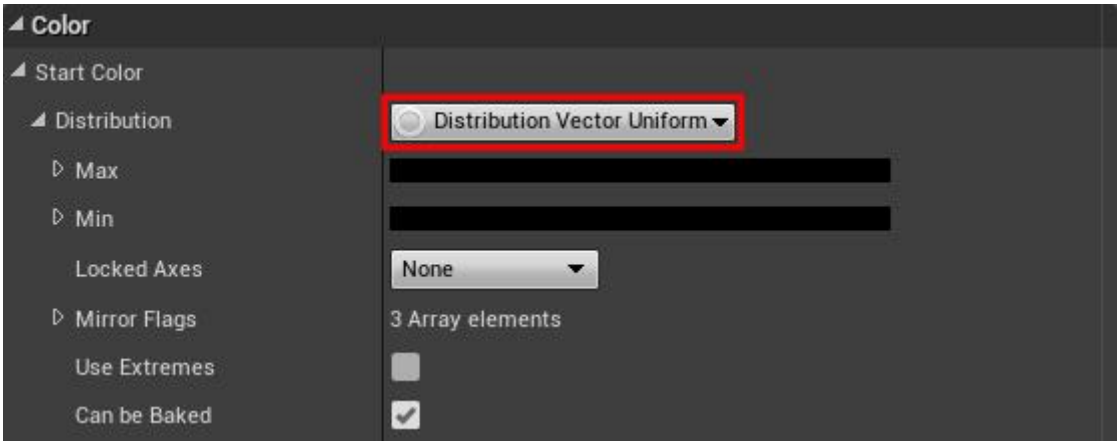

Рисунок 3.76 – Розділ Start Color в панелі Details

У цій роботі колір буде перебувати в інтервалі від оранжевого до червоного. Для цього потрібно задати Max значення (1.0, 0.0, 0.0), а Min - значення (1.0, 0.35, 0.0).

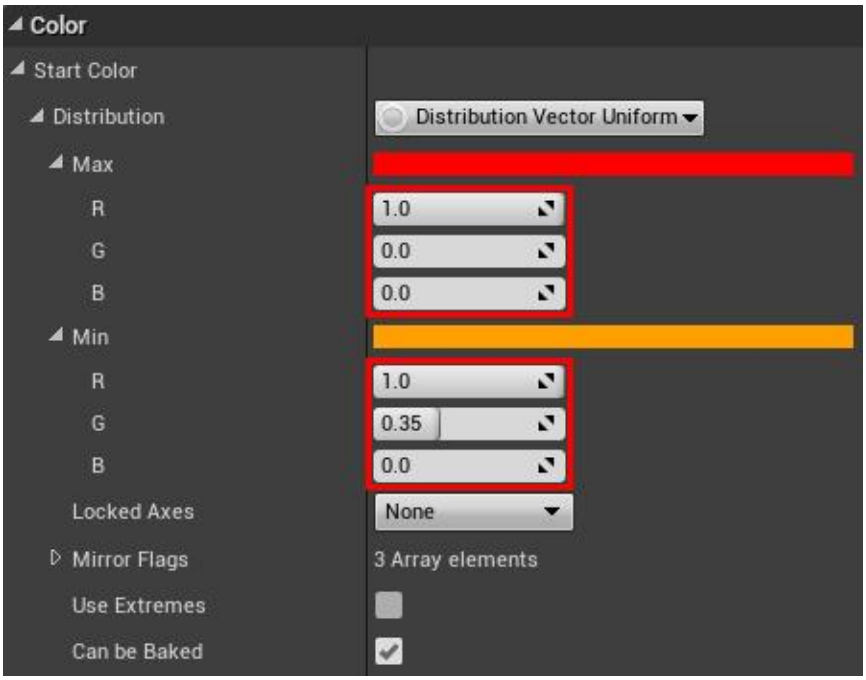

Рисунок 3.77 – Вкладка Color в панелі Details

Якщо ви подивіться на Viewport, то побачите, що колір поводиться дивно.

Так відбувається тому, що модуль Color Over Life постійно робить колір білим. Щоб виправити це, виберіть Color Over Life і натисніть Delete. Тепер ваш список модулів буде виглядати так:

Рисунок 3.78 – Панель Particle Emitter

Закрийте PS_Thruster в основному редакторі.

Тепер нам потрібно навчитися перемикати систему частинок.

Включення / відключення системи частинок.

Відкрийте BP_Player і знайдіть нод Event Tick. Додайте наступну схему в кінці ланцюжка нодов:

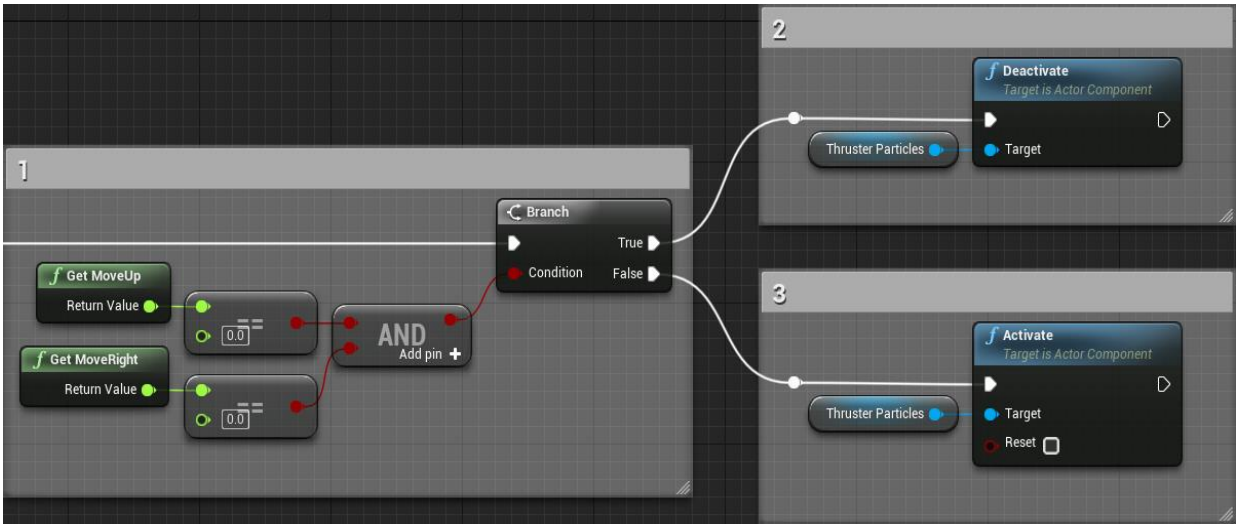

Рисунок 3.79 – Ланцюжок нодів

Давайте розберемося з тим, що робить ця схема:

Вона перевіряє прив'язки осей MoveUp і MoveRight. Якщо обидві повертають 0, то це означає, що гравець не натискає клавіш руху.

Якщо Branch повертає true (гравець не натискає клавіші руху), то деактивує ThrusterParticles

Якщо Branch повертає false (гравець натискає кнопку руху), то активується ThrusterParticles

Натисніть на Compile і закрийте BP_Player.

Далі я почну створення ефекту вибуху.

Замість створення нової системи частинок, ми дублюємо частки двигуна. Перейдіть в папку ParticleSystems, натисніть правою клавішею миші на PS_Thruster і виберіть Duplicate. Перейменуйте дублікат в PS_Explosion і відкрийте його.

Для вибуху всі частинки повинні Спауном одночасно, а не одна за одною. Цей ефект називається імпульсна випускання.

Створення імпульсу.

Для початку нам потрібно задати швидкість Спауна, рівну нулю, тому що ми не хочемо використовувати поведінку Спауна за замовчуванням. Виберіть модуль Spawn і задайте Spawn \ Rate \ Distribution \ Constant значення 0.

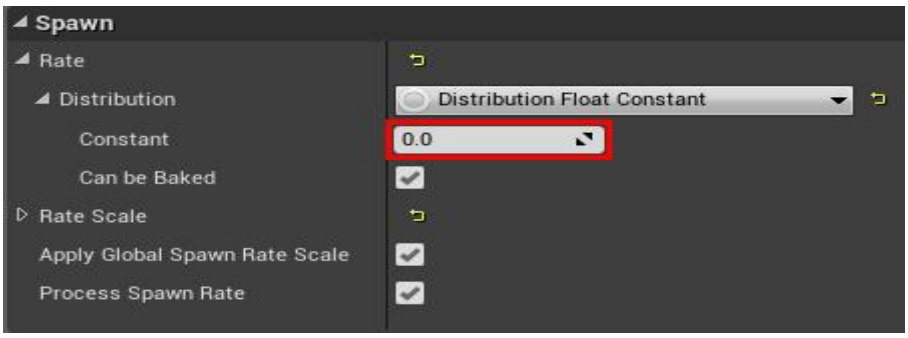

Рисунок 3.80 – Вкладка Spawn

Далі потрібно повідомити емітера, що ми хочемо створити імпульс. Прокрутіть вниз, до розділу Burst і додайте новий елемент в Burst List. Це можна зробити, натиснувши на значок +.

