Державний вищий навчальний заклад
“Прикарпатський національний університет імені Василя Стефаника”
Кафедра інформаційних технологій

УДК 004

ДИПЛОМНИЙ ПРОЕКТ

Тема Розробка гри на "Unreal Engine"

Спеціальність 121 “Інженерія програмного забезпечення”
код і назва спеціальності

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДП.ІПЗ-30.1-ПЗ

(позначення)

Рецензент
д.т.н., проф.Кузь М.В.
(посада) (підпис) (дата) (розшифровка підпису)

Студент
ІПЗ-41 Баліцький Олег Ігорович
(шифр групи) (підпис) (дата) (розшифровка підпису)

Нормоконтролер
д.т.н., проф.Кузь М.В.
(посада) (підпис) (дата) (розшифровка підпису)

Керівник дипломного проекту
к.т.н., доц. Козленко М.І.
(посада) (підпис) (дата) (розшифровка підпису)

Допускається до захисту

Завідувач кафедри

к.т.н., доц. Козленко М.І.
(посада) (підпис) (дата) (розшифровка підпису)

(рік)

Державний вищий навчальний заклад
«Прикарпатський національний університет імені Василя Стефаника»
Факультет математики та інформатики Кафедра інформаційних технологій
Спеціальність Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Завідувач кафедри к.т.н., доц. Козленко М.І.
„25 ” жовтня 2019р.

ЗАВДАННЯ
НА ВИКОНАННЯ ДИПЛОМНОГО ПРОЕКТУ

Студенту Баліцькому Олегу Ігоровичу
(прізвище, ім’я, по батькові студента)
1. Тема проекту Розробка гри на "Unreal Engine"

Затверджена розпорядженням по факультету математики та інформатики
„25 ” жовтня 2019р. № 7
2. Термін здачі студентом закінченого проекту 2020-05-2020
3. Вихідні дані до дипломного проекту Стандарт кафедри Інформаційних Технологій ПНУ “Вимоги до змісту та оформлення”, Unreal Engine EULA, Unreal Engine API, ISO 8601

4. Зміст пояснювальної записки (перелік питань, що їх належить опрацювати)
“Ігровий рушій Unreal Engine і його застосування, постановка задачі” , “Гейм дизайн проекту, вибір візуального стилю”, “Розробка ігрового білда”, “Методи розповсюдження, економічне обгрунтування”

5. Перелік графічного матеріалу (з точним забезпеченням обов’язкових креслень): “Мета роботи”, “Outline, зображення до застосування оператора собеля”, “Зображення після застосування опеартоа собеля”, “Виділення шумів від оператора собеля”, “демонстрація маски глибини”, “Застосування оператора собеля до маски глибини”, “Результат роботи Outline”, “Cel-shading, оригінальне зображення та альбедо”, “Ділення оригінального зображення на альбедо”, “Постеризація”, “Фінальне зображення з Cel-shading”, “Головне меню гри, Налаштування”, “Розширене налаштування”, “Демострація зброї”, “Алгоритм не покадрової анімації повернення зброї”, “Інтерфейс та інтерактивні обєкти Неігрові персонажі”, “Навігаційний меш”, “Один з елементів карти”, “Згенерована карта” ,”Скріншот гри”

6. Дата видачі завдання 2019-10-25

Керівник _____ Козленко
(підпис) (розшифровка підпису)

Завдання прийняв до виконання _____ Баліцький
(підпис) (розшифровка підпису)

КАЛЕНДАРНИЙ ПЛАН

Номер і назва етапів дипломного проекту	Термін виконання етапів проекту	Примітка
Перший попередній захист	2020-01-31	
Другий попередній захист	2020-05-8	
Здача диплому	2020-05-21	

Студент _____ Баліцький
(підпис) (розшифровка підпису)

Керівник проекту _____ Козленко
(підпис) (розшифровка підпису)

РЕФЕРАТ

Пояснювальна записка: 89 сторінок (без додатків), 41 рисунків, 0 таблиць, 31 джерел, 4 додатоків на 46 сторінках.

Ключові слова: Ігровий рушій, гра

Об`єктом дослідження є Unreal Engine

Мета роботи: Розробка гри, аналіз рушія і оцінка Blueprints

Стислий опис тексту пояснювальної записки:

В даній роботі розглянуто особливості ігрового рушія “Unreal”. У цій роботі розглянуто основні тенденції ігрової індустрії та як розроблюються сучасні ігри. В тексті обговорюються етапи розробки відео ігор. Також безпосередньо описується процес розробки проекту дипломної роботи. В роботі присутні замітки що до альтернативної графічної мови програмування всередині рушія.

Буде розглянуто процес розробки проектів на рушії, буде показано етапи роботи шейдерів і методи їх розробки і алгоритми що стоять за ними. Буде відтворено роботу інтелекту неігрових персонажів. Буде розроблена основна ігрова логіка і пояснено ті чи інші рішення які впливають на ігровий баланс.

Буде розглянуто ігрову індустрію як одну із галузей розваг і її переваги та недоліки серед інших таких галузей і місце програмних інженерів в них.

ABSTRACT

Explanatory note: 89 pages (without appendix), 41 figures, 0 tables, 31 references, 4 appendix on 46 pages.

Key words: Game engine, game

Object of study: “Unreal engine”

Brief description of the text of the explanatory note:

The following work contains descriptions of the main features of “Unreal” game engine as well as the way that main trends in the gaming industry are been implemented in the development process. The work discusses the pipeline of video game development and the process of diploma work project development. There are also notes on the alternative graphical programming language inside the engine, that contains the main features and issues of it. There are also notes on the alternative graphical programming language inside the engine.

The process of project development on the engine is been explained, the stages of shader operation and methods of their development and algorithms behind them are been be shown. During project development, the non-played character intelligence production is discussed. With that been said, the following work implies gaming as one of the areas of entertainment industry, the main advantages and disadvantages among other such industries and the place of software engineers in them are been researched.

ЗМІСТ

ВСТУП

У цій роботі буде розглянуто основні тенденції ігрової індустрії та як розроблюються сучасні ігри. Проект цієї роботи буде базуватись на рушії “Unreal”, під час роботи я буду набиратись досвіду в роботі з ним і я розширю свої навички програмного інженера. Я збираюсь розробляти різні алгоритми на візуальній мові програмування “Blueprint”. На отриманих знання буде базуватись оцінка яку я дам цій мові в порівнянні з більш класичними мовам програмування. Сам рушій підтримує “C++”, але я вирішив відмовитись від нього в угоду навчальних та експериментальних цілей.

На даний період, ігрова індустрія зростає, з кожним роком появляються нові рішення проблем, нові ігри, рушії і так далі. Маючи в безплатному доступі потужні рушію, люди можуть самостійно працювати над проектами різної складності і підвищувати свої навички. Такі проекти згодом можуть принести своїм створювачам користь у вигляді фінансового потоку з продаж і розповсюдження інформації про себе.

Ігрова індустрія зародилась ще 70-х роках і на початках від неї не було багато вигоди розробникам. З роками кількість розроблюваних продуктів зростала, почали долучатись видавці та спонсори. Протягом кінця 70-х та початку 80-х років відбувався період який люди зараз називають золотим віком аркадних ігор. Згодом індустрія почала все більше і більше розвиватись і наближатись до потужності таких гігантів недійних розваг як кіноіндустрія. Зараз, в період коли телевізори відмирають, а на зміну кінотеатрам приходять стрімингові сервіси, на кшталт “Netflix”, ігри є дуже прибутковою і розвиненою частиною культурної спадщини людства. Протягом останніх кількох років і до тепер, індустрія переживає певний занепад, оскільки планка якості ігор понизилась, що можна особливо замітити на проектах AAA класу: зараз складно довіряти видавцям і великим компаніям, які роблять хаотичні вибори в розробці відеоігор, базуючись чисто на аналізі маркетингового відділу. Також

зараз існує тенденція до випуску неготових продуктів, які в деяких випадках полірують і роблять повноцінний продукт або залишають, в наслідку чого попадають в забуття. Попри це, ігрова індустрія все ще зростає з кожним роком, і можливо вона в якийсь момент перевершить голлівудську кіноіндустрію.

В Укрііні ігрова індустрія розвинена не так сильно, проте у нас все-таки є як мінімум одна компанія, яка наблизилась до ААА рівня – “4A-Games”, яка при таких еронічних обставин називається АААА, що є неіснуючою одиницею вимірну, на кшталт рангу “S” в літеральній системі оцінювання. З під руки цієї компанії вийшла серія ігор, яка розвивається в пост ядерній Москві: “Metro 2033”, яка є ігровою адаптацією однойменного твору Дмитра Глуховскього. Крім цієї серії ігор, можна ще виділили більш популярну серію ігор “Stalker”, від менш відомої нині компанії “GSC Game world” (назва якої є ініціалами її засновника, Григоровича Сергія Костянтиновича). Ця серія ігор є культовою в СНГ та за її межами, її дії відбуваються в альтернативній реальності, і якій на Чорнобильській Атомній Електростанції відбувся повтотрий вибух, який викликав появу паранормальних на науково-фантастичних речей. Існують інші вітчизняні розробки по типу “Анабіоз: сон розуму”, “Вівісекція” або “ Collapse ”, проте вони не отримали такого розголосу, я прелічені раніше проекти.

1 ІГРОВИЙ РУШІЙ UNREAL ENGINE І ЙОГО ЗАСТОСУВАННЯ, ПОСТАНОВКА ЗАДАЧІ

Основною задачею цієї роботи, є дослідження рушія “Unreal”, його сфери використання і його вплив на вітчизняний ринок відеоігор. Також до постановки задачі входить розробка проекту на даному рушії, використовуючи сторонні елементи візуального оформлення та самостійно розробленою програмною реалізацією. Ще потрібно розглянути особливості візуальної мови програмування і порівняти її з традиційними. Була розроблена специфікація вимог, якої бажано дотримуватись при розробці проекту(див. додаток А).

1.1 Основні ігрові тенденції

Якщо порівняти новітні ігри, з іграми які вийшли 10-тих, 90-тих або в нульових роках, можна виділити багато інтересних речей, які стосуються їх візуалізації, геймплею (того як вони граються) або інших деталей. Наприклад деякі старі ігри до сих пір виглядають достатньо нормально. Як от “ Warcraft 3”, в якому “можна перерахувати полігони моделей на пальцях лівої ноги”, але завдяки художнім рішенням вона виглядає колоритно, автентично і загалом приємно. Також деякі старі ігри мають хорошу саме ігрову складову (так, більшість людей так думає із-за відчуття ностальгії, але це дійсно так), наприклад “Counter – Strike 1.6” яка хоч вкрай уже всіма заїжджена, але її геймплей живе до сих пір в сучасній “Counter – Strike: Global Offensive”, в яку кожен день грає від 300 до 700 тисяч гравців одночасно. Деякі ігри майже не старіють, як от перші 2 частини “Fallout”, візуальну застарілість яких можна успішно списати на стилізацію (хоч це і не правда і це дійсно дуже візуально застаріла гра). Ігри, які побудовані навколо сильно продуманого сюжету, як в випадку “Star Wars: Knights of the Old Republic 2”, взагалі не старіють. Можна

згадати вітчизняну серію ігор “Stalker”, в яка уже давно по-ігровому застаріла, але вона все-ще жива і в неї грають і тепер, в основному не із-за відчуття ностальгії, а із-за користувацьких розширень, які кардинально міняють гру в візуальному або геймплейному плані. Деякі старі ігри просто не мають своїх аналогів або їх мало і тому вони залишаться актуальними, як ігри жанру “immersive sim” і гра “Deus Ex”. А деякі ігри просто стають класикою, як серії “Half-life” або “Quake”.

Думаю, що розробці своєї гри варто взяти в увагу різні хороші ігри і чинники, яки привели до їх успіху. Проте за останнє десятиліття змінилася не тільки методологія розробки ігор та їх візуальна частина, також сильно змінились потреби покупців: деякі відео-ігрові жанри стали нішевими, наприклад стратегії в реальному часі (скорочено в народі ще називаються RTS від real time strategy) не э такими популярними як в 0-х та 10-х роках цього століття. Так, зараз (на момент написання 29.01.2020) “Warcraft 3” знаходься в тренді “Twitch” (9 місце і 34 тисяч глядачів), але це сталось тільки завдяки виходу ремастера “Reforged”, який нікому не сподобався (якщо судити по тому, що він отримав 2.4 бала з 10 на “Metacritc”) і який передали на оутсорс в Малазію і в якому тільки замінили 3Д моделі з текстури, хоча було заявлено набагато більше змін. Якщо не брати уваги це виключення, то як правило окрім якості, на прибутковість гри ще впливає ще і жанр з сетінгом гри. Наприклад років зо 5-7 назад дуже популярними були ігри по типу “Left 4 Dead” з сетінгом “зомбі” і в, приблизно той самий, період дуже популярною була гра (яка також знаходиться на території сетінгу “зомбі”) “Dayz Z” жанру “survival”. Авжеж, такі ігри можуть і зараз отримати якийсь певний успіх, але по суті їх “золотий вік” вже закінчився. Все-таки в ці 2 грають і нині, якщо відкинути жанр “зомбі” від “Left 4 Dead” то ми отримаємо класику жанру “со-ор-шутера”, а в випадку з “Dayz Z” ми отримаємо надзвичайно гнучку платформу для модифікацій.

Велика половина розробників ігор AAA-розряду сильно регулюється “ринковими потребами” і вкізвками від інвесторів: іменно за останні кілька років це стало сильно замітно з метаморфозою методів монетизації, побудовою гри і вибором жанрів. Значна кількість сучасних ігор-це лайв-сервіси (life-services) які живуть завдяки постійному добавлянню контента та реіграбельності за рахунок мультиплеєрної складової. Такі ігри як правило живуть завдяки продажу предметів(косметичних або іноді корисних для покупців) всередині гри, продажу платного ігрового контенту і в деяких випадках навіть “бустерів” (прискорювачі, які тимчасово підвищують шанс випадіння предметів, кількість отримуваної ігрової валюти або/і балів досвіду). Вперше, цей спосіб підтримки почали використовувати в іграх жанру MMO-РПГ (багатокористувацькі рольові ігри) по типу “World of Warcraft”, проте зараз його можна зустріти в любих жанрах, але найчастіше саме в тих жанрах, які добре “продаються”, наприклад в іграх типу “looter-shooter” або “МОВА”. Також на кінець можна сказати, що великі компанії, хоч і беруть до уваги західний ринок, але вони дуже часто напір на Китайский ринок, наприклад “World of Warcraft” на релізі не могла там продаватись із-за питань з цензурою (в Китаї заборонено зображувати духів та скелетів в іграх), але після того як розробили за цензуровану версію, Китай став приносити прибуток більший, за всі інші країни разом. Також можна згадати, що вийти на Китайський відео-ігровий ринок не так просто, оскільки для цього потрібно мати компанію-проксі, яка буде від вашого імені вести там спарви. Деякі жанри просто популярні, наприклад “Battle royale”, який хоч багатьом надоїв, але він залишається одим із самих популярних на західному ринку, самою популярною такою грою є “Fortnite” (розроблена компанією “Epic Games” і побудована на рушії “Unreal”) яка досягала 10.8 мільйонів одночасних гравців. Також дуже популярними є ігри жанру “МОВА”, одна з ігор цього типу є “League of Legends”, яка знаходиться на 1 місці стрімингової платформи “Twitch” і яку

дивиться близько 130 глядачвів одночасно і яка досягла 8 мільйонів одночасних гравців в 2019 році.

Я, як розробник ігор, не зобов'язаний опиратись на методологію ААА-копаній повінстю, оскільки моєю сферою розробки буде інді-індустрія. До цієї сфери належать малі девелопери, які працюють без інвесторів (хоча вони можуть мати краудфундингову підтримку на "Kickstarter", "Pateron" або інших сервісах). Ігри таких розробників не мають обмежень в принципі, вони можуть робити ігри любих жанрів та сетінгів і можуть собі дозволити (хоча це опціонально) опиратись на більш вузькі кола гравців.

Загалом, в моєму випадку потрібно зробити якусь візуально просту, але дуже стилізовану гру, яка має бути реіграбельною завдяки своєму геймплею. Я буду починати з жанру "шутера від першого лиця", але я не буду обмежувати себе тільки цим жанром, в випадку якщо я захочу поміняти жанр на етапі розробки гри або геймдизайну. Було-би добре реалізувати "Rougle" механіку рівнів, коли після смерті гра починається заново, в іншому згенерованому сценарію (цей жанр досить популярний в інді-індустрії).

1.2 Особливості "Unreal"

Теперішнє покоління "Unreal Engine"[22] використовує мову програмування "C++" для кодування алгоритмів (і систему "Blueprint" як допоміжну) і сильно опирається на об'єктно-орієнтоване програмування. Blueprint – це система візуального кодування в "Unreal", яка є повною системою сценаріїв ігрових процесів, заснована на ідеї використання простого візуального інтерфейсу для створення логіки і візуалізації гри через "Unreal Editor". Хоч ця мова програмування не така багатостороння і оптимізована як "C++", вона все одно є достатньо гнучкою та потужною, також вона надає можливість дизайнерам використовувати практично повний набір рішень та

інструментів які, за звичай, доступних лише програмістам “традиційного” програмування. Також, рішення, характерні для “Blueprint”, можна в реалізації “C ++”, що дозволяє програмістам створювати базові системи, які можуть бути розширені дизайнерами. Ця мова працює за допомогою з’єднання нодів (node - вузол) для досягнення тих, чи інших результатів, наприклад: створення об'єктів, виконання окремих заготовлених функцій та виклик загальних ігрових подій, які є специфічними для кожного екземпляра “Blueprint” з метою реалізації поведінки та інших функціональних можливостей. Також можна замітити схожість цієї мови з аналогічної в програмі “Matlab” під назвою “simulink”, так у них по суті різні призначення але їх принцип роботи достатньо схожий, що видно зразу.