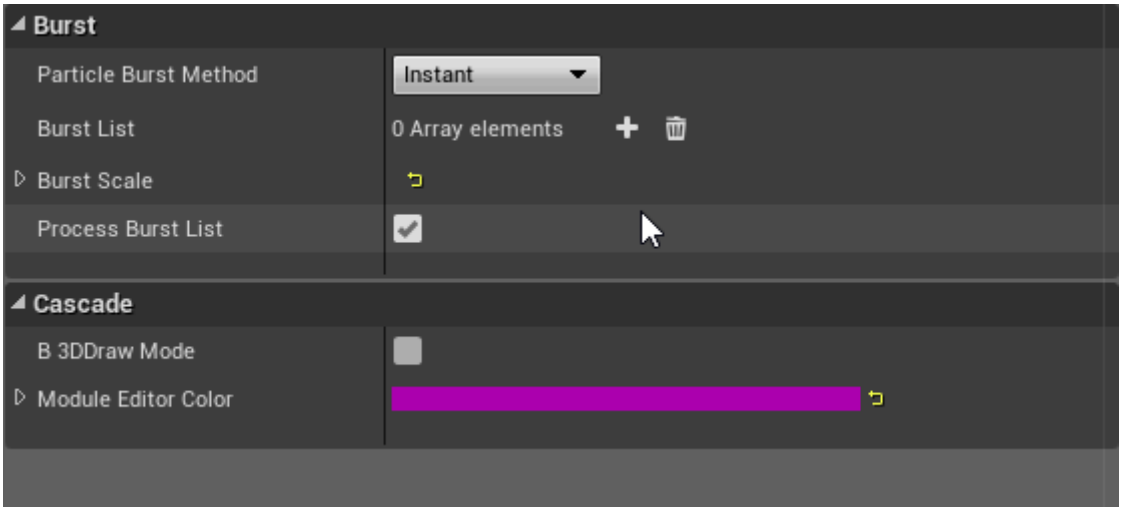

Рисунок 3.81 – Вкладка Burst

Кожен елемент буде містити три поля:

Count: кількість створюваних частинок. Вкажіть значення 20.

Count Low: якщо більше або дорівнює 0, то кількість створюваних частинок буде змінюватися в інтервалі від Count Low до Count. Залишимо тут значення -1.

Time: момент Спаун частинок. Значення 0 означає початок терміну життя емітера. Значення 1 означає кінець терміну життя емітера. Залишимо тут значення 0.0.

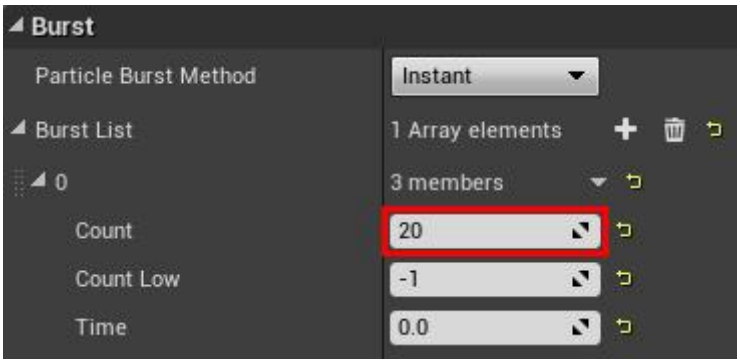

Рисунок 3.82 – Вкладка Burst

Це означає, що емітер створить 20 частинок на початку свого життя.

Щоб зробити імпульс схожим на вибух, нам потрібно додати швидкість, з якою частинки будуть розлітатися.

Створення розльоту частинок.

Оскільки гра має вигляд зверху, нам потрібно вказати тільки швидкості по X і Y. Виберіть модуль Initial Velocity і розгорніть Start Velocity \ Distribution. Задайте Max значення (1000, 1000, 0), а Min - значення (-1000, -1000, 0).

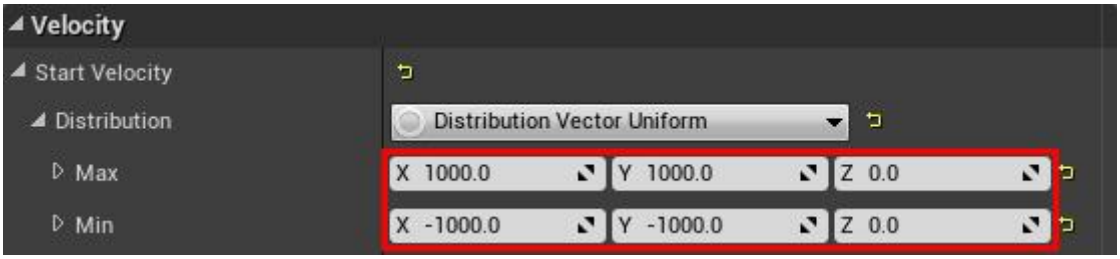

Рисунок 3.83 – Вкладка Velocity

Ми вказали інтервал від негативних до позитивних значень, тому частинки будуть розлітатися від емітера.

Тепер потрібно задати кількість повторів емітера.

Завдання повторів емітера

За замовчуванням емітери повторюються нескінченно. Це відмінно підходить для таких ефектів. як дим і вогонь, але імпульс повинен програватися тільки один раз. Щоб виправити це, нам потрібно повідомити емітера, що він повинен повторюватися тільки один раз.

Виберіть модуль Required і знайдіть розділ Duration. Задайте Emitter Loops значення 1.

Рисунок 3.84 – Вкладка Duration

Тепер ми будемо відтворювати вибух при смерті ворога.

Створення частинок при смерті противника

Поверніться в основний редатор і перейдіть в папку Blueprints. Відкрийте BP_Enemy і знайдіть подія OnDeath.

Щоб Заспаунени систему частинок, можна використовувати нод Spawn Emitter at Location. Створіть його і з'єднайте з Destroy Actor.

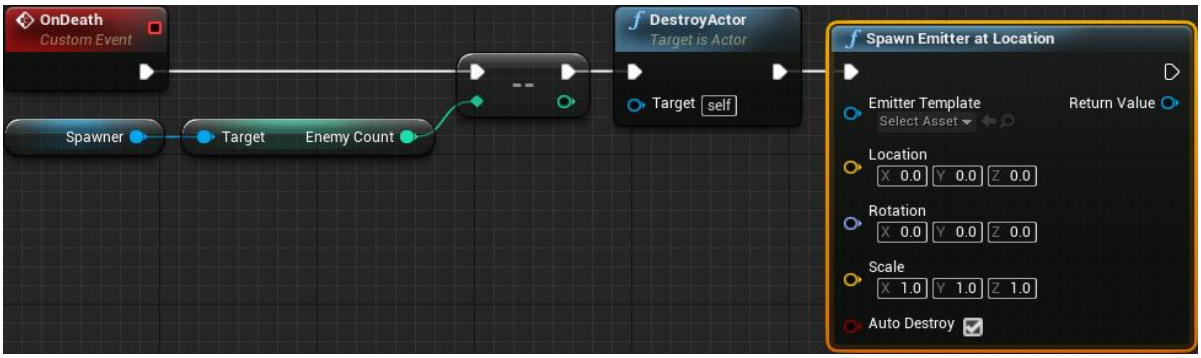

Рисунок 3.85 – Зображення події OnDeath в BP_Enemy

Потім задамо Emitter Template значення PS_Explosion.

Рисунок 3.86 – Вкладка Spawn Emitter at Location

Нарешті, створіть GetActorLocation і з'єднайте його з контактом Location.

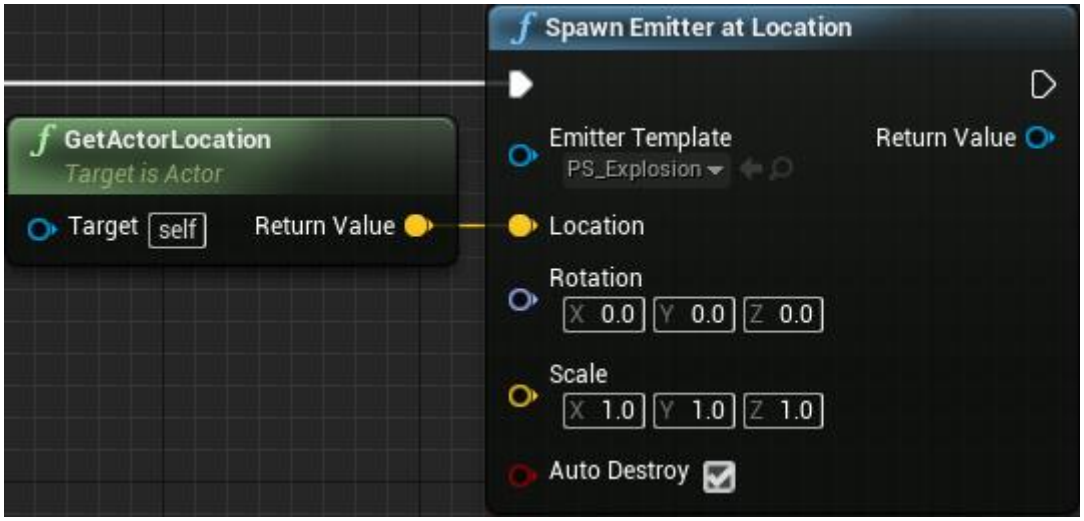

Рисунок 3.87 – З'єднання GetActorLocation з контактом Location

Тепер при смерті ворога подія буде створювати екземпляр PS_Explosion в точці розташування ворога.

Натисніть на Compile і поверніться в основний редактор.

Заміна кольору вибуху на колір ворога

Щоб скористатися кольором, нам потрібен спосіб отримання такої інформації з Blueprints. На щастя, в Cascade є тип розподілу, що дозволяє це зробити.

Відкрийте PS_Explosion і виберіть модуль Initial Color. Задайте Start Color \ Distribution значення Distribution Vector Particle Parameter.

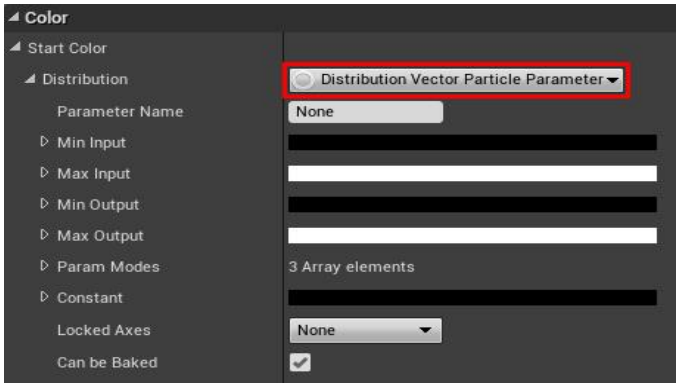

Рисунок 3.88 – Вкладка Color

Це дасть нам параметр, який ми зможемо змінювати за допомогою Blueprints. Дайте Parameter Name назву PrimaryColor.

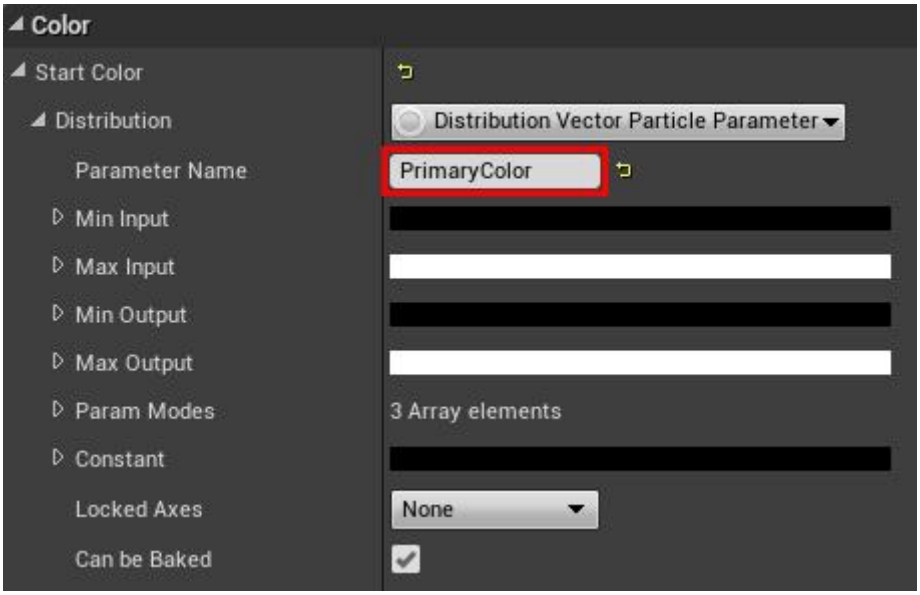

Рисунок 3.89 – Вкладка Color

Для вибуху ми будемо використовувати обидва кольору ворога. Для використання другого кольору нам потрібен ще один емітер. Клацніть правою кнопкою миші на порожньому просторі в емітер і виберіть Emitter \ Duplicate and Share Emitter. Так ми дублюємо емітер.

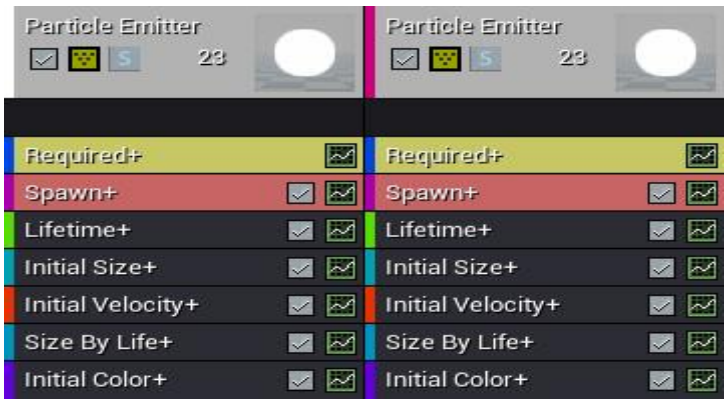

Рисунок 3.90 – Панель Particle Emitter

Ви помітите, що у кожного модуля тепер є значок +. Завдяки використанню Duplicate and Share Emitter замість Duplicate, ми зв'язали модулі, а не взяте. Всі зміни, що вносяться в один модуль, будуть відображатися на тому ж модулі іншого емітера. Це корисно, якщо ми хочемо змінювати властивості у всіх емітерах, наприклад, розмір.

Єдиний модуль, який нам потрібно змінювати - це Initial Color. Однак якщо ми внесемо зміни, то вони відіб'ються на обох емітерах. У цьому випадку нам не потрібно, щоб модулі були пов'язані, тому що їм потрібні окремі назви параметрів. Найпростіший спосіб відключення їх зв'язку полягає у видаленні дубльованого модуля Initial Color і створенні нового.

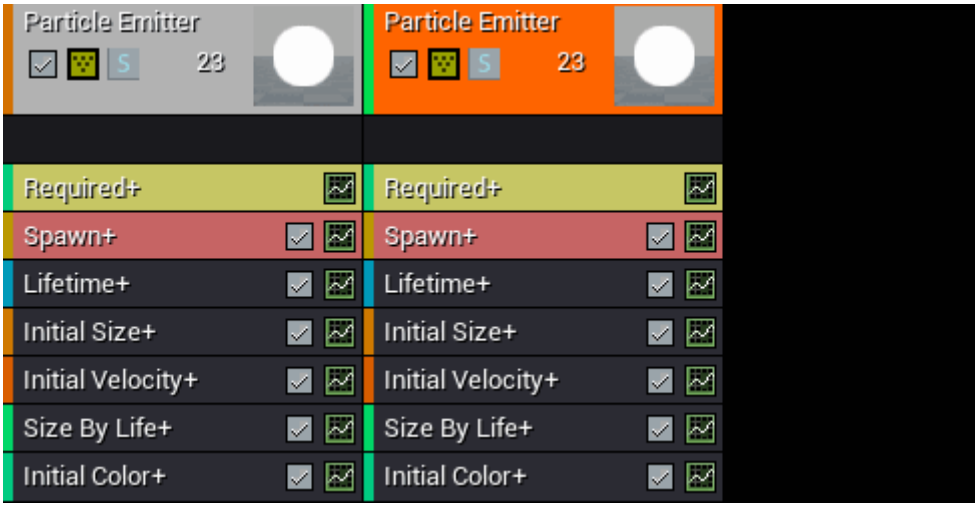

Рисунок 3.91 – Панель Particle Emitter

Виберіть новий Initial Color і задайте Start Color \ Distribution значення Distribution Vector Particle Parameter. Потім задайте Parameter Name значення SecondaryColor.

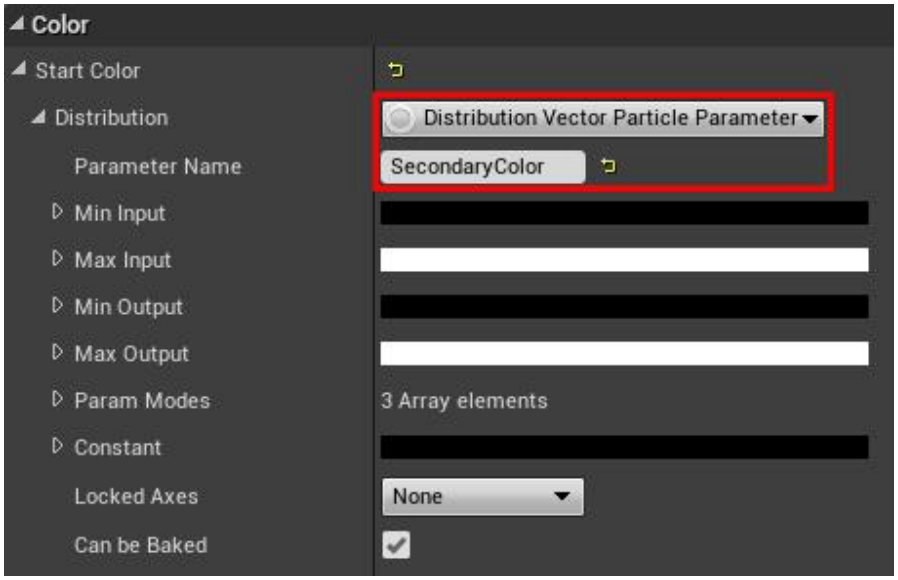

Рисунок 3.92 – Вкладка Color

На цьому етапі система частинок вже готова. Закрийте PS_Explosion.

Далі нам потрібно буде задавати параметри за допомогою Blueprints.

Завдання параметрів частинок за допомогою Blueprints

Відкрийте BP_Enemy і додайте після Spawn Emitter at Location виділені Ноди:

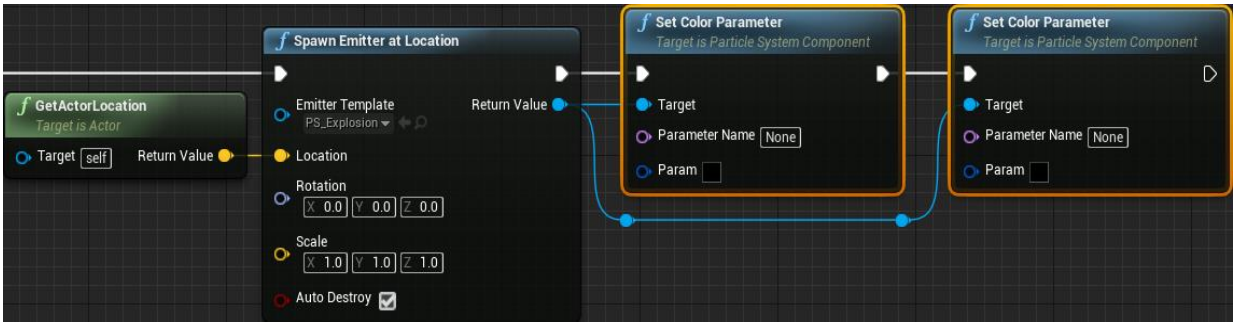

Рисунок 3.93 – Ланцюжок нодів

Це дозволити змінювати два параметра PS_Explosion.