Рушій “Unreal Engine” має основні базові класи, на яких він базується і які в ньому знаходяться ще з самих ранніх версій. Самим основним, з точки зору логіки гри, є клас “Actor” (актор) – самий базовий клас (по суті, це своєрідний аналог класу “Object” для ігрового серидовища), який можна розмістити на рівні, він підтримує переміщення, масштабування та повертання по трьом осям простору[2]. Створювання екземплярів оголошених акторів називається спавном (spawn). Функція спавну потребує тільки назву актора, всі інші параметри (такі як початкове розміщення, або його власник) є не обов’язковими, і виконується вона через функцію “SpawnActor()” мови “C++” або через відповідну функцію в “Blueprint”, яку показано на наступному скріншоті[22] (див. рис. 1.1).

Рисунок 1.1 – демонстрація SpawnActor в середовищі “Blueprint”

По суті, актори – це вмістилища, в яких зберігаються спеціальні типи об'єктів, які звуться компонентами. Компоненти можуть використовуватися для різних цілей, на кшталт керування переміщення акторів, їхня візуалізація та взаємодія з іншими акторами. Не менш важливою їх функцією Акторів є реплікація властивостей інших акторів, для забезпечення синхронізації між мережевими користувачами. При створенні акторів, вони безпосередньо не містять в собі інформацію, натомість до них назначають компоненти, в яких все зберігається. Наприклад, інформація про розміщення, масштаб та повороти акторів в трьохвимірному просторі зберігається в кореневому компоненті, яки ще називають root, всі інші компоненти цього актора унаслідують розміщення рута. Компонентів багато, але з них можна виділити деякі ключові, наприклад:

“USceneComponent” – компонент в якому міститься інформація про розміщення обєкта (якщо його не приєднати, то актор не зможе виконувати будь які маніпуляції з своїм розміщенням), саме цей компонент, як правило, виступає в ролі кореневого.

UPrimitiveComponent – це компонент зберігає інформацію про візуальну репрезентацію, також він тримає в собі параметри фізичних взаємодій (реакція на зіткнення, вага, тощо)

“UActorComponent” – базовий компонент, який не може мати прояв в трьохвимірному просторі, натомість їх використовують для концептуальних прорахунків, як наприклад керування штучним інтелектом. Саме цей компонент відповідає за роботу обновлюючого лічильника: після певного часового проміжку відбувається перехід на новій тік (Tick) і всі данні прораховуються знову, опираючись на зміни, які відбулись за цей часовий проміжок.

1.3 Ітерації рушія “Unreal” та набутий ним набір інструментів

“Unreal engine” був вперше пред’явлений публіці в 1998 році, на якій була побудована одноіменна відеогра “Unreal”. Робота над “Unreal” почалась в 1995 році. Перше покоління рушія було опубліковано в 1998 році (хоча доступ до технологій отримали деякі сторонні компанії, на кшталт MicroProse Legend Entertainment). Після цієї публікації, компанія перейменувалась в “Epic Games”.

Розробником цього рушія (як і саме оригінальної відео гри “Unreal) є компанія Epic Games, яка в свою чергу, була заснована ще на почаку 90-х років. Засновником компанії є Тімом Суїні (який до сих пір є головним виконавчий директором цієї компанії), який закінчив Університет Меріленда на спеціальності “машинобудувальна інженерія “ та заснував “Potomac Computer Systems”. В жовтні 1991, компанія року випустила відео гру “ZZT” для платформ “DOS” та “MS-DOS”, яка продалась тиражом в 5 тисяч копій. Згодом компанія була перейменована в “Epic MegaGames”. Також компанія, до совго перейменування випустила такі відео ігри як “Jill of the Jungle”, “One Must Fall” та “Jazz Jackrabbit”[6].

1.3.1 Перше покоління “Unreal”

Перше покоління було спочатку реалізовано для ПК та макінтоша, хоча згодом його було реалізовано на GameCube, Xbox і PlayStation 2[6]. Ця ітерація рушія підтримувала кольорові джерела освітлення, примітивні, текстурне згладжування та систему фізичних зіткнень. В цьому поколінні також були добре реалізовані halo ефекти (по суті це трохи інакша інтерпретація lens flare ефекту) навколо джерел світла, які повністю зникали при втрачанні зорового контаку з джерелом світла. Приклад цього ефекту можна бачити на даному скрінншоті (рисунок було взято з відеогри “Unreal Gold”[23]) (див. рис. 1.2).

Рисунок 1.2 – демонстрація halo ефекту

Також у вільному доступі знаходився редактор рівнів для гри “Unreal” під назвою “UnrealEd”, що є значним плюсом.

Проте в рушії також були деякі нові, раніше ніким не реалізовані, особливості:

Технологія скайбоксів – це метод візуальної реалізація неба (або точніше горизонту), яка базується на проекції неба на внутрішнії поверхні

своєрідної коробки. По суті це досягалось за допомогою створення зменшеної копії навколишнього середовища, яке потім “транслювалось” на цю коробку. Це рішення дозволяло створювати досить (на той час) фото-реалістичні і динамічні зображення. Також за допомогою технології, яку рушій використовував одним із перших, мультитекстурування (технологія накладання кількох шарів текстур на поверхність трьохвимірних і двовимірних об’єктів за один такт проходу, хоча можна накладати за більше ніж 1 так, але це сильно зменшує швидкодію) за допомогою якого, можна було імітувати візуальне підвищення деталізації ландшафту трьохвимірних поверхонь без Bump mapping-гу, за умови якщо джерела світла були розставленні і настроєні правильно.

Технологія warp – це створення своєрідних “порталів”, через які могли переходити деякі об’єкти. Це досягалось за допомогою заміни однєї поверхні, проекцією іншої паралельної поверхні[6].

Також одним з нововведень було створення дзеркальних поверхонь.

Загалом на період 1999 року існувало 16 грових проектів, таких як The Wheel of Time, Deus Ex

Перед виходом наступного покоління (а саме “Unreal engine” 2) було створено проміжну версію рушія, яка виправляла деякі проблеми попередньої версії рушія. Поліпшеннями цієї ітерації було додавання лицевої анімації, більш продвинута система партиклів, розширення максимальної роздільної здатності поверхонь до 1024 пікселів та розширенням інструментарію “UnrealEd”. Конкретно на цій версії рушія базується перша частина багатокористувацької гри ”Unreal Tournament”. За допомогою особливостей реалізацій мережевого коду, в одній сесії гри могли грати користувачі як і з ПК так і з Макінтоша одночасно. Загалом, якщо вірити Вікіпедії, на цій версії рушія вийшло понад 20 ігор

1.3.2 Друге покоління “Unreal”

“Unreal engine 2” – друге покоління рушія вийшло в 2002 році разом з поліпшеною версією редактора рівнів та з грою “ America's Army ”[6]. Згодом вийшла гра “Unreal Tournament 2003”, а сам рушій перейшло на 32-ти бітну графіку і повністю поміняв ядро графіки, також інструментарій розробників сильно розширився. По суті деталізація картинки збільшилася в 100 розів, в порівнянні з першим поколінням. На наступному малюнку можна побачити різницю в деталізації (ліворуч UE1[23, 25], праворуч UE2[24]) (див. рис. 1.3):

Рисунок 1.3 – демонстрація різниці деталізації між 1 та 2 поколінням

Варто зазначити, що із-за переходу на 32-х бітну графіку, технологія warp стала складно-реалізованою і сильно ресурсозатратною. Також в цій ітерації рушія появились різні нововведення, частина з яких стала доступної із-за використання фізичного рушія Karma (продук діяльності американського підрозділу компанії MathEngine). Серед них:

Фізика ragdoll (буквальних переклад цього терміну звучить як ганчірна лялька, але всі використовують неперекладений варіант) з розширеними фізичними властивості 3-х вимірних об’єктів (по суті це досить масштабне розширення для системи фізичних зіткнень) – технологія за допомогою якої

будується процедурна анімація складних тіл, у яких може бути скелет з суглобами шарнірного або кульового зєднання. При генерації поведінки тіла, на анімацію впливає початковий імпульс, умовна гравітація, вага тіла (як і вага сегментів скелету), навколишнє 3-х вимірне середовище (з яким відбуваються зіткнення) і різні навколишні явища, на кшталт вибухів, вітру або водяної поверхні. Як було сказано раніше, це не проста фізика ragdoll, а розширена і тому в рушії скелети можуть бути обмеженими петельними (шарнірними) зєднаннями. В відеоіграх ця технологія, частіше за все, випростовується після вбивства суперника, поведінка переможеного тіла математично прораховується на стороні клієнта. Результати ранніх версій цієї технології були вражаючими на свій час, але фізика тіла тоді була не достатньо правильно побудована, вона дійсно в якомусь сенсі була схожа ганчірну ляльку, але в реальності властивості людського тіла були більш схожими на “мішок з картоплею”, а ніж на ганчірку.

Обробка поведінки і управляння транспортом – на рушії появилась можливість передачі контролю одного об’єкта, іншому. Хоч в оригинальному “Unreal Tournament 2003” ця функція було не до кінця дороблена, але тим не менше, згодом було реалізовано 2 типа транспорту – стаціонарний та рухомий. Рухомий транспорт базувався на технології ragdoll (з використанням потенціалу методів karma) і до них можна було додатково прикріпити нерухомий транспорт в вигляді станкового кулимета. В основному транспорт слідував вказівкам гравця, який “сидів за рулем”, але в деяких випадках транспорт міг працювати автономно[1].

Система листя (foliage) – це принцип, по якому на поверхні об’єкта або землі, знаходяться дрібні деталі або рослини, які зникають коли гравець не спостерігає за ними або на достатній відстані від них

Водяна поверхня – імітація води через анімацію високо полігональної поверхні. За допомогою цього методу також можна створювати динамічні ефекти, наприклад реакція води на зіткнення з фізичним об’єктом.

Також до нововведень можна віднести передачу голосу в межах однієї онлайн-сесії і розпізнавання голосових команд (за допомогою Microsoft Speech API, можна було надавати вказівки комп'ютеру, який грав замість користувача або неігровим персонажам NPC). Сам звук був модифікований за допомогою технологій EAX 3.0 (розробка Creative Labs), завдяки яких, звук ставав більш обємним.

Загалом, якщо вірити Вікіпедії, то на цій версії рушія (і його модифікацій) вийшло понад 90 ігор.

"Unreal engine" 2.5 – проміжна версія рушія яка може використовує 64-х бітні системи і може підтримувати графічні API функції OpenGL 2 і Direct3D 9. Була реалізована підтримка 16-ти бітного тексту в юні коді. Оптимізація була покращена, а максимальна роздільна здатність текстур була збільшена до 4096 квадратних пікселів.

"Unreal Engine 2 Runtime" – некомерційна версія UE2, в комплекті присутній мінімальний набір ассетів (моделей та текстур) для демонстрації можливостей. Для комерційного використування розробник має придбати ліцензію, але без неї він може використовувати його в навчальних або некомерційних цілях[1,6].

"Unreal Engine 2X" – модифікація UE2 для розробки відео ігор для ігрової консолі "Xbox". Крім заточення під консоль, цей рушій був біль оптимізований, в ньому було реалізовано ефект світіння (Bloom), розширення текстур було модифіковано для підвищення реалістичності, для камери будо додано ефект глибини різко зображуваного простору і ефекти статичного та рухомого розмиття, автоматична корекція яскравості в реальному часі. Також було реалізовано підтримку аудіо-чату та сервісу "Xbox-Live".

1.3.3 Третє покоління “Unreal”

“Unreal Engine 3” – наступна ступінь еволюції рушія була представлена в 2004 році [6]. Першою грою для XBox рушія стала “Gears of War” 2006 року, для платформи Windows стала гра “RoboBlitz” також 2006 року випуску. На старті рушій підтримував підтримував “DirectX “9 та 10 версій разом з “OpenGL” 2, 3 версій API та платформи Windows, PlayStation 3 та Xbox, згодом в 2010 році було портовано рушій на платформи iOS та Android (першою грою Android стала Dungeon Defenders, а для iOS – Infinity Blade). Пізніше рушій модифікували для підтримки Adobe Flash Player 11 та консолі “Wii U”. Також “Epic games” об’єдналась з Mozilla” для того, щоб портувати до Web застосунків за допомогою компілятора “Emscripten” та підмови “asm.js”. Попередні версії UE базувались на модульній системі і використовувала об’єкт-орієнтовану парадигму, саме тому більшість ассетів (включно аж до ассетів UE1) попередніх ітерацій все-ще залишались в розпорядженні розробників ігор. Проте замітною для розробникив зміною рушія було значне поліпшення графіки, оброки звуку, роботи з асетами та інструментарій. Значним поліпшенням стала переробка оброки освітлення, в попередніх версіях рушія тіні та інші світлові ефекти прораховували для кожного вертекса (вертекс – найменша одиниця полігональної моделі), в цій же ітерації прораховувався кожен піксель. Також в попередній версії рушія, істументарій рейдерів був більш простим і лінійним, в новій же версії була реалізована система, якою можна програмувати шейдери і використовувати їх на повний потенціал (при цьому старий інструментарій був все ще доступним). Якість зображення можна продемонструвати цим скріншотом (взято з гри “Gears of War”[14])(див. рис. 1.4):

Рисунок 1.4 – наглядне представлення якості картинки UE3(Gears of War)

Фізична модель рушій була замінена з “Karma” на “PhysX”. Пізніше , компанія-розробник “AGEIA” розширив бібліотеки “PhysX”, завдяки чому стало доступним використання технологій симуляції тканин та рідини в реальному часі. Лицеву анімацію реалізували використовуючи анімаційний рушій “FaceFX” компанії “OC3 Entertainment”. Редактор рівнів “UnrealEd” було роширено за допомогою переписування з використовуванням багатоплатформної бібліотеки віджетів “wxWidgets”. Загалом, якщо вірити Вікіпедії, на цій версії рушія вийшло понад 280 ігор.

“Unreal Engine 3.5” – візуальна модифікація, через яку отримав можливість використовувати Ambient occlusion (буквально перекладається як “навколишня оклюзія”, хоча ніхто не використовує не англомовний термін). Ця функція змінює освітленості об’єктів, опираючись на навколишні об’єкти. Простіше кажучи, рушій затемнює ділянки, в які світло потрапляє менше за все. При обробці затінення не беруться до уваги параметри світла, ось приклад зображення без освітлення з вимкненим і увімкненим Ambient occlusion (на рушії Source[19]) (див. рис. 1.5):

Рисунок 1.5 – наглядне представлення Ambient occlusion

Також значним нововведенням цього обновлення є реалізація Unreal Lightmass, яка базується на Global illumination (Глобальне освітлення), яка дозволяє обробляти відбиті промені світла. Ця реалізація освітлення є дуже ресурсозатратною, але вона дозволяє реалізувати набагато більш реалістичнішу картинку, ніж у Direct illumination (прямого світла). Серед інших змін цієї версії – підвищена швидкість компіляції коду мов програмування C++, покращення симуляції руйнувань навколишнього середовища та м’якої матерії, оптимізація взаємодії з багатопроцесорними системами та оптимізація водяних поверхонь.

“Unreal Development Kit” – в 2009 році “Epic games” випустили безплатний набір із засобів розробки (SDK) для UE3, любий бажаючий міг завантажити рушій і почати розробляти свої ігри (при дотримуванні умов використання). До цього, доступ до рушія мали лише особи які оплатили ліцензію (при цьому інструментарій розробки модифікацій для ігор був безплатний)[6].

Також, варто зазначити, що деякі компанії купляли ліцензію на рушій і робили свою, модифіковану версію.

1.3.4 Теперішнє покоління “Unreal”

“Unreal Engine 4” – сама актуальна версія рушія компанії “Epic games” (на період написання 01.01.2020 сама свіжа версія – 4.23.1), SDK знаходиться в вільному доступі для всіх безплатно. Найраніше згадування датується 18 серпня 2005 в заяві віце-президента М. Рейна в якій говорилось що розробка 4-того покоління почалась ще в 2003 році, проте в 2006 ця заява була спростована виконавчим директором компанії Тімом Суіні, за його словами, 3-тя покоління є достатньо інноваційним і буде залишатись актуальним ще приблизно 4 роки[6]. Офіційно, розробка над 4-тим рушієм розпочалась в 2008 році і перша його публічна збірка стала всім доступною в 2014 році. На старті “Epic games” мали нову модель розповсюдження “UE4” – для доступу потрібно було, всього лише, оформити підписку вартістю 19 доларів США в місяць (при цьому, навчальні заклади і їх студенти отримували його безвинагородно), і як результат, ви тоді отримували необмежений доступ до всього інструментарію, сирцевого коду через гілку на “GitHub” і широкий набір проектів, які служать прикладами конкретних рішень і ігор, або демонстрацією тих, чи інших його можливостей. Наступного року, розробники вирішили скасувати щомісячну підписку і зробили рушій (разом з усіма його наступними оновленнями) безплатним (при цьому розробники любих проектів мають платити 5% від прибутку). За заявою виконавчого директора компанії, кількість користувачів рушія збільшився в 10 разів після відміни підписки. Також в цьому самому році, компанія представила “Marketplace” через який, користувачі могла продавати і купляти різні ресурси для своїх проектів, там можна знайти 3Д моделі, готові анімації, скрипти, звукові набори та інші речі. Більшість з цих ресурсів є платною, проте “Epic games” завантажила деякі свою ресурси туди з можливістю скачувати їх безплатно (деякі користувачі іноді завантажуються свою товари з безплатним доступом). Кожного місяця компанія робить 100% знижку на 5 предметів з “Marketplace”.

Якщо звернути увагу на технічні, візуальні та “quality of life” зміни, то в цьому поколінні є надзвичайно багато різних деталей, які варто згадати:

Система Bluepirnt-ів – хоч і в SDK залишилась можливість працювати з C++, зявилась можливість програмувати рушій без коду за допомогою візуального. Так, ця система не така гнучка і оптимізована як традиційне програмування, але її наявність це є однозначний плюс, оскільки вона дозволяє створювати ігрову логіку людям, які не вміють кодувати.

Стара система пакетів більше не використовується і на томість кожен елемент (будь то матеріал, звук чи 3д модель) зберігається в окремому файлі. Це дозволяє редагувати ці файли на льоту і перекидувати їх між комп’ютерами без плутанини з пакетами. Також ці самі файли тепер набагато простіше імпортувати, а кількість підтримуваних файлів сильно зросла

Повна переробка роботи з анімацією – до кожного анімаційного файла присвоюється свій скелет, що позволяє швидко портувати анімацію з одного скелета на інший. Процес роботи, який базувався на інтерфейсі “AnimTrees” був повністю замінений, завдяки переходу на “AnimBlueprint”. Це сильно спрощує процеси створення складних анімаційних послідовностей і дозволяє створювати “стейт машини”.

Рушій тепер більше полягається на C++. Потрібність в використанні “Kismet” та “UnrealScript” повністю відпадає, і хочу них була дуже розвинута система роботи з “стейт машинами”, за допомогою C++ ми також можемо їх створити, і навіть більше: ми можемо створити кращі рішення задач за допомогою, наприклад, дерева поведінки штучного інтелекту. Також тепер можна компілювати код підчас роботи рушія, що значно пришвидшило зручність.

І на сам кінець, рушій тепер дуже легко можна розширювати за допомогою плагінів і скриптів. В попередньому поколінні це булл дуже великою проблемою, розробники дуже часто жалілись на відсутність розширюваності “Unreal Engine 3”. Тепер ми можемо завантажувати різні

застосунки з “Marketplace”, що може сильно покращити час роботи та редактор. Разом з SDK іде набір демонстраційних та експериментальних функцій[6].

Ще можна згадати можливість підтримки одночасної гри 100 клієнтів. Ця можливість була реалізована в 2017 році завдяки своєрідній “гонці Батл Роялів”, коли різні ігрові компанії замітили попит режиму “Battle Ropyale” в грі “Players Unknown Battleground”. Суть режиму полягає в одночасній конкуренції в грі за виживання 100 гравців, на карті, радіус якої постійно звужувався. Коли “Epic games” взяли участь у цій “гонці” і зробили обновлення “Fortnite”, яке добавляло в гру цей режим і робило її умовно – безплатною.

Загалом, за допомогою широкого інструментарію на цій платформі можна реалізувати такі речі як симуляція 3-х вимірних об’єктіві, рендер надреалістичного або вкрай стилізованого зображення для фільмів, візуалізація архітектурних рішень і на сам кінець, ігри з майже найсучаснішими інтерактивними та захоплюючими ігровими враженнями та високою якістю оформлення (також рушій використовується в навчальних цілях).

Першою відеогрою на цьому рушії стала “Daylight” компанії “ Zombie Studios ”, яка була випущена в 2014 році.

В цілому, якщо вірити Вікіпедії, на цій версії рушія вийшло понад 280 (як і в попереднього покоління) ігор, але ця цифра є дуже неточною, із-за великої кількості інді-ігор (ігор які були розроблені без видавця) і вільності доступу до рушія. Скоріше за все, цифра кількості проектів на UE4 є чотирьох-значною.

Що - до правил використання і розповсюдження, розробники повинні платити 5% за продукти, дохід яких складає більше 3,000$ на квартал. Продукти “Marketplace” заборонено розповсюджувати в не компільованому вигляді. Обмежень на зміст і підтекст ігор немає, хоча були заяви розробників що – до adult контенту. По їх словам, це питання колись буде вирішено.

1.4 Порівняння Unreal з іншими ігровими рушіями

Є багато рушіїв крім “Unreal”, думаю основними його конкурентами на даний момент є: “CryEngine”, “Unity”. Так є багато і інших рушіїв, по типу “GoDot” який використовує мову “Python” для написання ігрової логіки, але він не достатньо популярний і про нього занадто мало інформації, щоб включити його в цей список. Якщо порівнювати “Unreal” з “Unity”, можна виділити кілька переваг та недоліків:

Якщо гра має працювати на мобільних пристроях, по типу “Android”, то “Unity” показує себе з ліпшої сторони, є дуже багато ігор в “Play Market” на цьому рушії. “Unreal” також може підтримувати такі платформи, але це він робить з великою натяжкою і нестабільністю.

З точки зору візуального оформлення, “Unreal” виходить вперед із-за своєї гнучкої системи рейдерів. За допомогою доступних ресурсів цього рушія можна згенерувати зображення вищого ґатунку. “Unity” також має систему рейдерів, але вона і близько не така потужна, хоча при достатньому умінні теж можна добитись результатів.

Набір сторонніх ресурсів. У цих обох рушіїв є “Asset Store”[6], але аналог “Unity” має набагато більший асортимент, одних тільки безплатних асстеів тільки 5500 штук, у “Unreal” їх приблизно 360. Ситуація з платними ресурсами не сильно краще: 10440 у “Unreal” і 58005 у “Unity”. З точки зору мови програмування, скоріше за все краще “Unreal”, але це залежить від вподобань – він використовує “C++” і “Blueprint”. “Unity” в сою чергу, використовує “C#” або “Javascript”.

Що до ціни, “Unreal” безплатний до тих пір, поки ви не отримаєте 3000$ прибутку за минулий квартал[22]. В випадку коли ви перейдете цей поріг, вам прийдеться платити 5% від доходу. Ситуація з “Unity” трохи складніше, вона різні версії зрізною ціною (по 40$ та 150$ на місяць), але є також і безплатний варіант. Серед обмежень безплатної версії: ВІДСУТНІСТЬ ЧОРНОГО

ІНТЕРФЕЙСУ, низька настроюваність інтерфейсу та відсутність інтерфейсу налагодження для оптимізації. Вам не прядеться платити нічого окрім підписки(або ліцензії) до тих пір поки ви не отримаєте 100000$ за рік

Одним із самих головних аргументів стає стабільність: хоч інструментарій іноді “вилітає”, самі ігри виходять набагато оптимізованішими і стабільнішими ніж у “Unity”.

Тепер що до “CryEngine”, якщо його порівнювати “Unreal”, то можна виділити насутпні переваги та недолікіки:

Перше що кидається в очі – це обмеження що-до вмісту. В цьому рушії забороняється мати контент такої направленості: образливий, расові чи етнічно образливий, вульгарний, явно сексуально виражений, наклепницький, порушуючючий права на приватне життя чи публічність, або, на думку розумної особи, неприйнятний. В принципі в тому концепті гри, що я задумав нічого такого нема, але факт залишається фактом.

З точки візуалізації “CryEngine” показує себе краще, хоча обидва рушія є новітніми, в конкретних випадках ситуація може бути достатньо високою.

Аналог “Asset Store” для “CryEngine”, також є, але він дуже скудний і налічує тільки 1336 елементів, серед яких, 119 є безплатними.

“CryEngine” Використовує мови програмування “C++”, “Lua” та “C#”. Також на базу рушія є багато готових рішень, що дозволяє не винаходити квадратні колеса для велосипеда заново. В плані ціни, “CryEngine” вам прийдеться платити 5% як і у “Unreal”.

2 ГЕЙМ ДИЗАЙН ПРОЕКТУ, ВИБІР ВІЗУАЛЬНОГО СТИЛЮ

2.1 Основна ігрова ідея і вибір ресурсів

Виходячи з отриманих результатів дослідження і з коментарів керівництва, я трохи змінюю рішення що до основної ідеї. По першій задумці цей проект мав стати іграбельним прототипом rougle-like динамічного шутера на стадії Альфа або Бета тестування. Натомість я вирішив сильно спростити концепцію (а також понизити планку якості і деталізації), щоб можна було загально довести проект до стадії релізу.

При виборі основних ігрових ідей я опирався на прості класичні ігри, по типу шутера від першого лиця “Quake” або наприклад ізометричної “Hotline Miami”, в яких ігрова механіка і є простою, але це іде їй на руку завдяки швидкому геймплею. Взагалі такий “простий” тип ігорна кшталт “Quake” вже поступово втрачає актуальність, і зараз все частіше появляються шутери з біль складними механіками. Наглядним підтвердженням цього служать ігри “Doom 2016” і “Doom Eternal” (яка на момент написання не вийшла, але не неї вже є відгуки від журналістів), перша частина “перезапуску” з точки зору ігрової механіки не є складною, і все сходиться до простого “цілься і стріляй”, а вот сіквел уже сильно опирається на продуманість процесу і комбінування ігрових механік (також варто замітити, що в грі фукція “присідання” була замінена швидким ривком, оскільки нею ніхто не користувався, що ще сильніше підтверджує мої слова про біль складні ігрові механіки). Хоч я і буду намагатись слідувати за нинішніми тенденціями, проте завершеність гри є в більшому пріоритеті, ніж різновидність ігрових механік. Побудову гри я буду починати з більш простих механік, і якщо буде достатньо часу, то я дороблю більше механік, на готовність ігрового продукту не ніяк не вплине. При нинішній ситуації з виробниками ігор, межа між готовим і неготовим продуктом є стертою – в основному це стало сильно замітним з “бумом” гіор з

системою розповсюдження “Early access”, яка пропропонує купити неготову гру і використовувати її по прямому призначенню. Дуже часто почали появлятись неготові гри, які навіть не доходили до релізу і це вплинуло і на повноцінні закінчені ігри – при нинішніх реаліях, розробники можуть легко випустити гру в якій частина контену не є повною. Технічно гра незакінчена, але це ніяк не впливає на факт того, що гра продається і в неї грають. Ця гра насправді є закінченою, а недороблений контент можна випустити пізніше в вигляді безплатного або платного доповнення.

Від сюжета, як такого, по факту приходиться відмовлятись, ніхто не забороняє просто придумати якийсь дуже простий сюжет, написати його в тектовому редакторі і закинути його файл в папку з горю, як колись робили в старих іграх). На перший погляд, це може здатись чимось негативним, але виходячи з цитати хресного батька шутерів Джона Кармака (розробника таких культових ігор як “Doom”, “Quake” і “Wolfenstein”) , “Сюжет в відеоіграх є невід'ємною частиною, проте у нього така сама роль як у проно фільмах. Він є необхідний але він ні на що не впливає”. Так, всіма було уже давно, що сюжет в відеоіграх може грати більшу роль ніж оформлення, геймплей або якійсь інші аспекти (це підкріплюється існуванням такого жанру як “візуальна новелла”, або кінематографічними іграми компаній “Tlltale Games” або “Quantic Dream” на кшталт “Heavy rain”), проте в деякий нових іграх сюжет дійсно не грає важливу роль. Наприклад, в “Doom 2016” сюжет є, але ради нього ніхто не грає, або наприклад в деяких іграх “From Software”, в яких хоч і є дуже пророблений “задній сюжетний фон” (який прийнято називати лором), але підчас гри сам сюжет сильно не розвивається.

Щодо назви гри, людина може проявити інтерес до проекту виходячи з її обгортки і тому оскільки це абстрактна гра і в ній сюжету як такого нема, я вирішив жартівливо назвати її “PNU’s nameless videogame, 2 layers of irony”. 2 шари іронії являють собою пост-іронію, яка базується на концепції при якій

складно побачити різницю між серйозністю і іронією (по суті, жарт заключається в тому, що це не жарт).

При виборі ігрових ресурсів (моделй, анімацій, тощо) я вирішив не виходити за межі безплатного контенту “Marketplace” у “Epic games store”.

2.2 Переміщення, функціонал та інтерфейс персонажа

В грі має бути інтерфейс головного меню з базовими настройками, екран загрузки рівня, екран програшу та екран перемоги (який буде появлятись перед зміною рівня). На інтерфейсі гравця має відображатись його життя та патрони, також там буде відображатись його кількість балів.

Хоч раніше і було сказано, що гра буде від першого лиця (як “ Half Life” або “Deus Ex”), проте я вважаю доцільним буде розібрати цей вибір більш детальніше, оскільки його можливо варто змінити. Якщо розібрати цей термін з точки зору літератури, то прописані від першої особи персонажі як правило визначаються через вживання займенників “ми” або “я”. Також існують ігри від третьої особи (до таких належать ігри “Gears of War”, “Mass Effect” тощо), в яких камера знаходиться заді персонажа і ми ним керуємо, можна сказати це не зовсім правильно, оскільки з точки зору літератури “третя особа” визначаються через займенники “вони”, “вона” або “він”. Виходячи з цього, можна сказати що для такої ситуації більше підходять ігри жанру “стратегія”, проте цей термін вже встиг відокремитись від загальних правил літератури і отримав нове значення. Ігор від другої особи (в літературі, займенники “ти” і “ви”), як таких не існує, хоча можна виділити одну гру, яка підходить під задні критерії, а саме гра “ Driver: San Francisco” жанру “гонки”. В основному гра проходиться від першої особи або третьої особи, а конкретніше, ми керуємо або своїм автомобілем, або чужим (по сюжет, антагоніст може переселятись в чужі тіла). Але в одному сегменті гри, ми керуємо одночасно свій автомобіль і автомобіль

який нас переслідує, управління рулем здійснюється одночасно в 2 машинах. Реалізація дуже оригінальна, але складно тому її я використовувати не буду. Також існує ізометричний під-жанр, в якому камера знаходиться зверху або зверху і трохи під нахилом (як наприклад в грі “ Hotline Miami”). Такі ігри є простими в розробці за ігри від першої особи, проте Unreal Engine має багато готових рішень саме для першої особи, тому цей варіант не є рентабельним. У нашому розпорядженні залишається перша і третя особа. Різниця між ними не є дуже значною, проте я залишу свій вибір за першим. Цей вибір я роблю посилаючись на 2 значних аргументи, а саме –

1. Сильно динамічних шутерів від третьої особи дуже мало, і конкретні екземпляри такої комбінації на думку зразу не спадають.
2. Для третьої особи потрібно анімувати модель протагоніста, що значно підвищить ресурсозатратність на розробку гри.

Наступним питанням в розробці головного героя персонажа стає те, яка саме він буде переміщатись. Тут значну роботу за нас вже виконав рушій, оскільки в ньому вже є повністю робоча система переміщення, яка нам підходить. Нам потрібно виставити правильні параметри в налаштуваннях, щоб персонах переміщався як в швидких, динамічних шутерах.

Крім цього, варто задуматись над реалізацією деяких додаткових засобів переміщування: можна реалізувати “подвійних стрибок” (гравець може повторно стрибнути в повітрі), якщо ми зробимо ігрове поле в якому це буде корисно, або ми можемо зробити “ривок” (персонаж швидко переміститься в сторону). І ще я вирішив добавити в інтерфейс користувача його “комбо” або“серію убивст” (користувач буде бачити скільки ворогів він подолав за короткий проміжок часу, така деталь часто добавляється до слешерів як “Metal Gear Rising ”, але іноді добавляється і до шутерів, як наприклад “Hotline Miami”).Це буде реалізовано через виконання запрограмованих функцій при натисканні кнопок.

Далі нам потрібно продумати арсенал гравця. Я вважаю що “гвинтівки”, “пістолета”, “ножа” і “дробовика” буде достаньо. Кожну зброю треба буде перезаряджувати, якщо його магазини стане порожнім, в деяких випадках можна зробити альтернативний режим стрільби (або навіть декілька). Нам вато обрати те, як будуть прораховуватись снаряди озброєння, на вибір є дві основні концепції – “hitscan” і “projectile”.

“hitscan” (з англійської буквально перекладається “як сканування ударів”) концепція, при якій снарядів як таких не існує. Гра прораховує промінь від дула зброї, і при вистрілі пуля моментально попадає в першу перешкоду на його шляху. Цей метод використовується в класичних іграх, по типу “Counter Strike”. Оскільки пулі фізично не існує, то розробникам довелось винайти декілька рішень, наприклад, пулі летять моментально, але їх дія після зіткнення відбувається з маленькою затримкою.