Тепер нам потрібно дати параметрам правильні назви. Задайте в якості Parameter Name першого Set Color Parameter значення PrimaryColor. Задайте в якості Parameter Name другого Set Color Parameter значення SecondaryColor

Рисунок 3.94 – Назви параметрів в ланцюжці нодів

Нарешті, нам потрібно передати кольору. Щоб спростити рабту, ми вже зберегли кольору в змінні PrimaryColor і SecondaryColor. З'єднайте кожну змінну з відповідним нодом.

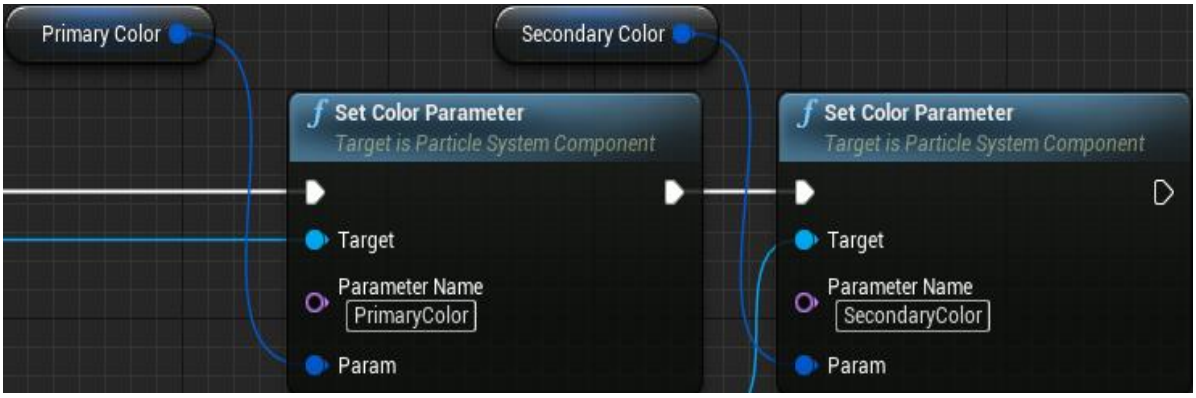

Рисунок 3.95 – З'єднання змінних в ланцюжці нодів

Ось, що у мене в результаті вийшло:

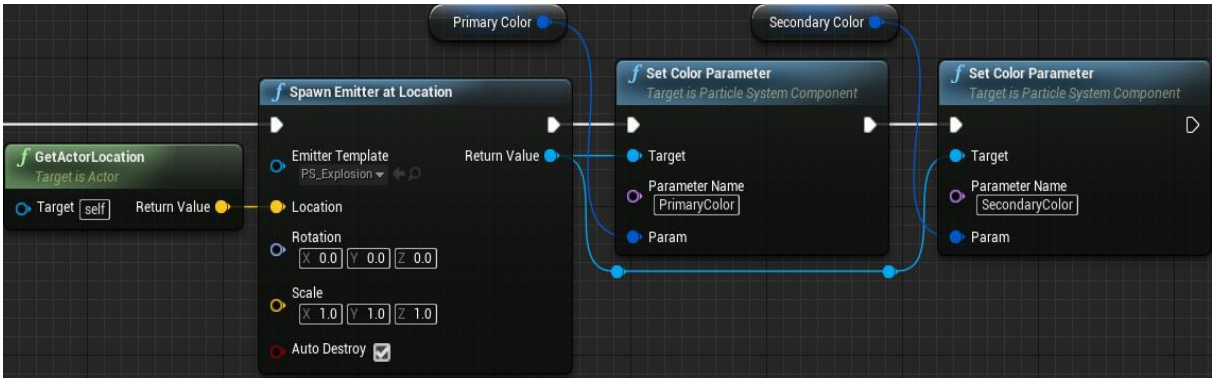

Рисунок 3.96 – Ланцюжок нодів

Давайте розберемося в події по порядку:

Коли ворог помирає, він Спауном екземпляр PS_Explosion в точці свого місця розташування

Задається значення параметра PrimaryColor PS_Explosion

Задається значення параметра SecondaryColor PS_Explosion

Натисніть на Compile і закрийте BP_Enemy.

Тепер я добавлю вибух при смерті гравця.

Відкрийте BP_Player і знайдіть подія OnDeath

Додайте нод Spawn Emitter at Location до контакту Then 1 нод Sequence. Задайте Emitter Template значення PS_Explosion.

Створіть GetActorLocation і з'єднайте його з контактом Location нода Spawn Emitter at Location

Створіть Set Color Parameter і з'єднайте його з Spawn Emitter at Location. Задайте Parameter Name значення PrimaryColor і з'єднайте змінну PrimaryColor з Param.

Створіть ще один Set Color Parameter і з'єднайте його з першим Set Color Parameter. Задайте Parameter Name значення SecondaryColor і з'єднайте змінну SecondaryColor з Param.

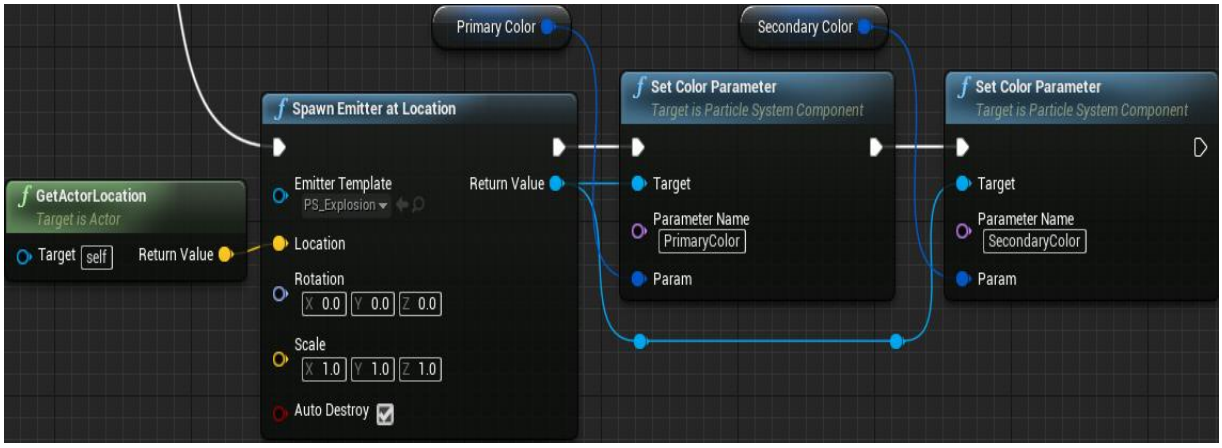

Рисунок 3.97 – Кінцевий варіант ланцюжка нодів

3.3 Керівництво користувача.

Початок нової гри проходить без екрана завантаження,гравецю під контроль дається заздалегіть створенний персонаж від третього лиця,з початковим першим рівнем. Гравець маючи під контролем персонажа має можливість: рухати свого персонажа за допомогою клавіш W,A,S,D,виконувати стрибки персонажем завдяки клавіші Space, та виконувати прості удари лівою клавішою миші завдяки чому персонаж може робити комбо з ударів.Карта гри має обсяг майже в два кілометра,що дає змогу персонажу гравця її досліджувати і використовуючи порталі гравець може миттєво переміщатись у важкодоступні місця з однієї в іншу. Контактувати з не ігровими персонажами, беручи в них завдання на вбивство ворожих не ігрових персонажів чи пошук втрачених предметів.Ворожі не ігрові персонажі(НПС) розположені в різних місцях карти з

обмеженим радіусом пересування і поля зору, в якому вони помітивши гравця розпочнуть його атакувати з підвищеною швидкістю пересування.Можливість вільно атакувати не ігрових персонажів. За вбивство ворогів гравець буде отримувати досвід який дає змогу підіймати рівень персонажа.Можливість завдяки якій персонаж гравця може підіймати сюжетні предмети.Можливість користуватись журналом завдань,а також можливість відмовитись від завдань.Також можна обрати щойно виконане завдання,що дасть змогу повторного його виконання.

4.Тестування програмного забезпечення

4.1 Вибір та обґрунтування методів тестування програмного забезпечення.

Тестування програмного забезпечення - це процес дослідження, випробування програмного продукту, який має на меті перевірку відповідності між реальною поведінкою програми і її очікуваним поведінкою на кінцевому наборі тестів, обраних певним чином.

Може оцінюватись:

- відповідність вимогам, якими керувалися проектувальники та розробники
- правильна відповідь для усіх можливих вхідних даних
- виконання функцій за прийнятний час
- практичність
- сумісність з програмним забезпеченням та операційними системами
- відповідність задачам замовника.

Оскільки число можливих тестів навіть для нескладних програмних компонентів практично нескінченне, тому стратегія тестування полягає в тому, щоб провести всі можливі тести з урахуванням наявного часу та ресурсів. Як результат програмне забезпечення тестується стандартним виконанням програми з метою виявлення багів.