“Projectile” (з англійської перекладається як “снаряд”) концепція, при якій снаряди маю свою балістику і він, на відміну від “hitscan”, не є абстрактним. Таку модель використовують ігри різних жанрів, починаючи від реалістичних, як “Arma” і закінчуючи аркадною “Doom”. На “Unreal engine” для цього вже існує своя система, в якій можна обирати різні параметри, такі як настильність, швидкість або реакція на попадання. Для нас основним параметром тут виступає саме швидкість, як швидко пуля долає відстань.

Тут я вибираю другий варіант, оскільки його буде простіше реалізувати і на ньому можна буде зробити декілька альтернативних режимів стрільби.

Концепція кожної зброї буде не сильно відрізнятись від класичних, проте я добавлю декілька альтернативних режимів стрільби:

“Пістолет” буде стріляти повільно, але точно в основному режимі стрільби і на оборот – швидко, але з сильними відхиленнями і віддачою.

“Гвинтівка” основному режимі буде стріляти з середньою швидкістю, а в альтернативному пулі будуть відстрибувати від поверхонь, але магазин буде вичерпуватись в 2 рази скоріше.

“Дробовик” буде вистрілювати кілька пуль за раз і низькою швидкістю стрільби.

Враховуючи всі обставини, я вирішив що на даному етапі прокет буде знаходитись під захистом доктрини ”сумлінного використання” у зв’язку з навчальним характером проекту (у зв’язку з тим, що я не буду комерціалізувати проект по своєму власному бажанню, бо не змушений це робити виходячи з програмних вимог до диплому ПНУ). Саме по цій причині я збираюсь використовувати звуки з гри “Half Life 2”. Вибір пав саме на цю гру опираючись чисто на мою суб’єктивно оцінку.

При виборі візуальних моделей, я обуду опиратись на безплатні ресурси з інтернету, режим доступу на використані моделі будуть знаходитись в списку посилань дипломного проекту.

Для реалізації зброї буде використовуватись об’єкт-орієнтоване програмування, де буде абстрактний суперклас зброї, види зброї будуть її нащадками. Теоретично, різновиди зброї не обов’язково мають бути окремими нащадками суперкласу (а просто її одиницями), але із-за особливостей інтерфейсу рушія, простіше зробити їх повноцінними нащадками. Далі знаходиться UML схема, яка наглядно показує таку реалізацію (див. рис. 2.1)::

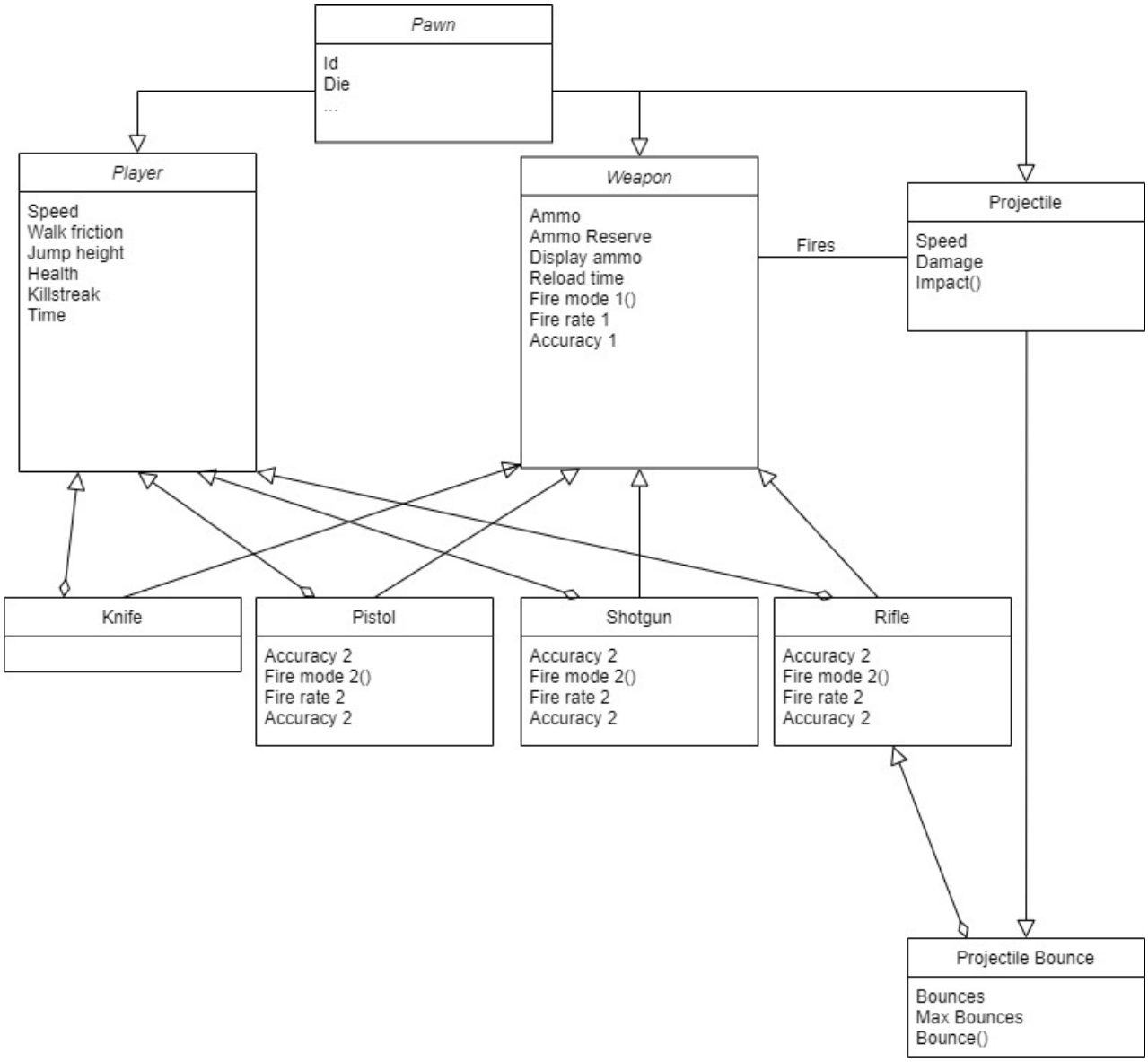

Рисунок 2.1 – UML таблиця відношень зброї

2.3 Навколишнє середовище

Дисципліна, яка відповідає за розробку рівнів називається “левел дизайном” або “мапингом” На даний момент, створення дизайну карт для кожного індивідуального рівня в новітніх гріових сутдіях як правило в більшості випадків починається саме з концепт-артів, ескізів, візуалізації та прототипізації фізичних моделей. Після завершення попередьої стадії, ці

концепції утворюють документацію яка містить інформацію про моделювання середовища та розміщення конкретних ігорвих акторів гри. Іноді цей етап проходить в редакторах рівнів, а не в редакторі рушія, також варто зазначити, що цей етап настає як правило після створення базового набору функцоналу гри (переміщення персонажу) щоб було легше розставляти геометрію на рівнях, з якою будуть взаємодіяти гравці. Тим не менше, мені доведеться протй повз цього етапу, оскільки цей підхід має на увазі сприйняття дизайну карт, як окремої галузі, а в цій дипломній роботі я намагаюсь узагальнити весь процес до меж одного проекту.

Раніше було сказано, що гра буде мати часткову генерацію рівнів. Я би хотів розглянути це питання до того як ми візьмемось за візуальне оформлення рівнів. До мене вже вирішували це питання, і найкращим виходом з ситуації буди використання публічних за стосунків і бібліотек. Але для їх правильного застосування, потрібно розібратись як вони працюють: в відео-лекції по процедурні кімнати в “Unrean Engine” була наведено приклад з використанням дерева квадратів, на якій базується “гра життя”. В представленій реалізації було наглядно показано як саме відбувається поділ на кімнати. Раніше згадане дерево квадратів діє за наступним принципом: воно поступово, в випадковому порядку, розрізає область на 4 прямокутних зони, ця дія повторюється до тих пір, поки ми не отримаємо клітини необхідного розміру[12,28]. Очевидно, що клітини можуть ділитись до безкінечності (що також очевидно може викликати багато проблем, включно з через мірним навантаженням системи і навіть критичними помилками). Щоб цього не траплялося, нам в край необхідно їх зупинити в конкретний момент. Першими на думку спадають 2 наступних рішення цієї проблеми – ми можемо це вирішити через вказування мінімального розміру клітин, менше яких нічого генеруватись не буде, або ми можемо зробити максимальну кількість поділів. Також у нас є вибір в самому методі поділу (точніше в їхньому порядку), ймовірно це не буде сильно впливати на швидкодію роботи програми, проте ми маємо вибір серед таких

методів – ми можемо поділяти зразу всі куби, які знаходяться в одній ієрархії або спочатку поділити в глибину 1 куб. На наступному рисунку зображено перший і другий, ліворуч і праворуч відповідно (див. рис. 2.2):

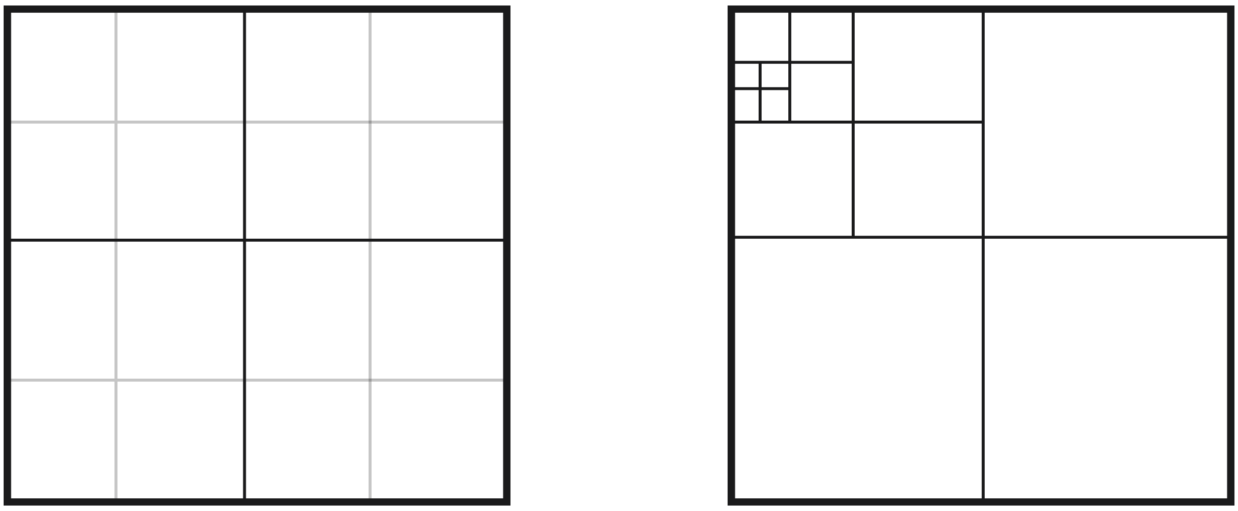

Рисунок 2.2 – Рисунок з двома можливими методами поділу

Після цього, випадковим чином будуть генеруватись кімнати а між ними будуть проведені “коридори”.

Далі зображено приклад, який поділяє область на кімнати, опираючись на тільки описаний метод дерева квадратів. На рисунку зображено синій (самий вищій ієрархії) квадрат, який поділяється на 4 синіх секції, далі кожна з цих секцій ділиться на 4 червоних, далі на 4 фіолетових, 4 білих і 4 помаранчевих прямокутника. Далі в кожному прямокутнику вставляється 1 кімната (позначені зеленим кольором) яка також обирається випадковим чином[12,28] (див. рис. 2.3):.

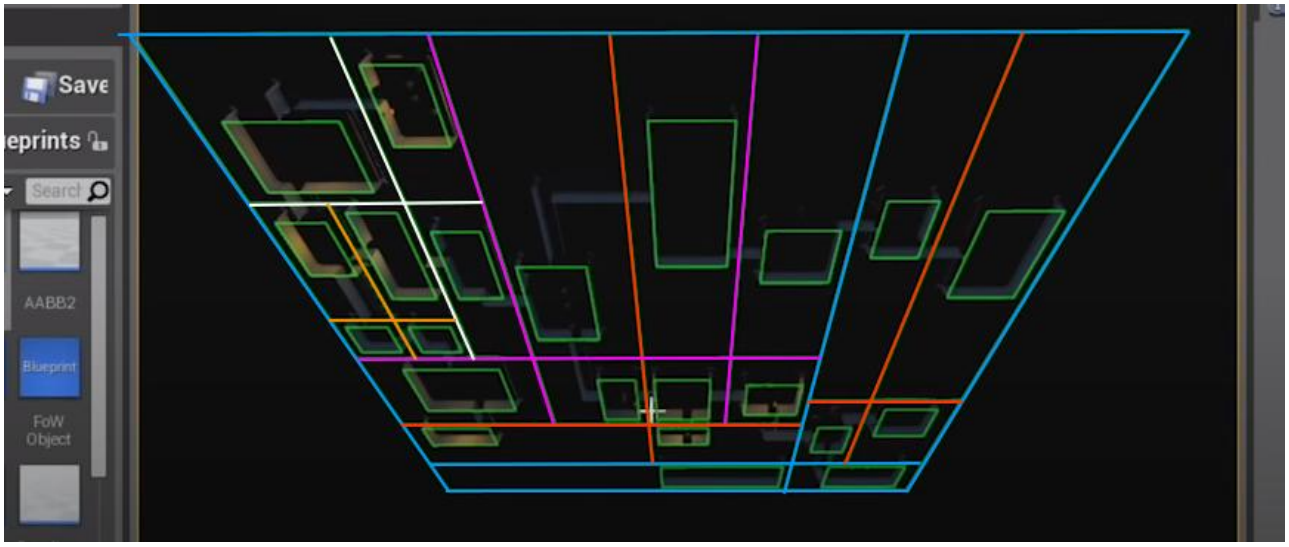

Рисунок 2.3 – Приклад поділу на кімнати за допомогою методу квадратів

В цьому прикладі показано не весь розподіл, на справді він продовжує ділитись надалі на клітини, які будуть використовуватись для коридорів.

Цей алгоритм є лише першою частиною генерації кімнат, тепер ми маємо клас даних в якому є всі необхідна атрибути для побудови карти. Варто зазначити, що класи даних не є доступними в “Blueprints”(навідміну від “C++”), проте цей недолік можна легко обійти, якщо використовувати акторів для цієї цілі[12]. Так актори можуть сильно навантажувати систему, але цього не буде, якщо їх не візуалізовувати. Таким чином, ми можемо швидко генерувати велику кількість акторів і не перейматись за перенавантаження обчислювальних ресурсів, тому що вони є достатньо “дешевими” в цьому плані без візуалізації.

Ще ми маємо прорахувати де саме будуть коридори.

На даному етапі закінчується математичні розрахунки і починається побудова на отриманих даних. Ми маємо оприділитись з тим, які саме кімнати ми будемо вставляти в окремі ділянки. Є знову 2 варіанти розвитку подій – кімната генерується зі стін, які її утворюють або ми вставляємо кімнати, які були зроблені заздалегідь. Перший метод краще вставляє кімнати у вказаних координатах. Другий робить можливість більш деталізовано проробляти

індивідуальні локації (хоча кожну таку локацію доведеться окремо розробляти, що може бути ресурсозатратним з точки зору мапингу). Думаю найкращим рішенням буде використання обох методів одночасно, але опиратись більше на алгоритм першого. В такому випадку, стіни – це своєрідний атом, з якого б'ються локації. У кожного такого атома є свій розмір, який рівний одиниці. Унікальні кімнати також представляють собою такі атоми, але у них, в свою чергу, розмір є більший за одиницю і якщо він не занадто великий, то він може поміститись в ту чи іншу область. Також перший метод можна поділити ще на 2 – можна будувати або тільки сітни, або заповнювати блоками весь простір на карті, окрім кімнат. Різницю можна зрозуміти за допомогою наступного рисунку(див. рис. 2.4):

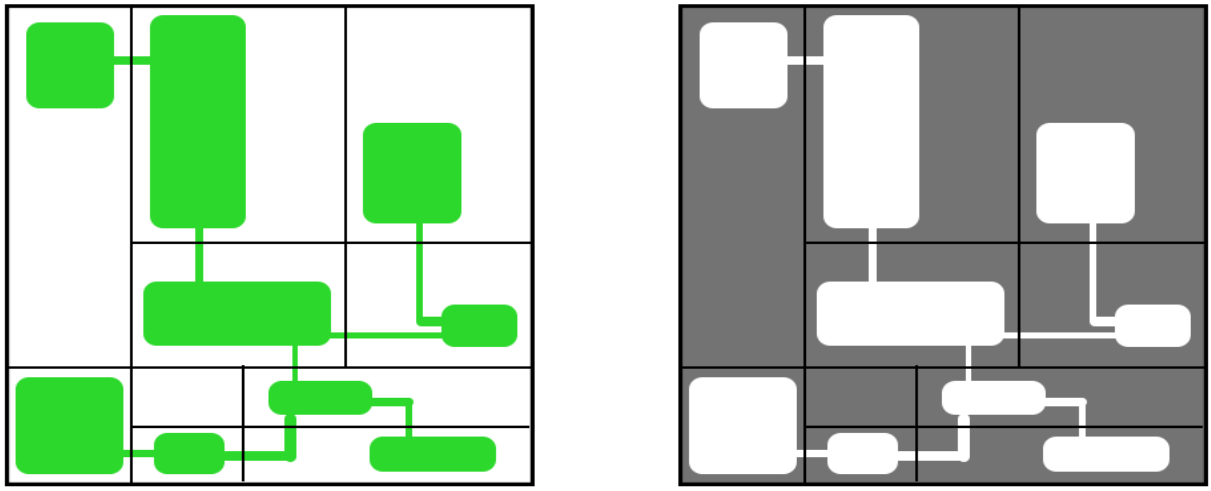

Рисунок 2.4 – Візуалізація

На питання "чому це суттєво?" є наступна, очевидна відповідь, а саме – у цих двох методів є також свої плюси і мінуси: перший може мати більшу варіативність поверхонь за рахунок відсутності необхідності робити блоки, які мусять завжди мати можливість дотикатись до інших блоків з усіх 4-х сторін (або навіть з 6-ти, якщо ми також маємо генерувати унікальні блоки на різній висоті), також цей метот є потенційно менш ресурсозатраним,оскільки нам

треба генерувати і візуалізувати менше блоків. При цьому, метод з заповненням всіх блоків поза меж кімнат має свої плюси: при використанні ізометричної камери ми отримаємо візуально біль привабливий результат, також тільки при використанні цього методу можна реалізувати систему руйнувань: це буде набагато простіше зробити саме в цій реалізації, оскільки при знищені куба, не треба буде заново створювати інші куби, з першого погляду це може здатись ресурсозатратним із-за потенційного навантаження візуалізованими кубами, але для вирішення цієї проблеми, достатньо просто ховати куби, які ми фізично не можемо побачити (також ми можемо зробити блоки не квадратними, квадратними, а шестигранними або будь якої іншої форми, ми ще можемо використовувати прості кубічні поверхні, на осові яких буде будуватись біль складний і реалістичний рельєф). Виходячи з думки, що наш проект не буде ізометричним і що ми не плануємо введення системи руйнувань блоків, я дійшов до винятку, що краще за все буде використовувати тут перший метод.