Тестування ПЗ може надавати об'єктивну, незалежну інформацію про якість ПЗ, ризики відмови, як для користувачів, так і для замовників.

Тестування може проводитись, як тільки створено виконуваний код. Процес розробки зазвичай передбачає, коли та як буде відбуватися тестування. Наприклад, при поетапному процесі, більшість тестів відбувається після визначення системних вимог і тоді вони реалізуються в тестових програмах. На противагу цьому, відповідно до вимог гнучкої розробки ПЗ, програмування і тестування часто відбувається одночасно.

Верифікація - це перевірка, підтвердження, метод доказів будь-яких теоретичних положень, алгоритмів, програм і процедур шляхом їх зіставлення з досвідченими (еталонними або емпіричними) даними, алгоритмами і програмами.

Корінь різного розуміння поняття верифікація криється в спектрі можливостей звірення відповідності кінцевого продукту визначеним вимогам. Верифікувати відповідність кінцевого продукту визначеним вимогам можливо, в залежності від ситуації, за прямими і непрямими характеристиками цього кінцевого продукту. А також існує процесний підхід, який відстежує просування продукту до визначеним вимогам.

Верифікацію слід відрізняти від фальсифікації. Відповідно до принципу фальсифікації, емпірична система (гіпотеза, концепція, теорія), що не допускає спростування на підставі експерименту, не є науковою.

Валідація - це зазвичай внутрішній процес управління якістю, що забезпечує згоду з правилами, стандартами або специфікацією. Простий спосіб запам'ятати різницю між валідацією і верифікацією полягає в тому, що валідація підтверджує, що «ви створили правильний продукт», а верифікація підтверджує, що «ви створили продукт таким, яким і мали намір його зробити». Ще один приклад типової верифікації: проведення випробування обладнання. Маючи певні вимоги на руках, ми проводимо випробування продукту і фіксуємо, чи дотримані вимоги. Результат верифікації - відповідь на питання «Чи відповідає продукт вимогам?».

Але далеко не завжди продукт, відповідний встановленим вимогам, можна застосовувати в конкретній ситуації. Наприклад, ліки пройшло всі належні випробування і надійшло в продаж. Чи означає це, що воно може бути застосоване якимось конкретним хворим? Ні, тому що кожен організм має свої особливості і конкретно для нього, це ліки можуть бути згубним, тобто хтось (лікар) повинен підтвердити: так, цього хворому можна приймати ці ліки. Тобто лікар повинен виконати валідацію: надати законну силу конкретного застосування.

Мета процесу валідації — переконатися, що специфічні вимоги для програмного продукту виконано, і здійснюється це за допомогою:

- розробленої стратегії і критеріїв перевірки всіх робочих продуктів;
- обговорених дій з проведення валідації;
- демонстрації відповідності розроблених програмних продуктів вимогам замовника і правилам їхнього використання;
- узгодження із замовником отриманих результатів валідації продукту.

Процес валідації може проводитися самим виконавцем або іншою особою, наприклад, замовником, що здійснює дії з впровадженню і проведенню цього процесу за планом, у якому відбиті елементи і задачі перевірки. При цьому використовуються методи, інструментальні засоби і процедури виконання задач процесу для встановлення відповідності тестових вимог і особливостей використання програмних продуктів проекту на правильність реалізації вимог.

Верифікація і валідація полягає в перевірці специфікацій і правильності виконання програм відповідно до заданих вимог і формального опису програми.

Верифікація програмного коду допомагає зробити висновок про коректність створеної програмної системи при її проектуванні і після завершення її розроблення. Валідація дозволяє встановити здійсниміть заданих вимог шляхом їх перегляду, інспекції і оцінки результатів проектування на процесах життєвого циклу для підтвердження того, що здійснюється коректна реалізація вимог, дотримання заданих умов і обмежень до системи. Верифікація і валідація забезпечують перевірку повноти, несуперечності і однозначності специфікації і правильності виконання функцій системи.

Верифікації і валідації піддаються:

- компоненти системи, їх інтерфейси і взаємодія об'єктів у розподілених середовищах;
- описи доступу до баз даних, засоби захисту від несанкціонованого доступу до даних різних користувачів;
- документація до системи;
- тести, тестові процедури і вхідні набори даних.

На інших процесах життєвого циклу виконуються додаткові дії:

- перевірка і контроль проектних рішень за допомогою методик і процедур перегляду ходу розроблення;
- звернення до CASE-систем , що містять у собі процедури перевірки вимог до продукту;
- перегляди й інспекції проміжних результатів на відповідність вимогам для підтвердження того, що програмна система має коректну реалізацію вимог і задовольняє умови виконання.

4.2 Розробка тестових наборів данних.

Таблиця 4.3 Відображення роботи гри

Модуль додатку	Вихідні данні,необхідні для тест-кейсу	Очікуванний результат по кожному кроку тест-кейсу
Журнал	1.Зайти в гру. 2.Натиснути на клавішу J. 3.Провірити появу інтерфейса журналу. 4.Підійти до НПС та натиснути клавішу E взявши завдання. 5.Перевірити появу завдання в журналі.	1.Виконується вхід в гру. 2.Виклик клавіші J. 3.Поява інтерфейсу у вигляді книги. 4.Виконується пошук квестового НПС. 5.В журнал додається завдання з його описом.

Продовження таблиці 4.3.

Інтерфейс характеристик персонажа	1.Перевірити наявність іконки персонажа та характеристик під нею.	1.Гравець бачить інформацію про персонажа в куті екрану гри.
Віднімання балів здоров'я.	1.Підійти персонажем до ворожого НПС. 2.Очікувати атаки НПС на гравця. 3.Провірити кількість віднімання здоров'я.	1.Виконується вхід у діапазон видимості НПС. 2.Ворожий НПС напрвляється до гравця з більшою швидкістю та атакує. 3.Шкала здоров'я зменшується.

4.3Аналіз результатів тестування.

Таблиця 4.4 Результат тестування гри

Дія	Реакція програми	Результат
1.Зайти в гру. 2.Натиснути на клавішу J. 3.Провірити появу інтерфейса журналу. 4.Підійти до НПС та натиснути клавішу E взявши завдання. 5.Перевірити появу завдання в журналі.	1.Виконується вхід в гру. 2.Виклик клавіші J. 3.Поява інтерфейсу у вигляді книги. 4.Виконується пошук квестового НПС. 5.В журнал додається завдання з його описом.	Правильно

Продовження таблиці 4.4.

1.Перевірити наявність іконки персонажа та характеристик під нею.	1.Гравець бачить інформацію про персонажа в куті екрану гри.	Правильно
1.Підійти персонажем до ворожого НПС. 2.Очікувати атаки НПС на гравця. 3.Провірити кількість віднімання здоров'я.	1.Виконується вхід у діапазон видимості НПС. 2.Ворожий НПС направляється до гравця з більшою швидкістю та атакує. 3.Шкала зменшується.	Правильно

Висновки

В ході представленої роботи було досліджено місце комп’ютерних ігор в сьогоднішньому світі. При розробці комп’ютерної гри було розглянуто існуючі ігрові рушії – для розробки комп’ютерної гри. Було виявлено найбільш найпопулярніші та найбільш потужні ігрові рушії, якими стали Unity та Unreal Engine 4. Було проведено їх порівняння і обрані найбільш актуальні засоби розробки для комп’ютерної гри а також було проаналізовано кожен із них, чим вони кращі та які в них недоліки. При аналізі можливостей кожного з ігрових рушіїв було створено його абстрактний рейтинг по відношенню до поставленої задачі гри.

Вибір пріоритетних засобів розробки проходив за двома критеріями: доступність і функціональність. В ході аналізу стало ясно, що при розробці комп'ютерної гри з простою ігровий механікою, варто звернути увагу на додаткові елементи гри, такі як сюжет і графічне оформлення. Це потрібно для того, що б утримати потенційного гравця і продовжити життєвий цикл розробки. На основі цього було зроблено висновок, що Unreal Engine 4 найкраще підходить для розробки комп’ютерної гри, з недоліком у тому, що в даний ігровий рушій має доволі високий поріг входу, проте ефективність ігрового рушія перекриває цей недолік. Після вибору засобів розробки було розпочато вивчення Unreal Engine, а також розробка самої гри. В ході розробки був вивчений ігровий рушій Unreal Engine і були придбані необхідні знання та вміння. Освоєння середовища розробки Unreal Engine несе не маловажний характер, так як в сучасному світі індустрія розробки ігор все сильніше поширюється в нашому суспільстві. Ігри перестали бути лише предметом для розваг, і тепер використовуються і в інших областях, наприклад, в науці або в навчанні користувачів. Тому розвиток в даному напрямку можна вважати одним з найбільш важливих в сучасному суспільстві. В ході реалізації проекту були виконані наступні завдання: 1) Аналіз ігрових рушіїв; 2) Проектування комп’ютерної гри; 3) Розробка комп’ютерної гри; 4) Тестування комп’ютерної гри;

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. The Elder Scrolls 5: Skyrim в порівнянні з The Witcher 3 - найкращий RPG [Електронний ресурс] – Режим доступу до ресурсу: https://www.vg247.com/2016/10/26/skyrim-versus-the-witcher-3-which-is-the-best-rpg/.
2. Матеріал по Unreal Engine Знайомство з движком і його основними функціями [Електронний ресурс] – Режим доступу до ресурсу : https://habr.com/ru/post/344394/.
3. Матеріал по Unreal Engine Blueprints та його функції [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/post/344446/
4. Матеріал по Unreal Engine.Звук та основи роботи з ним [Електронний ресурс] – Режим доступу до ресурсу – https://habr.com/ru/post/345018/.
5. Матеріал по Unreal Engine.Анімації та робота з ним [Електронний ресурс] – Режим доступу до ресурсу –https://habr.com/ru/post/344840/.
6. Матеріал по Unreal Engine.Інтерфейс та робота з ним [Електронний ресурс] – Режим доступу до ресурсу – https://habr.com/ru/post/344600/.
7. Unreal Engine [Електронний ресурс]– Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Unreal_Engine.
8. Welcome to unreal engine 4 [Електронний ресурс] – Режим доступу до ресурсу –https://www.unrealengine.com/en-US/blog/welcome-to-unreal-engine-4
9. Изучаем C++ создавая игры в UE4 [Електронний ресурс] – Режим доступу до ресурсу – https://codernet.ru/books/c_plus/izuchaem_c_sozdavaya_igry_v_ue4/
10. Unreal Engine 4.x Scripting with C++ Cookbook - Second Edition By John P. Doran, William Sherif, Stephen Whittle
11. Mastering Game Development with Unreal Engine 4 - Second Edition By Matt Edmonds
12. Unreal Engine C++ Developer: Learn C++ and Make Video Games [Електронний ресурс] – Режим доступу до ресурсу – https://www.udemy.com/course/unrealcourse/

13. Створення гри на Unreal Engine 4 [Електронний ресурс] – Режим доступу до ресурсу – https://habr.com/ru/post/327372/
14. Уроки по Unreal Engine 4 [Електронний ресурс] – Режим доступу до ресурсу – https://itproger.com/course/unreal-engine
15. Unity3D или Unreal Engine 4 [Електронний ресурс] – Режим доступу до ресурсу – https://stfalcon.com/ru/blog/post/unity3d-vs-unreal-engine-4
16. Створення гри в Unreal Engine 4 [Електронний ресурс] – Режим доступу до ресурсу – https://stdpub.com/unrealengine/kak-sozdat-prostuyu-igru-v-unreal-engine-4
17. Unreal Engine 4 Blueprints Tutorial [Електронний ресурс] – Режим доступу до ресурсу –https://stdpub.com/unrealengine/unreal-engine-4-blueprints-tutorial
18. Unreal Engine 4: использование материалов [Електронний ресурс] – Режим доступу до ресурсу – https://stdpub.com/unrealengine/unreal-engine-4-ispolzovanie-materialov
19. Руководство по пользовательскому интерфейсу Unreal Engine 4 [Електронний ресурс] – Режим доступу до ресурсу – https://stdpub.com/unrealengine/rukovodstvo-po-polzovatelskomu-interfejsu-unreal-engine-4
20. Unreal Engine 4: учебное пособие по созданию анимации – Режим доступу до ресурсу [Електронний ресурс] – https://stdpub.com/unrealengine/unreal-engine-4-uchebnoe-posobie-po-sozdaniyu-animaczii
21. Unreal Engine 4 Tutorial: работа с Искусственным Интеллектом – Режим доступу до ресурсу [Електронний ресурс] – https://stdpub.com/unrealengine/unreal-engine-4-tutorial-rabota-s-iskusstvennym-intellektom

Додаток А

Розробка системи відкритого світу в Unreal Engine 4

Локазюк В.Ю.
Науковий керівник – к.т.н.,доц. Медзатий Д.М.
Хмельницький національний університет

Перша версія движка Unreal Engine з'явилася в далекому 1998 році, коли компанія Epic Games випустила шутер Unreal. Уже тоді він демонстрував універсальність, поєднуючи в собі графічний і фізичний движки, систему штучного інтелекту, управління файлової і мережевий системами, а також включаючи готову середу для розробки ігор. Автори движка спростили взаємодія з ним, щоб розробники могли зосередитися на створенні основних елементів ігор, не відволікаючись на дрібниці на зразок налагодження мережевого коду або обчислення колізій. Розвиток Unreal Engine йшло поступово, рік за роком: движок міняв версії, обростав новими технологіями - багато в чому, до речі, революційними для свого часу. Кожна версія UE привносила нові вражаючі графічні ефекти, показання Epic Games в високотехнологічних демо (на кшталт «Самарянина»). Завдяки простоті використання, а також лояльним умовам ліцензування, движок використовували багато студій, від інді-команд до найбільших компаній, що випускають дорогі AAA-блокбастери.

Станом на 2020 рік Unreal Engine (вже в четвертій номерний версії) залишається надзвичайно затребуваним движком, дуже гнучким, зручним у використанні і абсолютно різноплановим - поставте поруч гранично похмуру RTS Battlefleet Gothic: Armada і веселу «Королівську битву» Fortnite, і ви ні за не здогадаєтеся, що обидві гри виконані на одному двигуні. Epic Games продовжує регулярно оновлювати його, використовуючи, в тому числі, і для своїх проектів, включаючи вищезгадану мегапопулярну Fortnite.

Як вже говорилося, Unreal Engine дуже гнучкий і універсальний. На відміну від Unity, який вимагає установки безлічі плагінів (часто - платних), UE4 вже «з коробки» забезпечений всіма необхідними інструментами розробки. Користувачі, які знають C ++, освоять движок швидше, адже він використовує саме цю мову програмування. Втім, для тих, хто не знайомий з C ++, Unreal Engine теж піддасться без проблем - все завдяки візуальному редактору Blueprints, який дозволяє створювати скрипти і розміщувати об'єкти, не написавши жодного рядка коду.

До того ж, движок містить безліч інструментів, які полегшують роботу з ним. Наприклад, підтримує безліч форматів текстур, точно передає фізичні властивості матеріалів, дозволяє змінювати об'єкти в реальному часі, задавати для них функції і коментарі, автоматично вибрати джерела освітлення, додати туман і інші ефекти, і так далі. Велику колекцію Ассет (платних і безкоштовних) можна використовувати при розробці ігор, а відкритий вихідний код движка дає можливість вносити в нього зміни при

необхідності. Движок гнучко підлаштовується під платформу розробки, що дозволяє оптимізувати гри під консолі, мобільні гаджети та ПК.

Epic Games стабільно тримає лідерські позиції в області передових технологій, тому Unreal Engine 4 напханий ними під зав'язку, забезпечуючи, при належному майстерності розробників відеогри, вражаючу візуальну складову з якісним освітленням (трасування променів в реальному часі, звичайно ж, підтримується), м'якими тінями, чесними відображеннями, достовірної анімацією персонажів і іншими ефектами. За рахунок цього, до речі, движок також використовується в створенні комп'ютерної графіки в кіноіндустрії - наприклад, його силами було створено дроїд K-2SO для «ізгоя-1» сцени з «У пошуках Дорі» і «Мандалорца».

Движок ідеально підходить для тривимірних AAA-проектів, особливо шутерів і пригодницьких екшенів, що підтверджується списком ігор, розроблених на Unreal Engine (причому всіх версій, а не тільки четвертої), в якому домінують представники саме цих жанрів. Ми відзначали, що UE створений для професіоналів, і в руках досвідченої команди цей движок здатний показати запаморочливу картинку.

Відкритий світ - місцевість без входу і виходу. Зазвичай це місто, острів або поселення, при спробах вибратися з яких персонаж наштовхується на стіну, пробку або щось подібне, що заважає перетнути кордон світу. В кінці-кінців, гравцеві це набридає, а при спробах повністю пройти карту він раз у раз знаходить нові цікаві мікролокації або квести, тому бажання знайти кінець світу відходить на другий план.

Завдання геймдизайнера при створенні безшовного світу - розробити локацію так, щоб вона повертала гравця до гри, а не супроводжувала його пошуків кінця карти. Тому міста можуть бути закільцьовані - всі вулиці будуть вести один на одного, а основна гра відбуватиметься десь в центрі карти.

У той же час безшовний світ відкриває нові можливості для досліджень місцевості. Наприклад, вентиляційні ходи і шахти в багатьох іграх з відкритим світом знаходяться в постійній досяжності. Дослідження їх захоплює людину і дає йому шлях безпечного переміщення між ділянками ігрової карти, що особливо важливо, якщо в грі часто ведуться поєдинки.

Незважаючи на те, що в реальному житті вентиляція є далеко не скрізь, а людина середньої комплекції навряд чи зміг би вільно по ній переміщатися, така опція в грі не здається дивною, а навпаки, додає ефект занурення. Це стосується і інших способів переміщення у відкритому безшовному світі.

Використання реалістичних локацій також йде на користь занурення. Театр, торговий центр, концертний зал - місця, в яких людина неодноразово бував в життя. Зберігаючи логіку будови локації, ви позбавляєте гравця від втрати часу на вивчення карти: опинившись в знайомої місцевості, простіше діяти інтуїтивно.