Після визначення того як ми будемо будувати карту, ми маємо розробити методологію її візуалізації. Якщо просто сторювати статичні примітиви по вимозі алгоритму, ми будемо сильно нагружати обчислювальні ресурси машини, оскільки рушій думає що кожен примітив є абсолютно унікальним з точки зору геометрії, текстури і можливо деяких інших параметрів. У нас є відносно невелика кількість блоків що повторюється, тому ми можемо втратити багато ресурсів буз ніякої вигоди.В цьому випадку нам допоможе використання конструктора одного з двох рішень. Перше рішення – це використання “Instanced Static Mesh”, який позволяє повторно будувати статичні фігури в різних координатах без сильного навантаження на процесорні та відео ресурси. Друге рішення – це збереження наборів геометрії в ”картах”: всі дії в ”Unreal” відбуваються на так званих “картах”, саме на них зберігається ігрове поле, проте ми можемо будувати карту з інших карт без особливої загрузки, що робить цей метод рівносильним до попередньо названого. А якщко ще й взяти до уваги, що другий метод дозволяє зберігати не тільки

статичну геометрію, а і об’єкти з якими можна взаємодіяти, то очевидно що другий метод є більш виграшний. Ще я планую зробити багаторівневість рівнів але при цьому кімнати не зможуть знаходитись одна на одній, виходячи з прироби дерева квадратів[6, 12, 22].

На рахунок візуального оформлення, я не планую використовувати багато сторонніх ресурсів і по суті макро-дизайн кімнат зведеться до простих “сталевих прямокутників”. Тим не менше я планую розбивати рівні на “секції” між якими вороги не зможуть пересуватись, сам же поділ на секції може появитись або не появитись, в залежності від випадковості генерації. З точки геймплею, я скоріше за все, не буду чимось заповнювати самі кімнати і якось розвивати їх дизайн.

Варто зазначити, що всі ресурси візуального оформлення будуть взяті зі базового набору рушія і на них посилання вказано не буде.

2.4 Неігрові персонажі, їх переміщення та їх логіка

Виходячи з доступного інструментарію рушія, я збираюсь придумати систему, яка буде випадковим чином генерувати ворогів в різних місцях. Самі неігрові персонажі будуть реалізовані використовуючи дерево поведінки – інструментарій який вирішує як саме вони будуть реагувати на ті чи інші збурники, і діяти в різних ситуаціях (в якому сенсі це дерево є реалізацією нечіткої логіки, оскільки вони поділяють багато спільних речей). Це дерево дуже схоже на те як було реалізовано “blueprint” – в нашому розпорядженні також є велика кількість вузлів, з’єдання яких утворює логіку штучного інтелекту. В свою чергу, всю інформацію з якими працює дерево поведінки, зберігається на дошці “Blackboard”. Після отримання інформації за допомоги органів чуттів, штучний інтелект перебирає варіанти з дерева і виконує їх в

певному порядку (а саме в напрямку з верху в низ або з ліва на право) беручи до уваги їх з’єднання і ієрархію.

Виходячи з усього вищесказаного, я побудував наступний прототип дерева поведінки (див. рис. 2.5):

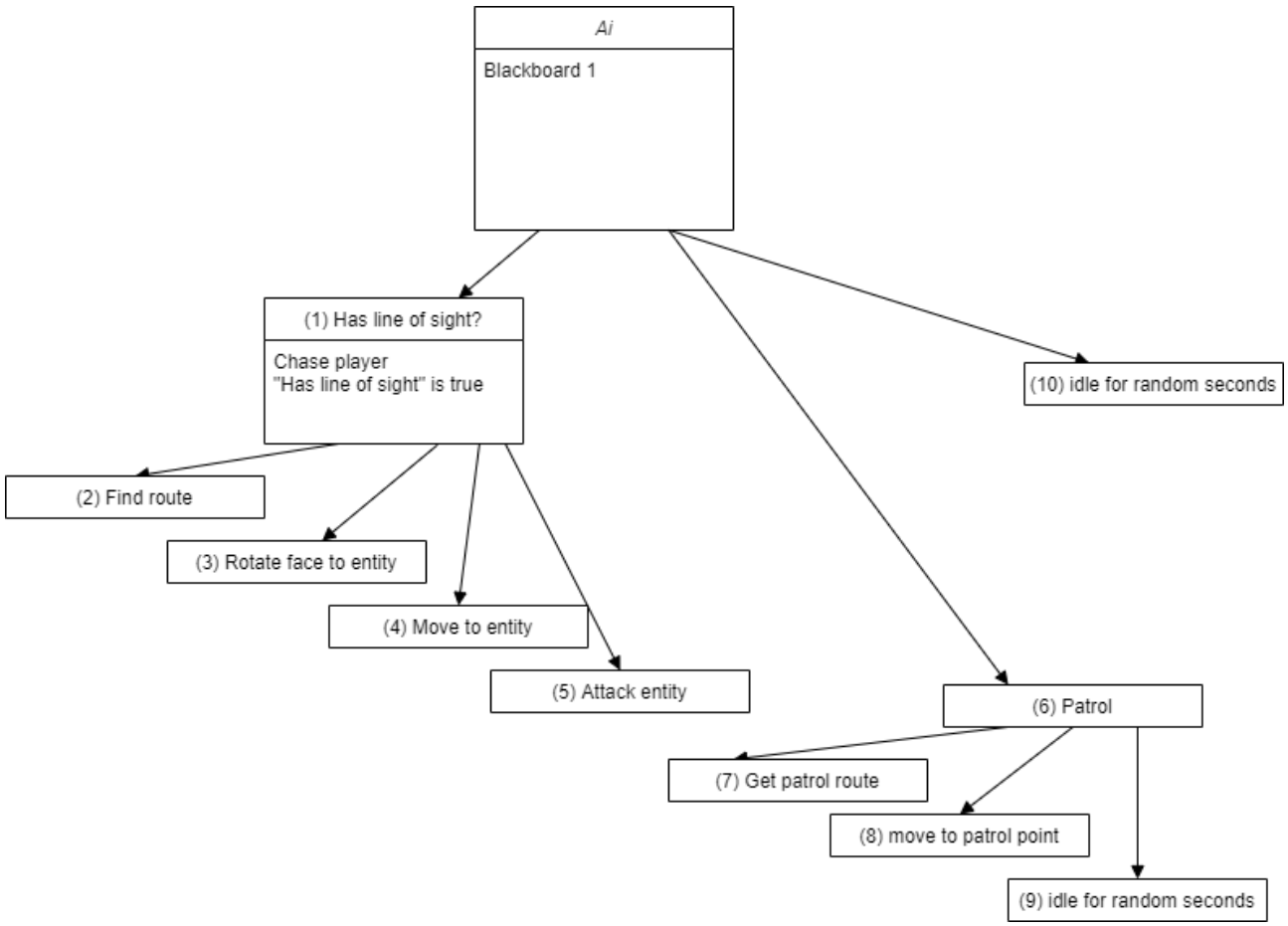

Рисунок 2.5 – Прототип дерева поведінки неігрових персонажів

При виборі візуальних моделей і анімації, я буду опиратись на ресурси з “Unreal marketplace”, режим доступу на використані моделі будуть знаходитись в списку посилань дипломного проекту.

Неігрові персонажі будуть прораховувати шлях руху за допомоги системи навігаційних мешів “Unreal”

2.5 Візуалізація за допомогою Cel-shading і outline

Вибором в стилізації зображення послужили різні сучасні ігрові прикладі, такі як “Borderlands”, “Persona” або незліченна кількість ігор жанру ”Файтинг”, які намагаються повторити візуальний стиль двовимірної японської анімації в трьохвимірному середовищі. На мою думку, відтворити двохвимірний стиль в фільмах і іграх, які використовують полігони для побудов обємних моделей в трьохвимірних декораціях, є дуже складною і потребує велику кількість роботи. Далі по тексту буде описуватись Cel-shading і Outline, які іноді використовуються коли розробники намагаються відтворити двохвимірний стиль проте я вважаю що необхідно пояснити чому цих двох методів не для цієї цілі достатньо і чому слід сприймати в першу чергу як більша узагальнені рішення проблем (і вважати окремим відгалуженням в стилізації), а не тільки рішеннями в проблемах імітації 2D стилю. Ці два методи – є тільки зовнішньою шейдерною оброкою, які працюють з колірними каналами зображення, картами нормалей, та їх глибинними картами.

Карта нормалей – це вектори, які вказують напрямок променів які виходять з певної криволінійної площини, і які також є перпендикулярними до площини в точці з якої вони випромінюються точці. В 3D графіці карти нормалей використовують для двох цілей. Перша ціль – це збільшення деталізації поверхонь об’єктів за рахунок геометричної імітації дрібних подряпин, корозії, випуклостей та впадин з якими активно буде взаємодіяти світло, і в залежності від його напрямку вони можуть динамічно генерувати малі тіні та засвіти. Друга ціль – це обчислення напрямку поверхонь для обчислення світла. Саме за допомогою цього елемента, рушій визначає які участи геометрії треба промалювати в тіні і які мають прийняти світло – якщо нормаль полігона направлена в протилежний бік до джерела світла, то цей полігон є на 100% освітленим (якщо не брати до уваги прорахування тіней) і

якщо вектор нормалі співпадає напрямку від джерела, то він не отримує ніякого освітлення відповідно (якщо не брати до уваги прорахування відбиття світла від інших поверхонь). Всі геометричні моделі в індустрії тривимірної графіки побудовані з полігонів (які як правило мають трикутну форму, хоча рідше зустрічаються і багатокутні варіації), які в свою чергу утворюють меш. Кути цих полігонів називають вертексами (або вершини від англ. слова vertices), і саме ці елемент фігури мають в собі інформацію про нормалі. Це є важливим аспектом в графіці, оскільки вертекси вважаться освітленими, якщо їх нормалі направлені в сторону світла

Далі представлена формула, по якій обчислюється вектор нормалей в точках на площині (хоч він може і не співпадати з тим як обчислюється нормалі на рушії “Unreal”, оскільки ця інформація є важкодоступною) (див. формулу 2.1):

$$n = \frac{\partial r}{\partial s} \cdot \frac{\partial r}{\partial t}, \tag{2.1}$$

де n – це вектор нормалей (X, Y, Z),
r – це координати в криволінійній системі координат,
s, t – дійсні числа напрямку.

Приклад карти нормалей можна побачити в наступному малюнку, де праворуч показано оригінальну текстуру зброї, а ліворуч показано карту нормалей, яка підвищує деталізацію роботи зі світлом без модифікації геометричної формат (приклад взято з набору ресурсів гри “Team fortress 2” на рушії “source”) (див. рис. 2.6):

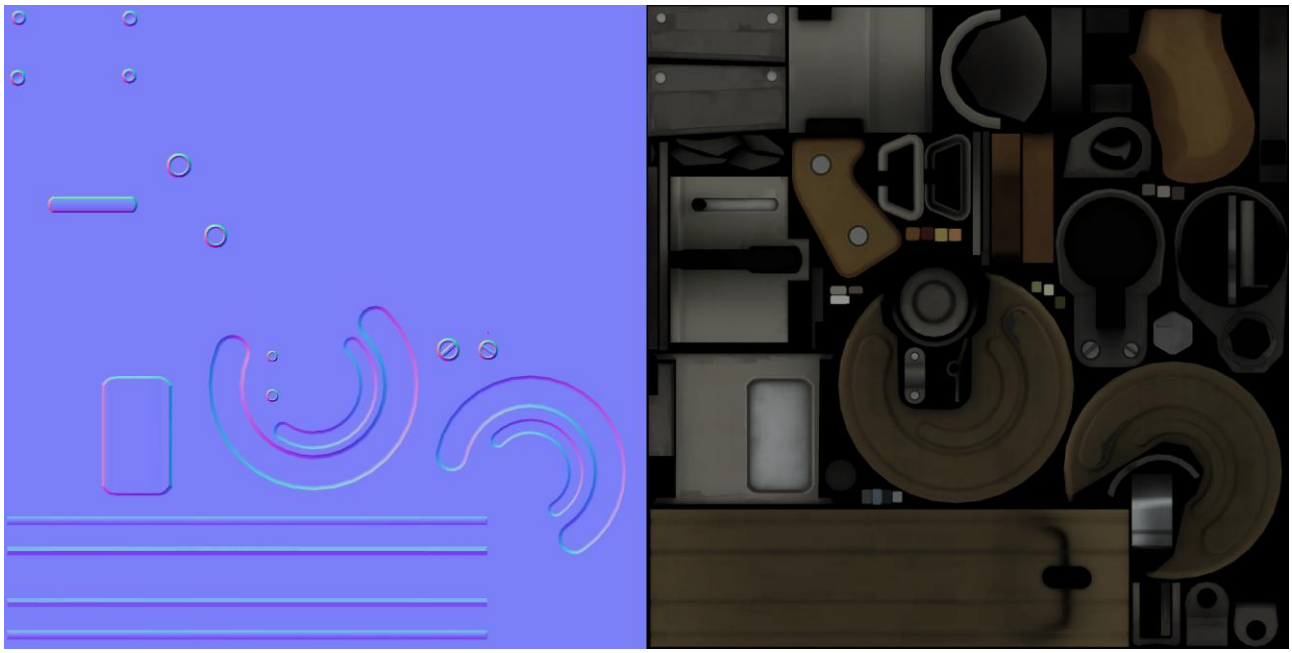

Рисунок 2.6 – Карта нормалей

Також варто зазначити, що кольори на картах вказують напрям, а саме, рожевий вказує підйом з права на ліво, а синій вказує спуск з права на ліво.. Кольори можуть відрізнятись в різних рішеннях рушіїв, проте концепція лишається незмінною. Карта нормалей не вказує глибини поверхонь, а тільки їх кут. Генерація нормалей без знання математично точної поверхні об’єкта є складною, тому отримання нормалей з фотографій є фактично неможливою (теоретично можна їх генерувати за допомогою штучного інтелекту, але це буде складно із-за тіней, відблисків світла та зміни в кольоровому діапазоні які обґрунтовані законами колірної теорії) і найточніша їх генерувати від руки. Проте з іншої сторони, у нас не має виникнути проблем при автоматичній генерації нормалей текстур, оскільки вони є двовимірними по природі, ймовірно саме так і були генеровані нормалі на представленому раніше рисунку (див. рис. 2.6). Саме за допомогою карт нормалей, розробникам “Half life 2 episode 2” вдалось зроби гру, яка використовує текстури двохрічної давнини і при цьому виглядати краще – вони просто добавили карти нормалей зверху на їх старі текстури.

Глибинні карти це інтерпретація картинки, в якій пікселі, що знаходяться ближче до спостерігача є світлішими за ті, що є від нього дальше, як і в випадку з нормалями, генерувати геометрично правдиві їх версії з фотографій просто так не вийде.