Основною складністю в геймдізайна є дотримання балансу між свободою відкритого світу і структурою драматичного сюжету. Оскільки гравці можуть виконувати дії, про які геймдизайнер не припускав, сценаристи гри повинні придумати дотепні способи нав'язати гравцеві сюжет, не перешкоджаючи при цьому його волі. Як наслідок, в іграх з відкритим світом іноді сюжет гри може бути розділений на серію завдань, також сюжет може бути представлений в досить спрощеному вигляді. У деяких іграх побічні завдання, пропоновані гравцеві, не припиняють виконання основного завдання. У більшості ігор з відкритим світом новостворений персонаж не має будь-яких виражених особливостей, щоб гравець міг самостійно втілювати в ньому свої задуми. Проте, в деяких іграх на зразок Landstalker пропонується розширений набір діалогів при створенні персонажа. У 2005 році Девід Бребен так описав структуру оповіді в комп'ютерних іграх: «мало відрізняється від фільмів з Гарольдом Ллойдом 1920-х років» і щиро вважав, що сюжети з відкритим фіналом стануть «Святим Граалем, який ми шукаємо в п'ятому поколінні комп'ютерних ігор». Геймдизайнер Манвір Хейр (англ. Manveer Heir), який працював в Electronic Arts над іграми Mass Effect 3 і Mass Effect: Andromeda, говорив, що в геймдізайн ігор з відкритим світом присутні складності в порівнянні з лінійним геймдізайн, оскільки досить складно передбачити підхід гравців до вирішення задач, поставлених перед ними грою, це повинно бути рушійною силою розробки з самого початку. На думку Хейр, критичні недоліки в Andromeda стали наслідком того, що відкритий світ був введений в геймплей на занадто пізній стадії розробки.

Відкритий світ може являти собою простір без будь-яких заздалегідь заданих меж і орієнтирів, як в Minecraft, але в багатьох випадках розробники вважають за краще «підштовхувати» гравця до створеним для нього цікавих місцях і завданням, змусивши його зосередитися на певній галузі світу. Одним із способів це зробити є поділ світу на регіони з визначеними кордонами, так, щоб гравець освоював їх послідовно. При цьому гра може пропонувати гравцеві спосіб відкрити всі пункти інтересів в певному регіоні відразу, в нагороду за виконання якогось завдання або випробування

Перелік посилань

1. Движок unreal engine 4 – Режим доступу до ресурсу – https://cubiq.ru/dvizhok-unreal-engine/

2. Level design в комп'ютерних іграх: основи створення безшовні світу – Режим доступу до ресурсу – https://koloro.ua/blog/3d-tekhnologii/level-design-v-kompyuternykh-igrakh-osnovy-sozdaniya-besshovnogo-mira.html

3. Unreal Engine 4.x Scripting with C++ Cookbook - Second Edition By John P. Doran, William Sherif, Stephen Whittle Mastering Game Development with Unreal Engine 4 - Second Edition By Matt Edmonds

Додаток Б

```
import os
import random
import shutil
import tarfile
import cv2 as cv
import numpy as np
import scipy.io
from tqdm import tqdm
def ensure_folder(folder):
if not os.path.exists(folder):
os.makedirs(folder)
def save_train_data(fnames, labels, bboxes):
src_folder = 'cars_train'
num_samples = len(fnames)
train_split = 0.8
num_train = int(round(num_samples * train_split))
train_indexes = random.sample(range(num_samples), num_train)
for i in tqdm(range(num_samples)):
fname = fnames[i]
label = labels[i]
(x1, y1, x2, y2) = bboxes[i]
src_path = os.path.join(src_folder, fname)
src_image = cv.imread(src_path)
height, width = src_image.shape[:2]
# margins of 16 pixels
margin = 16
x1 = max(0, x1 - margin)
y1 = max(0, y1 - margin)
x2 = min(x2 + margin, width)
y2 = min(y2 + margin, height)
# print("{} -> {}".format(fname, label))
if i in train_indexes:
```

```
dst_folder = 'data/train'
else:
dst_folder = 'data/valid'
dst_path = os.path.join(dst_folder, label)
if not os.path.exists(dst_path):
os.makedirs(dst_path)
dst_path = os.path.join(dst_path, fname)
crop_image = src_image[y1:y2, x1:x2]
dst_img = cv.resize(src=crop_image, dsize=(img_height, img_width))
cv.imwrite(dst_path, dst_img)
def save_test_data(fnames, bboxes):
src_folder = 'cars_test'
dst_folder = 'data/test'
num_samples = len(fnames)
for i in tqdm(range(num_samples)):
fname = fnames[i]
(x1, y1, x2, y2) = bboxes[i]
src_path = os.path.join(src_folder, fname)
src_image = cv.imread(src_path)
height, width = src_image.shape[:2]
# margins of 16 pixels
margin = 16
x1 = max(0, x1 - margin)
y1 = max(0, y1 - margin)
x2 = min(x2 + margin, width)
y2 = min(y2 + margin, height)
# print(fname)
dst_path = os.path.join(dst_folder, fname)
crop_image = src_image[y1:y2, x1:x2]
dst_img = cv.resize(src=crop_image, dsize=(img_height, img_width))
cv.imwrite(dst_path, dst_img)
def process_train_data():
print("Processing train data...")
cars_annos = scipy.io.loadmat('devkit/cars_train_annos')
annotations = cars_annos['annotations']
```

```
annotations = np.transpose(annotations)
fnames = []
class_ids = []
bboxes = []
labels = []
for annotation in annotations:
bbox_x1 = annotation[0][0][0][0]
bbox_y1 = annotation[0][1][0][0]
bbox_x2 = annotation[0][2][0][0]
bbox_y2 = annotation[0][3][0][0]
class_id = annotation[0][4][0][0]
labels.append('%04d' % (class_id,))
fname = annotation[0][5][0]
bboxes.append((bbox_x1, bbox_y1, bbox_x2, bbox_y2))
class_ids.append(class_id)
fnames.append(fname)
labels_count = np.unique(class_ids).shape[0]
print(np.unique(class_ids))
print('The number of different cars is %d' % labels_count)
save_train_data(fnames, labels, bboxes)
def process_test_data():
print("Processing test data...")
cars_annos = scipy.io.loadmat('devkit/cars_test_annos')
annotations = cars_annos['annotations']
annotations = np.transpose(annotations)
fnames = []
bboxes = []
for annotation in annotations:
bbox_x1 = annotation[0][0][0][0]
bbox_y1 = annotation[0][1][0][0]
bbox_x2 = annotation[0][2][0][0]
bbox_y2 = annotation[0][3][0][0]
fname = annotation[0][4][0]
bboxes.append((bbox_x1, bbox_y1, bbox_x2, bbox_y2))
fnames.append(fname)
```

```
save_test_data(fnames, bboxes)
if __name__ == '__main__':
# parameters
img_width, img_height = 224, 224
print('Extracting cars_train.tgz...')
if not os.path.exists('cars_train'):
with tarfile.open('cars_train.tgz', "r:gz") as tar:
tar.extractall()
print('Extracting cars_test.tgz...')
93
if not os.path.exists('cars_test'):
with tarfile.open('cars_test.tgz', "r:gz") as tar:
tar.extractall()
print('Extracting car_devkit.tgz...')
if not os.path.exists('devkit'):
with tarfile.open('car_devkit.tgz', "r:gz") as tar:
tar.extractall()
cars_meta = scipy.io.loadmat('devkit/cars_meta')
class_names = cars_meta['class_names'] # shape=(1, 196)
class_names = np.transpose(class_names)
print('class_names.shape: ' + str(class_names.shape))
print('Sample class_name: [{}]'.format(class_names[8][0][0]))
ensure_folder('data/train')
ensure_folder('data/valid')
ensure_folder('data/test')
process_train_data()
process_test_data()
# clean up
shutil.rmtree('cars_train')
shutil.rmtree('cars_test')
# shutil.rmtree('devkit')
```

Додаток В

```
import keras
from resnet_152 import resnet152_model
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import CSVLogger, ModelCheckpoint, EarlyStopping
from keras.callbacks import ReduceLROnPlateau
img_width, img_height = 224, 224
num_channels = 3
train_data = 'data/train'
valid_data = 'data/valid'
num_classes = 196
num_train_samples = 6549
num_valid_samples = 1595
verbose = 1
batch_size = 16
num_epochs = 100000
patience = 50
if __name__ == '__main__':
# build a classifier model
model = resnet152_model(img_height, img_width, num_channels, num_classes)
# prepare data augmentation configuration
train_data_gen = ImageDataGenerator(rotation_range=20.,
width_shift_range=0.1,
height_shift_range=0.1,
zoom_range=0.2,
horizontal_flip=True)
valid_data_gen = ImageDataGenerator()
# callbacks
tensor_board = keras.callbacks.TensorBoard(log_dir='./logs', histogram_freq=0,
write_graph=True, write_images=True)
log_file_path = 'logs/training.log'
csv_logger = CSVLogger(log_file_path, append=False)
early_stop = EarlyStopping('val_acc', patience=patience)
reduce_lr = ReduceLROnPlateau('val_acc', factor=0.1, patience=int(patience / 4), verbose=1)
```

```
trained_models_path = 'models/model'
model_names = trained_models_path + '.{epoch:02d}-{val_acc:.2f}.hdf5'
model_checkpoint = ModelCheckpoint(model_names, monitor='val_acc', verbose=1,
save_best_only=True)
callbacks = [tensor_board, model_checkpoint, csv_logger, early_stop, reduce_lr]
# generators
train_generator = train_data_gen.flow_from_directory(train_data, (img_width, img_height),
batch_size=batch_size,
class_mode='categorical')
valid_generator = valid_data_gen.flow_from_directory(valid_data, (img_width, img_height),
batch_size=batch_size,
class_mode='categorical')
# fine tune the model
model.fit_generator(
train_generator,
steps_per_epoch=num_train_samples / batch_size,
validation_data=valid_generator,
validation_steps=num_valid_samples / batch_size,
epochs=num_epochs,
callbacks=callbacks,
verbose=verbose)
```

Додаток Г

Дипломний проект

Розробка комп'ютерної гри в Unreal engine 4.

Виконав студент ПМм-19-1
Локазюк В.Ю.
Керівник К. Т. Н. Доцент
Медзатий Д.М

Об’єкт, предмет дослідження

Об’єктом дослідження: є розробка комп’ютерної гри в Unreal Engine 4.

Предметом дослідження: є метод розробки рпг-системи і системи відкритого світу.

Ціль та інші задачі

- Вивчення та дослідження уже існуючих та систем які використовуються для удосконалення відкритого світу;
- Вивчення та дослідження уже існуючих RPG;
- Вивчення та дослідження архітектури Unreal engine та RPG;
- Впровадити свою концепцію та бачення розробки RPG команді;
- Проектування архітектури гри на ігровому рушію;
- Визначити доцільність використання вже готових модулів для впровадження своїх власних ідей, які для цього не призначені;
- Розробка комп’ютерної гри;
- Розробка системи для відкритого світу;

Мета

Розробити комп'ютерну гру на базі Unreal engine. Яка задовільняє наступним вимогам: проводити вільний час користувачу в грі по типу жанра Action-RPG де можна: керувати персонажем,брати завдання,відкривати журнал,керувати списком завдань,вступати в бій з вороже налаштованими не ігровими персонажами, подорожувати по відкритій карті з вбудованою системою відкритого світу, підіймати свій рівень персонажа, використовуючи ПК платформу для запуска гри з мінімальними системними вимогами.


GeForce Experience
Снимок экрана
Фоторежим
Фильтр
Мгновенный повтор
Запись
ПРЯМАЯ трансляция
Галерея
Content
MegaGameMusicCollection
Add New
Import
Save All

Висновки

За результатом виконаних теоретичних та практичних досліджень у дипломній роботі було розроблено комп'ютерну гру в Unreal engine 4 з системою відкритого світу та удосконаленою рпг-системою.

Зокрема було розроблено керування моделями персонажів та створення завдань для гравця, а також було створено інтерфейс гравця з журналом завдань.

За темою дослідження опубліковано тези Локазюк В.Ю.

Розробка системи відкритого світу в Unreal Engine 4 //

«Інтелектуальний потенціал – 2020» - збірник наукових праць молодих науковців і студентів - Хмельницький: ПВНЗ УЕП, 2020. – Частина 1.

UNICHECK

Имя пользователя:
Kafedra TMIT KhNU

ID проверки:
1005430201

Дата проверки:
11.12.2020 11:02:46 EET

Тип проверки:
Doc vs Internet

Дата отчета:
11.12.2020 11:12:54 EET

ID пользователя:
100005657

Название файла: Локазюк ПМм-19-1(повторно)

Количество страниц: 126 Количество слов: 21466 Количество символов: 157932 Размер файла: 2.99 MB ID файла: 1005721657

2449 слов помечены как "исключенные" и не учитываются в подсчете слов

Обнаружены модификации текста (могут влиять на процент совпадений)

16.6%
Совпадения

Наибольшее совпадение: 8.5% с Интернет-источником (https://ela.kpi.ua/bitstream/123456789/26930/1/Markov_magist..

16.6% Источники из Интернета | 140 ..... Страница 128

Поиск совпадений с Библиотекой не производился

0.24% Цитат

Цитаты | 6 ..... Страница 129

Не найдено ни одной ссылки

0% Исключений

Нет исключенных источников

Модификации

Обнаружены модификации текста. Подробная информация доступна в онлайн-отчете.

Замененные символы | 452

Подозрительное форматирование | 31 страница

Anti-Plagiarism v-15.257

Максимальное совпадение с одним документом 17.0%

Словари проверки: en_US, ru_RU, ua_UA. **Ошибок в документах: 12%**

ID: 83529 Название: Розробка комп'ютерної гри в Unreal Engine Добавлено в БД: 2020-12-10 Авторы: Локазюк Владислав Юрійович Руководители: Медзатий Дмитро Миколайович Консультанты: Опоненты:	Документ		Суммарное совпадение по Базе Данных	
	Символы	Лексемы	Символы	Лексемы
	121899	1898	26338 (22%)	430 (23%)

Источник плагиата

ID	Описание	Наличие плагиата в документе	
		Символы	Лексемы
82100	Название: Розробка комп'ютерної гри в Unreal Engine Добавлено в БД: 2020-12-02 Авторы: Локазюк Владислав Юрійович Руководители: Медзатий Дмитро Миколайович Консультанты: Опоненты:	20491 (17.0%)	354 (19.0%)
81301	Название: Розробка комп'ютерної гри в Unreal Engine Добавлено в БД: 2020-11-25 Авторы: Локазюк Владислав Юрійович Руководители: Медзатий Дмитро Миколайович Консультанты: Опоненты:	19880 (16.0%)	344 (18.0%)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА ДИПЛОМНУ РОБОТУ

освітнього ступеня «магістр»

Магістр Локазюк Владислав Юрійович

Тема Розробка комп'ютерної гри в Unreal Engine

Спеціальність 113 «Прикладна математика»
спеціалізація «Математика та статистика»

Обсяг дипломної роботи освітнього ступеня «магістр»:

кількість листів креслень _____; кількість сторінок записки 124

1. Короткий зміст ДР та прийнятих рішень В рамках магістерської роботи проведено аналіз підходів до вирішення та реалізації розробки комп`ютерної гри. Розроблено власну систему відкритого світу та удосконалено рпг-систему на базі Unreal Engine 4. На основі Unreal Engine 4 було розроблено комп`ютерну гру.

2. Висновок про відповідність ДР дипломному завданню Дипломна робота освітнього ступеня «магістр» у повній мірі відповідає поставленому завданню як в теоретичній, так і в практичній частині дипломної роботи.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У вступі обґрунтовується актуальність теми роботи, дається аналіз досліджуваної проблеми і обґрунтовується застосовуваний підхід до її вирішення, формулюються цілі і завдання дослідження, описується наукова новизна і практична значимість отриманих результатів. У першому розділі якісно та в повній мірі проаналізовано вибір ігрових рушиїв та їх варіантів для розробки комп'ютерної гри. Наступні розділи присвячені розробці архітектурі гри, розробці системи відкритого світу та удосконаленю рпг-системи. Розроблено програмне забезпечення для комп'ютерної та проведене її тестування.

4. Позитивні сторони проекту Дипломна робота містить ряд інноваційних рішень, зокрема, розроблено систему відкритого світу та удосконалено рпг-систему, що дозволить гравцю насолодитися відкритим світом в грі та якісно і весело провести свій час.

5. Негативні сторони проекту Гра займає багато місця на жорсткому диску і має великі технічні вимоги від комп'ютера корустувача.

6. Оцінка графічного оформлення та пояснювальної записки роботи Графічне оформлення виконане відповідно до теми дипломної роботи з дотриманням стандартів. В загальному графічне оформлення виконане на достатньому рівні. Пояснювальна записка відповідає нормам для її оформлення.

7. Відгук про роботу в цілому В загальному дипломна робота заслуговує позитивної оцінки. Весь матеріал дипломної роботи структурований, чіткий та послідовний. Усі розділи роботи послідовні та логічні, що дозволяє чітко розуміти викладений матеріал в рамках тематики дипломної роботи. Графічний матеріал дозволяє наочно побачити доцільність та ефективність рішень, які були прийняті за основу для досягнення поставленої задачі.

8. Інші зауваження.

9. Оцінка дипломної роботи Розглянувши позитивні та негативні сторони представленої дипломної роботи, можна зробити висновок, що вона заслуговує оцінку «добре».

РЕЦЕНЗЕНТ (прізвище, ім'я, по батькові, посада, місце роботи)
Гурман Іван Васильович, доцент кафедри інженерії програмного забезпечення.

«10» 12 2020 . Гурман І.В. (підпис)

ХНУ. Зам. 54, тир. 6000, 2007

Завідувачу кафедри ТМІТ
_д-р.техн.наук Підченку С.К_____
здобувача вищої освіти
_Локазюка Владислава Юрійовича__
_ФПКТС, 2 курсу, ПМм-19-1_____

ЗАЯВА

З правилами чинного Положення «Про дотримання академічної доброчесності в Хмельницькому національному університеті» від 26.09.2020 (зі змінами від 26.11.2020), згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений (а). Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений (а) та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

25.11.2020
дата

підпис

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ

КАФЕДРИ ТЕЛЕКОМУНІКАЦІЙ, МЕДІЙНИХ ТА ІНТЕЛЕКТУАЛЬНИХ ТЕХНОЛОГІЙ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Розробка комп'ютерної гри в Unreal Engine

Автор: Локазюк Владислав Юрійович

Спеціальність: 113 – прикладна математика

Освітня програма: освітньо-професійна

Науковий керівник: Медзатий Дмитро Миколайович, к.т.н доцент

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом (далі – зазначаються підстави віднесення запозичень до правомірних). Робота приймається до захисту.	+
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:

Співпадіння складають 16,6%

Запозичення виявлені в роботі є законними і не є плігіатом, оскільки:

1) найбільші співпадіння з джерелами:

1. https://ela.kpi.ua/bitstream/123456789/26930/1/Markov_magistr.pdf 8,5%
2. https://cad.kpi.ua/attachments/093_2017dm_Kutsachenko.pdf 7,39%
3. https://cad.kpi.ua/attachments/093_2016d_Avramenko.pdf 5,71%
4. https://uk.wikipedia.org/wiki/Unreal_Engine 5,4%

Розміщені в розділах, які не описують безпосередньо авторське дослідження, і не стосуються результатів роботи

2) усі запозичення фрагментарні, або мають належним чином оформленні посилання;

3) усі інші джерела запозичення мають відсоток співпадінь менший 2%.

11.12.2020
Дата

Підпис Підченко С.К.

Підпис Медзатий Д.М.