Відповідь на питання ”Чому тема стилізації має таке вагоме значення в роботі програмного інженера” є наступна: будь яке стилістичне рішення художника має бід собою фундамент з програмного забезпечення інженера (як банально це би не прозвучало), наприклад розробка драйверів для графічних планшетів: нехай самі програмні інженери ними користуватись можуть рідко, проте самі художники драйвери не зроблять. Проте це не означає що між художником и програмним інженером є нездолання перепона – саме існування такої професії як ”технічний аніматор” (або “rigger”), яке вимагає не тільки знання анатомії і принципів анімації а і розуміння програмування, оскільки йому доведеться робити весь інтерфейс скелетів і працювати з скриптовим середовищем в “Blender” і “Maya” які використовують мову “Python 3”. Можливо я б і вибрав проект, який є в межах компетенції цієї професії (оскільки я є тенденція працювати з нею більше і оскільки у мене є такі уміння, це я можу довести своїм портфоліо аніматора[3]), проте ця думка прийшла до мене занадто пізно і можливо було б складно придумати достатньо масштабний проект.

2.5.1 Cel-shading

Складно сказати якою першою грою з використанням cel-shade була, судячи по даним з Вікіпедії, це “Mega Man Legends” 1998[4]. Проте основним орієнтиром в реалізації я вибрав саме “The Legend of Zelda: The Wind Waker” 2002 року (також в 2013 вийшло перевидання гри з покращеною графікою), хоч вона і по сучасним міркам застаріла, проте вона і зараз виглядає об’єктивно

добре (якщо не звертати на кількість полігонів місцями, хоча навіть це можна сприйняти як естетичну особливість гри). Є багато ігор які намагались використати цел-шейдингову стилізаціюб, у яких були невеликі проблеми із-за виборі в стилі – освітлення могло виглядати не дуже добре з естетичної точки зору. І саме у цієї вийшла найкраща реалізації цієї задумки (яку довели ще до більшого рівня в “Breath of the Wild”). Дуже складно дотримуватися балансу в налаштуванні цього рейдера – він має виглядати так як задумана і певні непродумані моменти можуть призвести до тіней які занадто темні і до засвітів які занадто яскраві. Ймовірно було-б добре розбирати гру, яка була розроблена на “Unreal”, проте ця гра на мою думку підходить ще краще із-за того що всі елементи були пророблені з нуля і вони не такі складні як в сучасних рушіях.

Повний контроль над рушієм розробників цієї гри, позволив їм запрограмовувати особливі параметри для об’єктів, наприклад вся динамічна геометрія може приймати тільки одне джерело світла (і при наближення до наступного ближчого джерела світла, відбувається швидкий але плавний перехід). Платформа “Game cube” для якої гра було розроблена, могла підтримувати до 8-ми джерел світла одночасно без проявів нестабільності, це підтверджує той факт що обмеження активних джерел освітлення є не обмеженням процесуальних здатностей, а художня задумка виробників. Причиною, по якій вони зробили цей вчинок, можна назвати бажання зробити фігури більш виразними – фігура буде кращою для сприйняття оком, якщо вона буде мати одну освітлену та одну затемнену зону, це в свою чергу робить силуети персонажів більш чіткими. Це всього лиш одна з особливостей, які було розроблено для того щоб дизайн був настільки приємний гравцям. Можна розібрати ще багато технік які було використано, проте більшість з них можна віднести до технології cel-shading і далі ми будемо розглядати як вона була використана для візуалізації персонажів.

Процес побудови цел-шейдингу в цій грі можна поділити на 3 етапи –

- Vertex Lightning – одна з перший технологій прорахунку світла в трьохвимірному просторі (ще її називають “Затемнення за Гуро”), яку було пояснено трохи раніше. Ще можна добавити, що ця технологія була дуже розповсюджена в 2000 роках, хоча математичну реалізацію було винайдено і опублікований Анрі Гуро в 1971 році[17].
- “Toon-ification” – так звана туніфікація, яка є ядром цієї концепції, вона проводить пастеризацію вертексного освітлення, що в свою чергу дає нам характерні стилю чіткі краї між тінями та засвітами.
- Color palette adjustment – етап на якому рушій задає тіням та засвітам унікальний відтінок, яскравість та насиченість кольору[17]: замість того щоб показувати тіні такими якими вони є правильними з точки зору фізики трьохвимірного середовища, задається нова колірна палітра за допомогою рейдерів. Це одночасно один із найпростіший і водночас і складніших процесів цел шейдингу, оскільки його відносно просто реалізувати з токи зору програмування, але при цьому дужа складно підібрати правильні кольори без знання колірної теорії.

Кожен з цих елементів сам по собі не надає картинці чогось особливого, проте разом вини утворюють синергію яку називають цел шейдингом. Важливо знати, що на релізі гри, ці всі технологію були відносно інноваційними: на старіших версіях консолей на яких були розроблені попередні частини можна було використовувати тільки вертексне освітлення в примітивні формі. На нових консолях, розробникам довелось винаходити технологію цел шейдингу заново, оскільки в їх розпорядженні була тільки поліпшена версія вертексного освітлення.

В грі можна замітити, що об’єкти, на яких напряму світло не мало-би падати (наприклад вони знаходяться за стіною), але при цьому вони освітлені. Цей феномен має логічно пояснюється тим, що ми перевіряємо тільки кути нормалей відносно джерела світла: утворені пакети даних не містять жодної інформації про перешкоди, які можуть знаходитись на шляху абстрактних

фотонів. Ця особливість не є стилістичним рішенням і вона наявна тільки із-за присутніх обмежень на обчислювальних ресурсах платформи “GameCube”, проте це не означає, що розробники не розглядали вирішення цієї проблеми: ми можемо бачити ефект затемнення екрану і появлення яскравого осередку з ефектом “відблиску об'єктива”, якщо ми будемо дивитися прямо на джерело світла. Цей ефект називається “Eye Adaptation”, або “Auto-Exposure” в “Unreal” і його основною ціллю є імітація пристосування ока до яскравих джерел світла (також його можуть використовувати для затемнення яскравого зображення з метою забезпечення більш комфортного процесу гри і меншим навантаженням на очі гравця). Це і є доведенням того, що розробники знали про присутню непостійність в певних елементах оточення (зокрема відсутність прорахунку тіней).

На передній план потрібно поставити той факт, що ця гра очевидно не намагалась відтворювати реалістичну візуальну складову середовища (навіть якщо взяти до уваги те, що візуально вона намагалась копіювати стиль мультиплікаційних фільмів). Розробники самі вирішували, які елементи мають відповідати реалізму і які ні (це також підкріплюється штучним обмеженням одночасних джерел світла, які освітлюють об’єкти одночасно): цей процес і однією з основних складових процесу розробки художнього стилю.

Після етапу розміщення вертексного світла, в грі починається стадія тіуніфвкації (на цьому етапі, до геометричних моделей ще не було застосовано текстур). В файлах гри присутні особливі текстури градієнтів освітлення, які використовують як пропускну маску освітлення, на наступному рисунку показано зверху градієнт, який відповідає вертексному освітленню, нижче же показано маску туніфікації (див. рис. 2.7):

Рисунок 2.7 – Градієнти вертексного освітлення і cel shading

Шейдер бере інформацію про освітленість вертекса і пропускає отриману інформацію через маску, на виході ми отримуємо наступний результат з цієї гри (для отримання зображення було використано сторонній ресурс який емулював середовище гри) (див. рис. 2.8). Для порівняння, я показую персонажа до і після застосування шейдера (також я показую третій варіант в якому замість стиснення градієнта відбувається його пастеризація, всі скріншоти було всязто з “The Legend of Zelda: The Wind Waker” [17, 20]) :

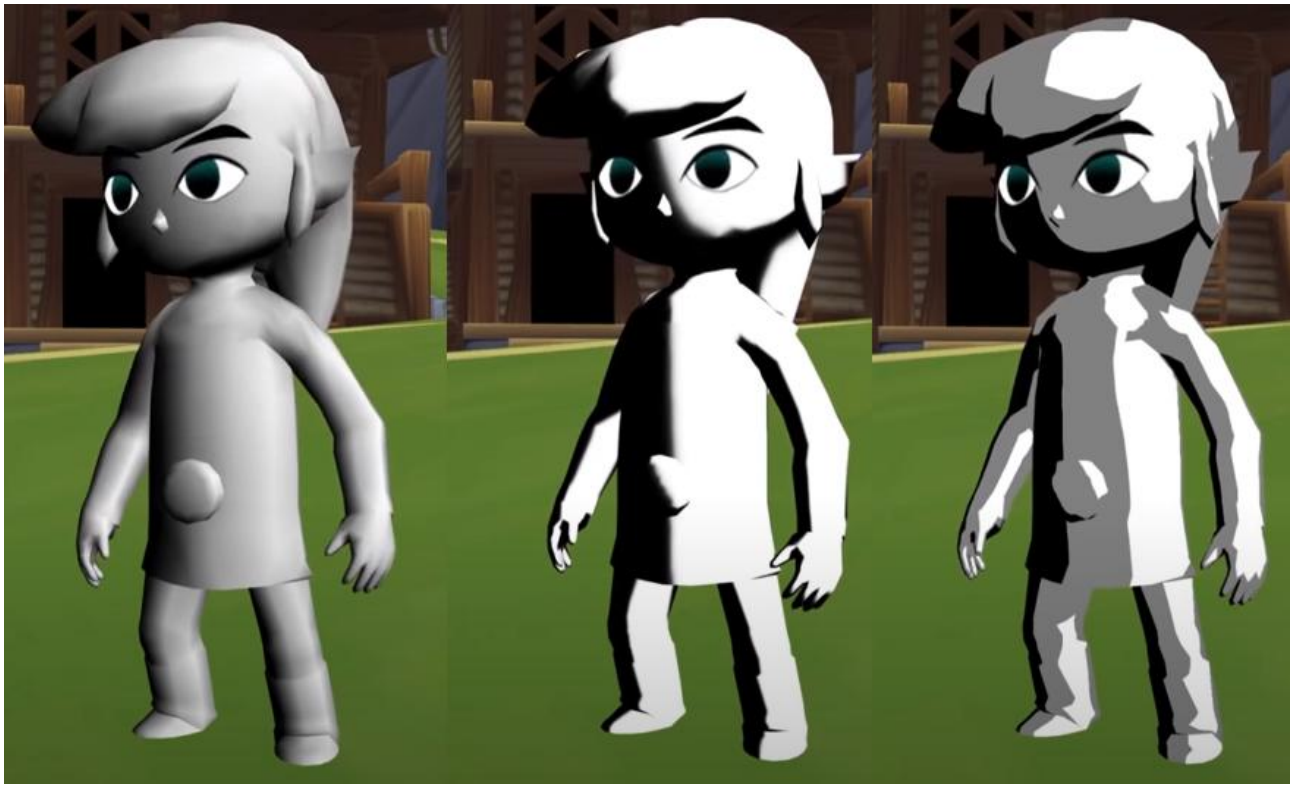

Рисунок 2.8 – Персонаж до і після застосування шейдера

На цьому моменті настає остання частина процесу рендера цейл рейдера, а саме ми обираємо який колір має відображатись на персонажі. Насправді в реальному житті колір поводиться слідуя певним законам, які пояснюються фізичними властивостями і які можна зрозуміти через теорію кольору, в нашому же випадку колір підбирався вручну.

Теоретично розробники могли просто генерувати градієнти, опираючись на те, що загальний його відтінок буде динамічний, а його затемнена частина буде на 30 або 20 відсотків темніше. Ця модель хоч і проста в використанні і розробці, але вона зменшує кількість контролю: всі градієнти слідують одному шаблоні і розробники не можуть поміняти насиченість або яскравість тільки в одній зоні градієнту, якщо вони так хочуть (також вони не можуть змістити перехід в градієнті горизонтально). Натомість команда розробників розробила широкий ряд унікальних градієнтів, які відповідають різним погоднім умовам, часовим циклам і навколишнім середовищам. В сумі можна нарахувати 6 часових циклів дня, 4 погодні умови, 5 об'єктів освітлення; якщо не брати до уваги можливі навколишні середовища, то ми отримуємо близько 120 вручну розроблених таких градієнтів. У ручного і автоматичного методів є свої плюси і мінуси але розробники вибрали саме ручний метод оскільки їм було вигідніше потратити більше часу на розробку градієнтів замість обмеженої кількості автоматично генерованих (не варто забувати що розробка гри відбувалася на початку 2000 років і зараз цей процес міг би бути набагато простіше). Єдиним винятком з цієї системи можна назвати реалізацію "адаптації ока", насправді це насильно закодоване правило, яке затемнює всі градієнти окрім неба і сонця.

2.5.2 Outline

Існує глибока помилка в сприйнятті поняття Outline та Cel shading: деякі люди вважать що цел шейдингм і є обведення контурів героїв, проте як уже

було раніше обговорено, він насправді просто образно кажучи застосовує пастеризацію до освітлення без ніякої обводки, після цього уточнення можа приступити до розгляду прикладів, пояснення що таке Outline і як він саме працює. Outline – це є більш узагальнене поняття: під ним можна розуміти як і монолітне обведення об’єкта по його контуру, так і обведення деталей на картинці безпосередньо. Зараз ми будемо розглядати тільки другий варіант, оскільки він біль складний і саме його ми будемо використовувати. До цю концепцію завжди використовують разом з алгоритмами цел шейдингу, і хоча це твердження можна проігнорувати, без цел шейдингу, обводка буде виглядати дуже неестетично.

Застосовувати алгоритми обведення в трьохвимірних сценах почали використовувати для імітації водночас грубого і естетично приємного “комікcного” візуального стилю і одним із самих характерних прикладів такого підходу в відео-ігровій індустрії можна назвати серію ігор “Borderlands”: сама перша частина цієї трилогії на початковому етапі розробки не передбачала якогось особливої візуалізації, проте ближче кінця розробки вирішили це поміняти. І хоч перша частина зустріла достатню болючу критику відносно наповнення гри, наступні платні доповнення і частини принесли компанії великий успіх. Достатньо важливим аргументом може послужити те що коли ми думаємо про цю франшизу, ми майже першим ділом згадуємо само про візуальний стиль, а потім ми уже починаємо задумуватись про інші особливості, яких досить багато, наприклад: в основі геймплейної задумки лежить використання ідеології “Diablo” стосовно ігрових предметів, ворогів та завдань разом з механіками шутера від першого лиця, що було досить інноваційно в свій час; сетинг гри міг на перший погляд показатись пост апокаліпсисом, проте своєрідний дикій захід в космосі зі своєю абстрактною золотою лихоманкою, і ніякого кінця світу не було (що також було на той момент досить оригінальним); в кожній, від самої першої, грі присутня випадкова генерація предметів з різних деталей (наприклад, який певний

автомат міг складатись зі ствола, прикладу, обойми, прицілу і корпуса), завдяки великій кількості елементів з яких можна було будувати озброєння ми могля отримувати незліченну кількість унікальних пристроїв (а саме сімнадцять мільйонів сімсот п'ятдесят тисяч в першій частині, сто п'ятдесят сім мільйонів, двісті дев'яносто тисяч сімсот шість в другій частині і близько мільярда в третій частині[16]); значна частина завдань в грі хоч і зводилась до банальних “іди туди і вбий тих” або “візьми і принеси”, проте майже всі персонажі з якими ви спілкуєтесь мають дуже колоритні і інтересні особистості і прописані діалоги; також варто не забувати про те що гра є представником нині рідкісного під жанру ігор, а саме вона є кооперативною PVE грою яких зараз мало, а на момент виходу першої частини їх було катастрофічно мало. Таким чином, ми бачимо що гра і має багато інтересних сторін, проте люди згадують про них не в першу чергу: перше що у них появляється перед очима це – унікальний візуальний стиль, який базується на алгоритмі пошуку країв на зображенні.

Задача з пошуку країв не є тривіальною, оскільки ці краї не завжди є очевидними, це і є основною проблемою в цій задачі якщо картинка не є примітивною. В нашому розпорядженні є широкий ряд вирішень цієї проблеми, які як правило опираються на один з двох методів: знаходження максимумів на наявному зображені та відшуковуванні нулів на наявному зображені. Перший метод – це знаходження країв за допомогою визначення контрастності країв, зазвичай че проводитсья з допомгою проразування похідної першого прядку, знайдення локальних екстремумів і отримання градієнтів, передбачення шляху краю далі залежить від вектору градієнта, оскільки вони між собою перпендикулярні. Другий метод базується на обробці даних, виходячи з того як перетинаються осі абсцис в похідних другого степеня, як правило це нулі операторів Лапласа або іноді нулі диференціального нелінійного рівняння. Всі методи як правило використовують фільтр Гаусса для згладжування кроків.

Для розгляду я оберу алгоритми пошуку країв “Canny edge detector”[5]: він проходить в 4 етапи і першим ділом ми повинні зменшити кількість шуму

на зображенні, робиться це за допомогою наступної формули (див. формулу 2.2):

$$F(x, y) = G * I(x, y), \tag{2.2}$$

де I – це вхідне зображення,
G – це фільтр Гаусса

Далі ми повинні прорахувати градієнти: на цьому кроці ми знаходимо в яких саме місцях відбувається найрізкіша зміна в відтінках сірого. Оператор Собеля використовується для отримання градієнту в кожному пікселі згладженого зображення. Оператор Собеля в на осі x є наступним (див. формулу 2.3):

$$D_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}. \tag{2.3}$$

Оператор Собеля в на осі y є наступним (див. формулу 2.3):

$$D_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}. \tag{2.3}$$

Маски Собеля розгортаються разом з зобраденням, що пройшло через фільтр Гаусса. Таким чином, ми можемо отримати чіткість і чутливість градієнтів за допомогою даної формули (див. формулу 2.4):

$$G = \sqrt{{G_x}^2 + {G_y}^2}. \tag{2.4}$$

Де *Gx* та *Gy* обчислюються за допомоги формули 2.5

$$G_s = D_s * F(i,j), \quad (2.5)$$

де *s* – це один з операторів/осей Собеля *x* та *y*.

Напрямок градієнту визначається за наступною формулою (див. формулу 2.6):

$$\theta = \text{artctan}(\frac{G_y}{G_x}). \quad (2.6)$$

Наступним кроком буде видалення всіх даних на зображенні окрім максимальних значень градієнтів. Кожен піксель *P(x,y)* обчислюється наступним чином:

- Заокруглення кутів градієнтів до найближчих 45 градусів, потів порівняйте яскравість градієнтів в різних напрямках: якщо градієнт направлений вгору, то порівнюйте його з тим що іде вниз.
- Якщо яскравіть межі в *P(x,y)* сильніша за різницю між градієнтами, тоді ми зберігаємо його значення (в іншому випадку ми його очищуємо)

На даному етапі ми маємо певний шум в отриманих результатах і тому нам потрібно встановити два пороги, які будуть його відсікати, значення порогів будуть позначатись як $П_{макс}$ і $П_{мін}$. Далі в кожному пікселі *P(x,y)* ми будемо перевіряти наступні умови:

- Якщо $G < П_{мін}$ то ми очищуємо межу.
- Якщо $G > П_{макс}$ то ми зберігаємо межу.
- Якщо $П_{мін} < G < П_{макс}$ і якщо рядом існують піселі (в радіусі 1 піксель), які більше $П_{макс}$, то ми зберігаємо межу.
- В любих інших випадках ми позбуваємося межі.

Отриманий нами результат буде зображувати всі виділені межі на зображені, це саме те зо нам потрібно. Проте цей алгоритм ми будимо

застосовувати не до самого зображення з гра, а безпосередньо до карти глибини, суть якої я пояснював раніше. Також на етапі розробки я можу поміняти алгоритм за яким будуть отримуватись межі на зображенні, якщо я буду вважати що “Canny edge detector” не підходить.

3 РОЗРОБКА ІГРОВОГО БІЛДА

3.1 Реалізація Cell shading і outline за допомоги технології шейдерів

Теоретично, цю частину роботи треба було би робити в кінці проекту, але я вирішив її робити на початку, оскільки з нарощуванням функціоналу рушій може споживати більше ресурів і тому краще зараз розробити ці шейдери, чим раніше.

Шейдер — програма для одного із ступенів графічного конвеєра, що використовується в тривимірній графіці для визначення остаточних параметрів об'єкта чи зображення. Вона може включати в себе довільної складності опис поглинання та розсіювання світла, накладення текстури, віддзеркалення і заломлення, затінення, зміщення поверхні і ефекти пост-обробки[8].

3.1.1 Outline

Першим ділом я створив сцену з кулею і манекеном для тестування. Наступний рисунок був зроблений за допомогою стандартного набору ресурсів рушія і саме до цієї сцени ми будемо застосовувати шейдери[22](див. рис. 3.1):

Рисунок 3.1 – Зображення до застосування рейдерів

Перед тим як перейти до правильного рішення цього питання, я продемонструю чому ми маємо використовувати карту глибин. На наступному рисунку ви можете спостерігати інвертований результат проходу алгоритму Канні[5, 22](див. рис. 3.2):

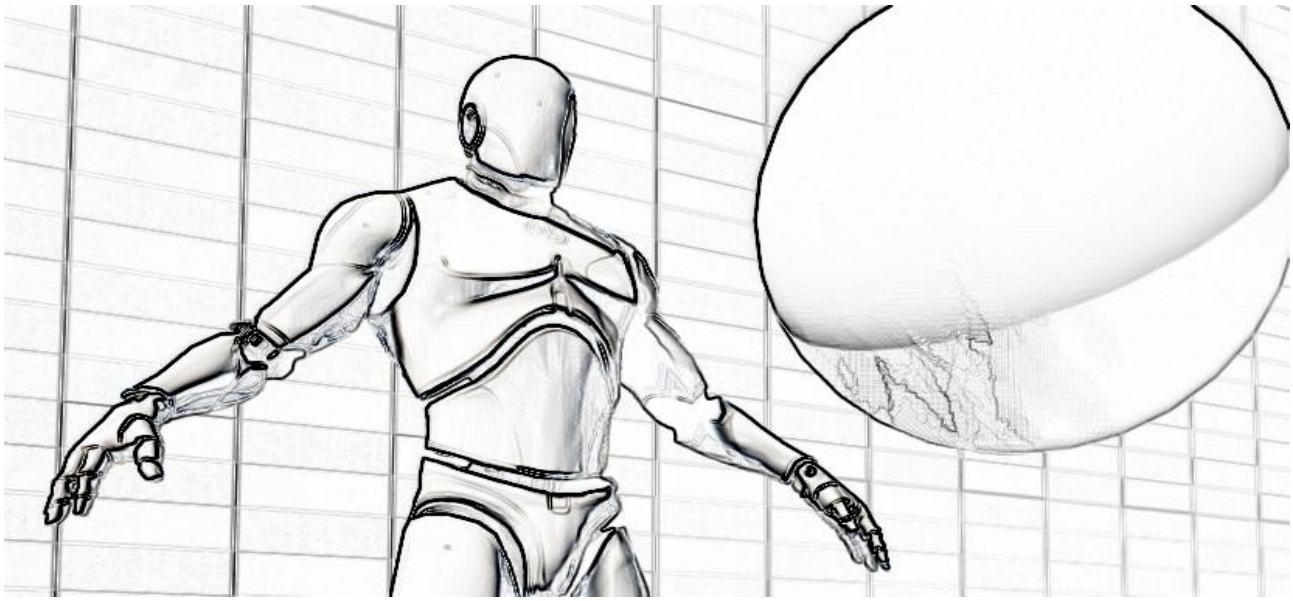

Рисунок 3.2 – Зображення після проходу алгоритмом Канні

На перший погляд вам може здатись, що все прохід був успішний і в такому випадку ви будете праві. Проте на даному рисунку занадто багато шуму

і деякі регіони виділені занадто детально. Проблему простіше замітити на результаті після накладання методом множення[5, 22] (див. рис. 3.3):

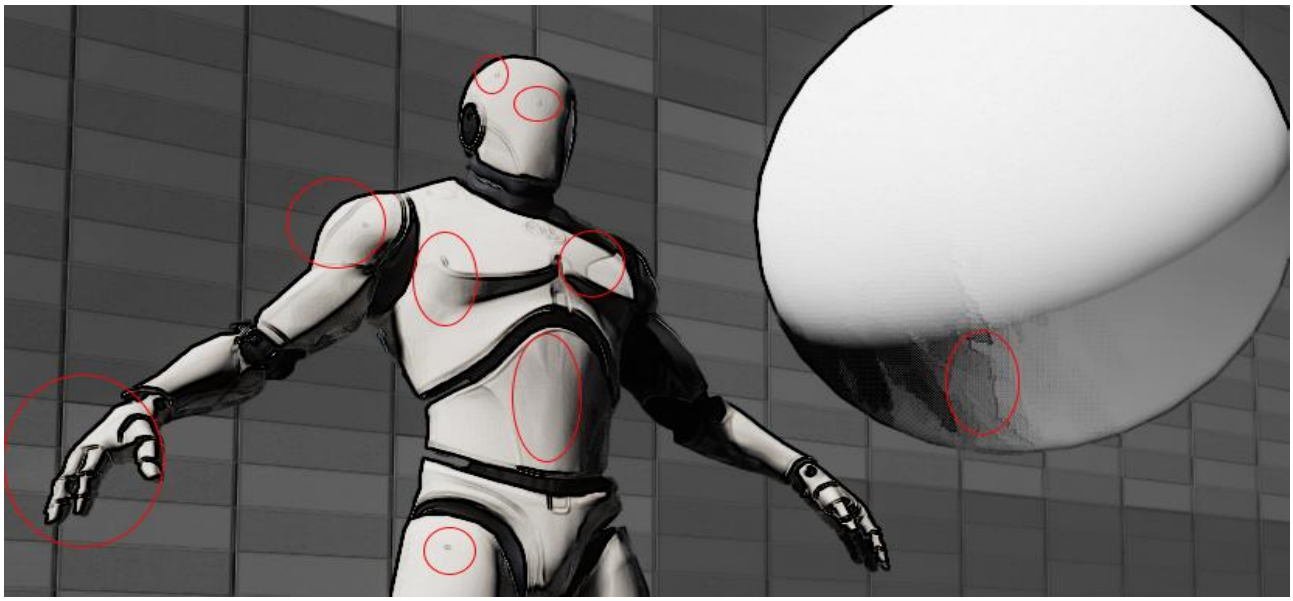

Рисунок 3.3 – Зображення з обведеними контурами

Я обвів червоними кільцями зони які, на мою думку, виглядають погано. Ми бачимо на плоских поверхнях, що фільтр обвів багато дрібних деталей, що в свою чергу привело до перенасичення зображення. Також фільтр обвів такі деталі як відблиски на поверхні манекену.

Щоб отримати більш прийнятний результат, ми повинні накласти фільтр Канні на карту глибин, саму карту глибин я отримав з допомогою вмонтованих функцій рушія “Unreal” (а саме, через “SceneTexture(SceneDepth)”), проте оригінальний результат видавав тільки білу картинку і мені довелось методом підбору провести кілька математичних дій(див. формулу 3.1) над зображенням, щоб отримати наступний результат[22] (див. рис. 3.4):

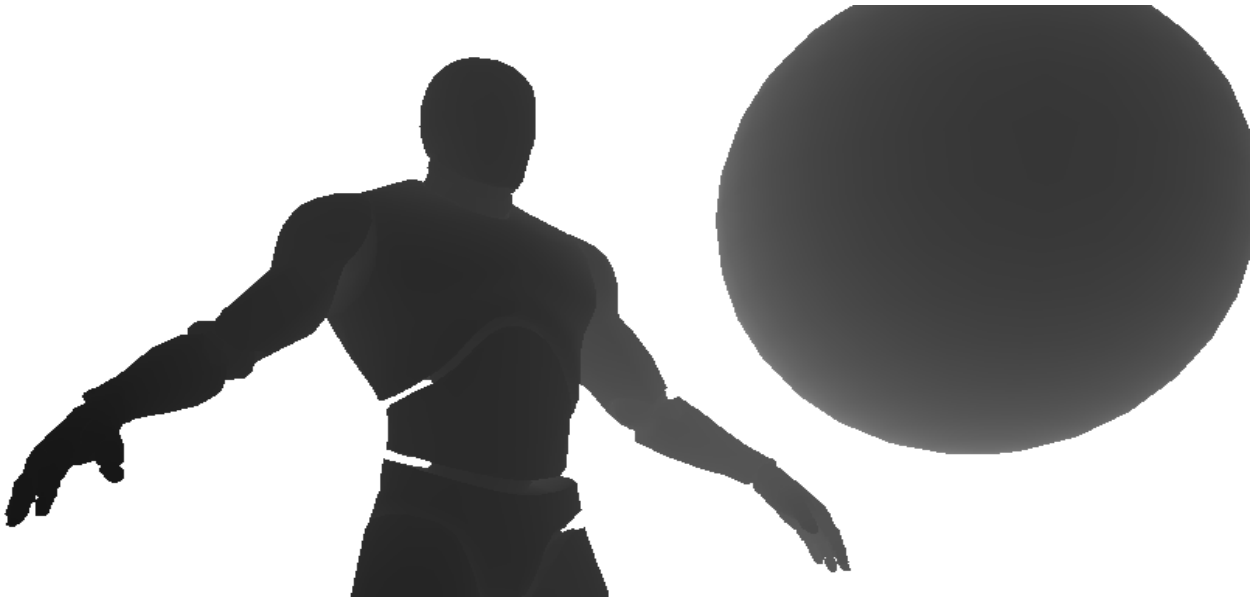

Рисунок 3.4 – Отримана карта глибини

Щоб отримати саме таке зображення, мені довелось математично відрегулювати область яку ми будемо бачити на виході. Я помножив карту глибин на 0.003 і підвів до 4-того степеня (значення я відібрав вручну). Чим ближче множник до нуля, тим темніша карта глибин; чим вище степінь, тим контрастніша карта глибин. Пояснюється це тим, що в своєму початковому стані, значення на карті можуть бути більше 1, але при виведенні на екран ми бачимо тільки значення від 0 до 1 (0 це абсолютно чорний і 1 це абсолютно білий).

Далі ми застосовуємо фільтр Канні безпосередньо до карти глибин. Результат буде наступним[22](див. рис. 3.5):

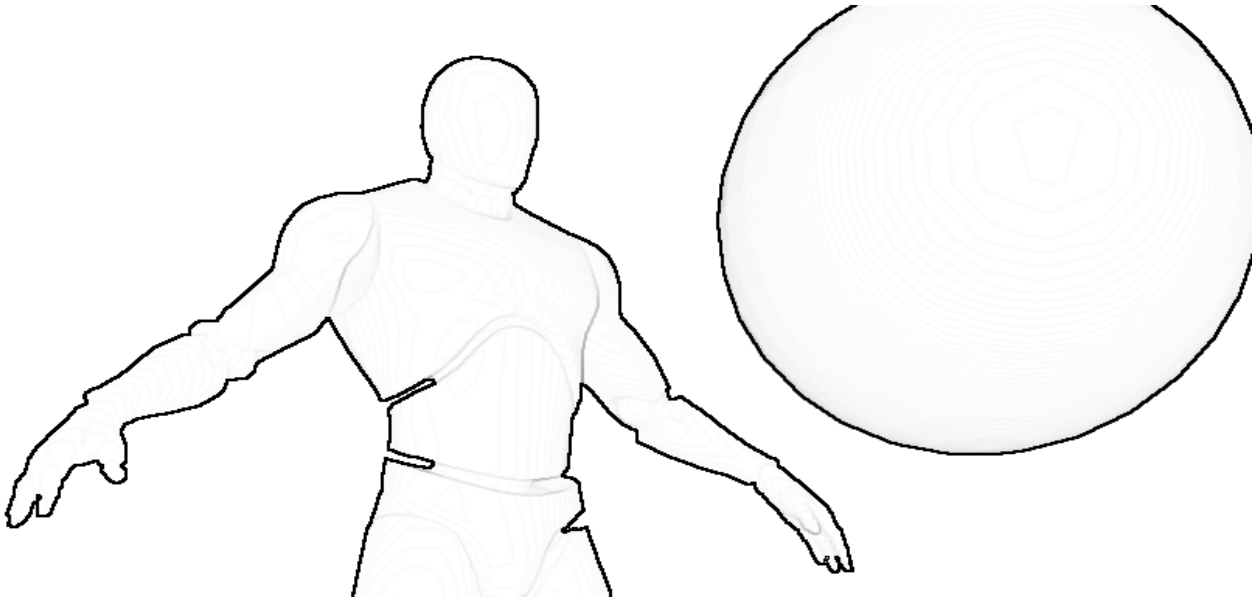

Рисунок 3.5 – Застосування фільтру Канні до карти глибин

Отриманий результат ми будемо вставляти в вхідне зображення за допомогою накладного методу множення: виходячи з поняття, що білий піксель рівний вектору (1, 1, 1) (червоний, зелений та синій кольори відповідно), то при множенні на любий інший підксель ми не отримаємо змін. Абсолютно чорний піксель з вектором (0, 0, 0) в свою чергу повністю замінить старий піксеь на такий же чорний. Результат такого накладання можна бачити на наступному зображенні[22] (див. рис. 3.6):

Рисунок 3.6 – Фінальний результат накладання outline

В ході побудови шейдера я зробив певну кількість змінних, якими я можу міняти його параметри при створенні пресетів. Також я створив наступну формулу, по якій працює цей шейдер (див. формулу 3.1).

$$S1(i,\ DM) = Canny((DM * 0.003)^4) * i\ , \qquad (3.1)$$

де *S1* – це вихідне зображення, функція шейдера,
i – це вхідне зображення,
DM – це маска глибини картинки,
Canny – це функція фільтру Канні.

3.1.2 Cell shading

Cell shading є простішим за outline із-за того що нам не потрібно опиратись на роботу якоїсь математичної функції (як наприклад фільтр Канні у випадку з outline).

Для подальшої розробки я вирішив поміняти колір манекену на червоний. На верхній половині наступної картинки показано зображення без моїх шейдерів, на нижній половині показано альбедо вхідного зображення (істинний колір без будь-якого впливу світла, його можна отримати через спеціальну шейдерну функцію “SceneTexture:BaseColor(for lighting)”)[2,22] (див. рис. 3.7):

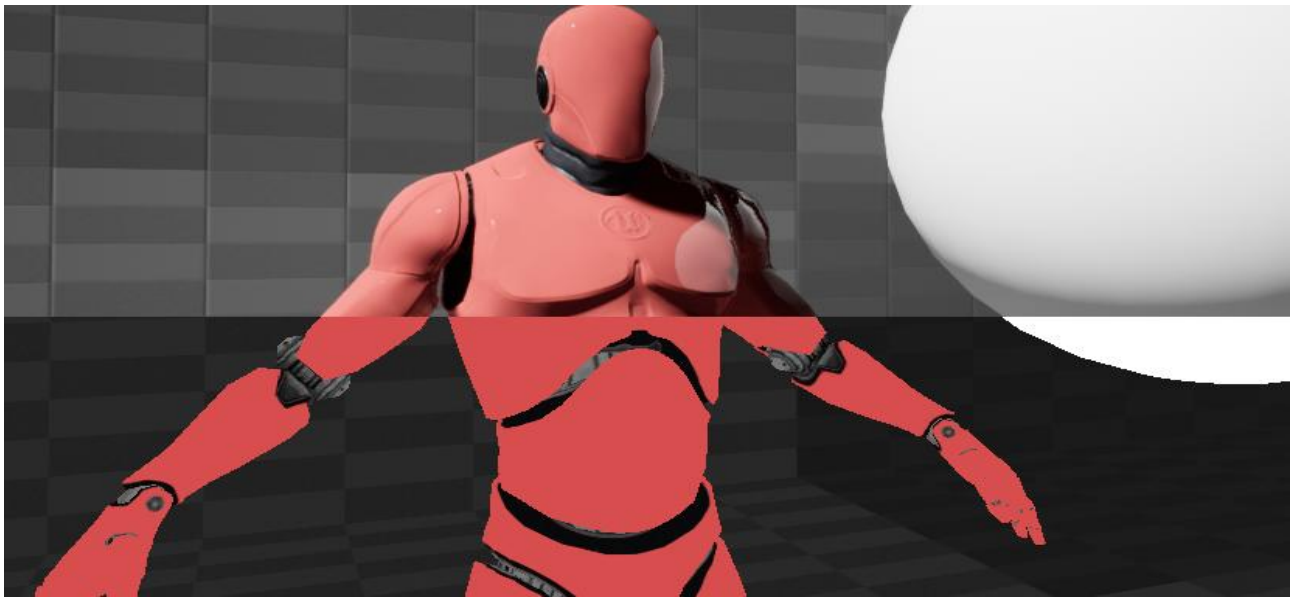

Рисунок 3.7 – Оригінал та альбедо об’єктів на зображенні

Для досягання потрібного мені ефекту, я маю провести постеризацію тільки над ефектами світлотіні (Detail Lighting) сцени, для цього мені спочатку потрібно відокремити самі ефекти світлотіні. Проводиться це з допомогою ділення зображення сцени, на його альбедо. Результат такої дії можна спостерігати на наступному зображені [22](див. рис. 3.8):

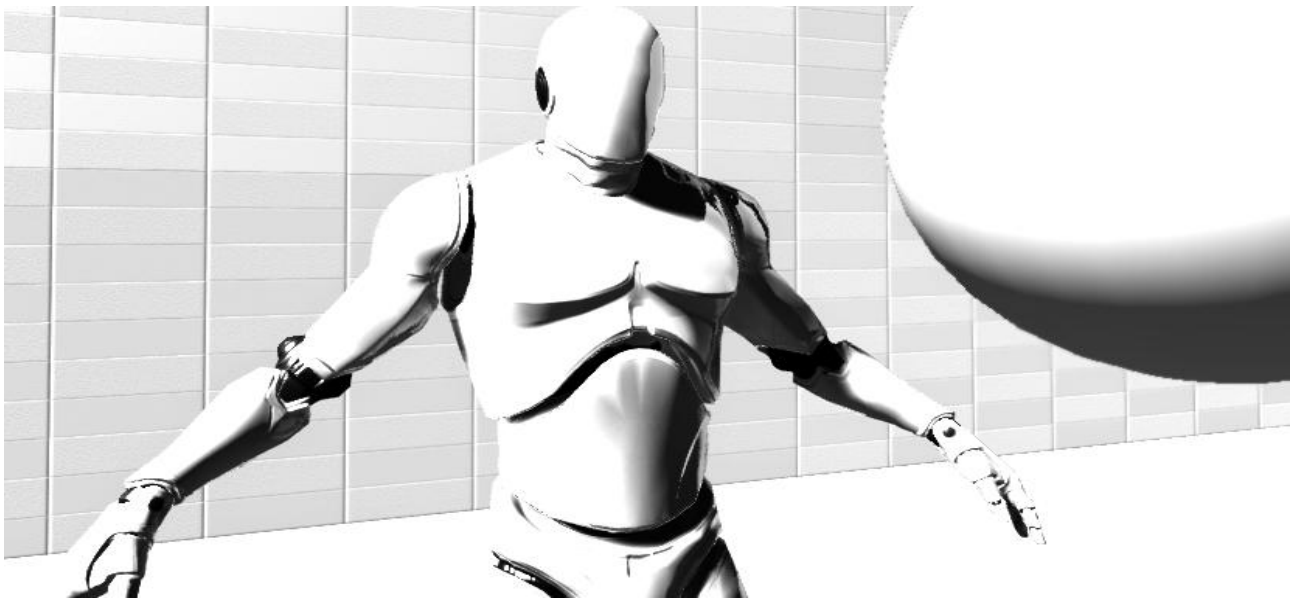

Рисунок 3.8 – Зображення без шейдерів поділене на альбедо

Далі ми проводимо пастеризацію відтінків сірого над отриманим зображенням [22](див. рис. 3.9):

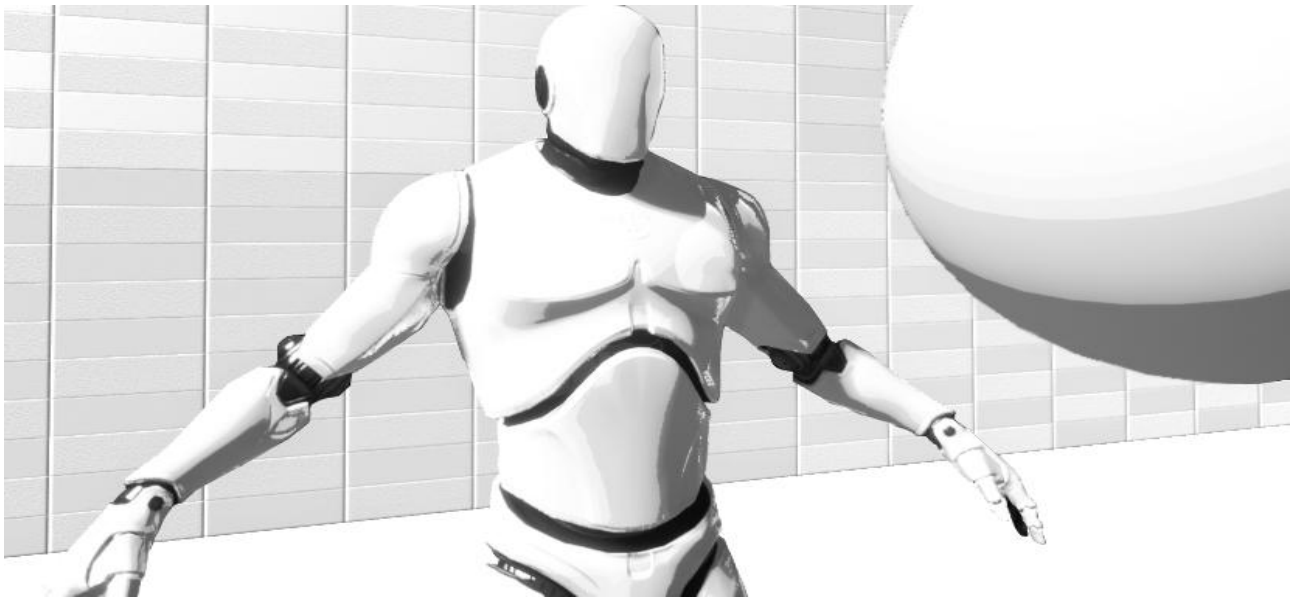

Рисунок 3.9 – Постеризоване зображення без кольорів

Наступним кроком буде повернення альбедо назад на зображення, для цього ми множимо пастеризоване зображення на альбедо[22](див. рис. 3.10):

Рисунок 3.10 – Фінальний результат роботи рейдера

Отриманий результат є задовільним і можна переходити до наступного етапу розробки гри. Cell shading і outline між собою не конфліктують при накладенні один на одного. Також я розробив функцію, яка репрезентує роботу цього шейдера (див. формулу 3.2).

$$S2(i,\ Alb, p) = \text{int}(\frac{i}{Alb*p}) * p * Alb, \quad (3.2)$$

де *S2* – це вихідне зображення, функція шейдера,

i – це вхідне зображення,

Alb – це альбедо зображення,

p – це ступінь пастеризації.

3.2 Розробка головного героя та інтерфейсу гравця

Цей етап є важливим, оскільки представлені в ньому функціональні елементи попадаються в першу чергу.

3.2.1 Головне меню

Головне меню це перше що бачить людина, коли відкриває гру (якщо ігнорувати можливі відео-заставки та логотипи компаній перед ним). Я вирішив не робити інтерфейс сильно продуманим в візуальному плані у зв'яку відсутністю часу та такої потреби. Також я остаточно опріділився з назвою гри: "Epic PNU Game" та переводится як "Епічна Гра ПНУ" – це своєріндна гра слів[7], оскільки розробником рушія є компанія "Epic Games".

При відкритті гри ми будемо бачити на передньому плані власне саме меню, а на задньому плані я розмістив сцену з плакатом, на якому повільно

анімується скріншот з титульною сторінкою титульної сторінки диплому, тут було використано такий художній засіб як “руйнування четвертої стіни”, що в свою чергу представляє звернення автора того чи іншого твору безпосередньо до споживача.

Саме меню побудоване на фремворці віджетів в “Unreal”. Widget Blueprints – це інтерфейс в кому можна працювати з “Motion Graphics” які будуть розташовуватись на інтерфейсі гравця або в тривимірному середовищі. В ньому є кнопка нової гри, кнопка запуску тестового рівня, кнопка налаштувань та кнопка виходу з гри, результат ви можете бачити далі (див. рис. 3.11):

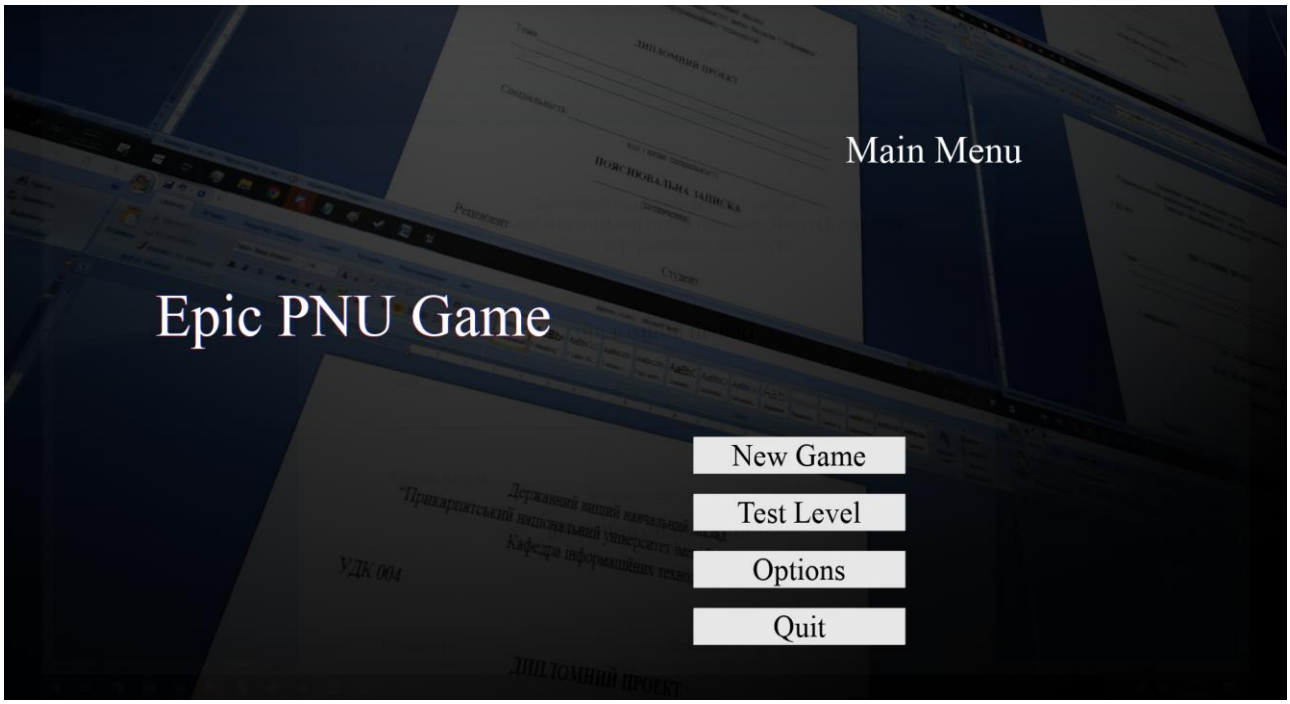

Рисунок 3.11 – Головне меню гри

Після натискання на кнопку налаштувань, можна побачити інше меню через яке ви можете обрати підрозділ налаштувань, а саме налаштування гри, налаштування відео та налаштування звуку(див. рис. 3.12. Нижче показано меню налаштування гри, ви тут міняєте значення які міняють змінні таких налаштувань: Розмір прицілу, колір прицілу (червоний, зелений, синій та прозорість), чутливість мишки, поле зору (кут поля зору – чим вище, тим

більше камера гравця захоплює зображення, при високих значеннях появляються викривлення тому значення є обмеженими, це значення є альтернативою до “розміру лінзи” яке вказується розміром лінзи в міліметрах замість кута в градусах), автоматична зміна зброї (якщо включено, гравець буде автоматично міняти зброю на іншу при повному вичерпанні боєкомплекту), автоматична перезарядка (якщо включено, гравець буде автоматично перезаряджати зброю при вичерпанні патроні в магазині зброї)(див. рис. 3.12).

Рисунок 3.12 – Налаштування ігрових параметрів

Представлені зверху значення взаємодіють з новими параметрами, які я розробив, проте параметри відео налаштувань, які представлені нижче, вже є параметрами рушія. В списку є такі параметри(див. рис. 3.13).:

- Роздільна здатність – розмір вікна гри в пікселях
- Співвідношення сторін монітора (не працює в віконному режимі)
- Скалювання роздільної здатності – зміна роздільної здатності без зміни розміру інтерфейсу та вікна гри.
- Якість ефектів пост обробки (наявно 4 варіанта вибору)
- Якість згладжування пікселів (наявно 4 варіанта вибору)
- Кількість проходів при обчислюванні тіней (наявно 4 варінта вибору)
- Частота кадрів (наявно 4 варіанта вибору)

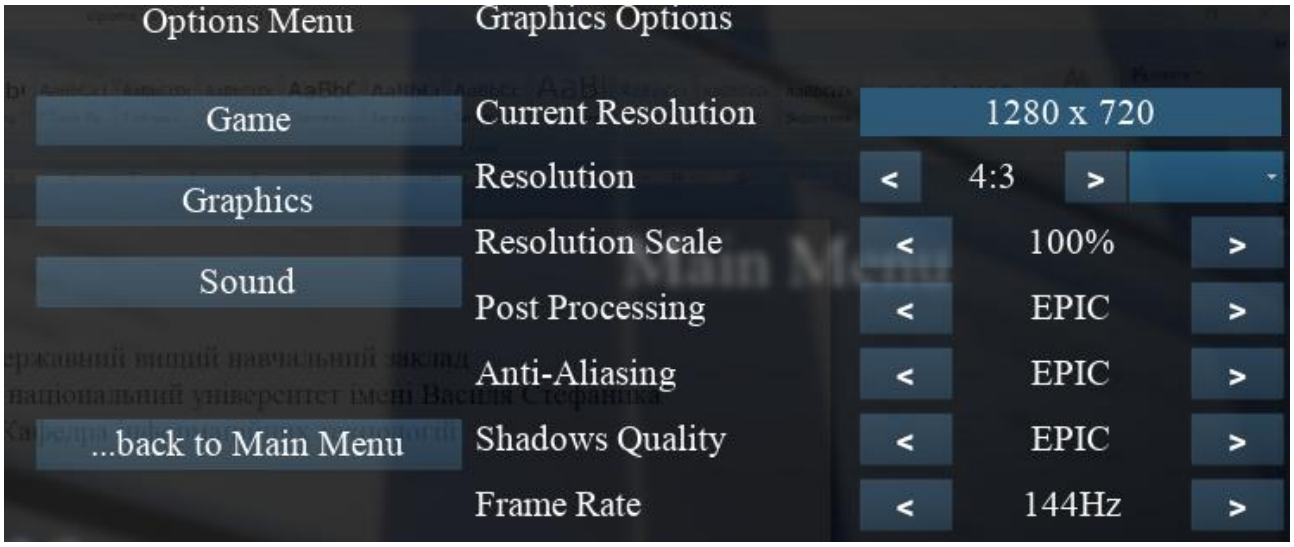

Рисунок 3.13 – Налаштування відео-параметрів

Нижче показано елементи управління гучністю. Можна вказати потужність загального звуку, потужність музики та потужність звукових ефектів(див. рис. 3.14).

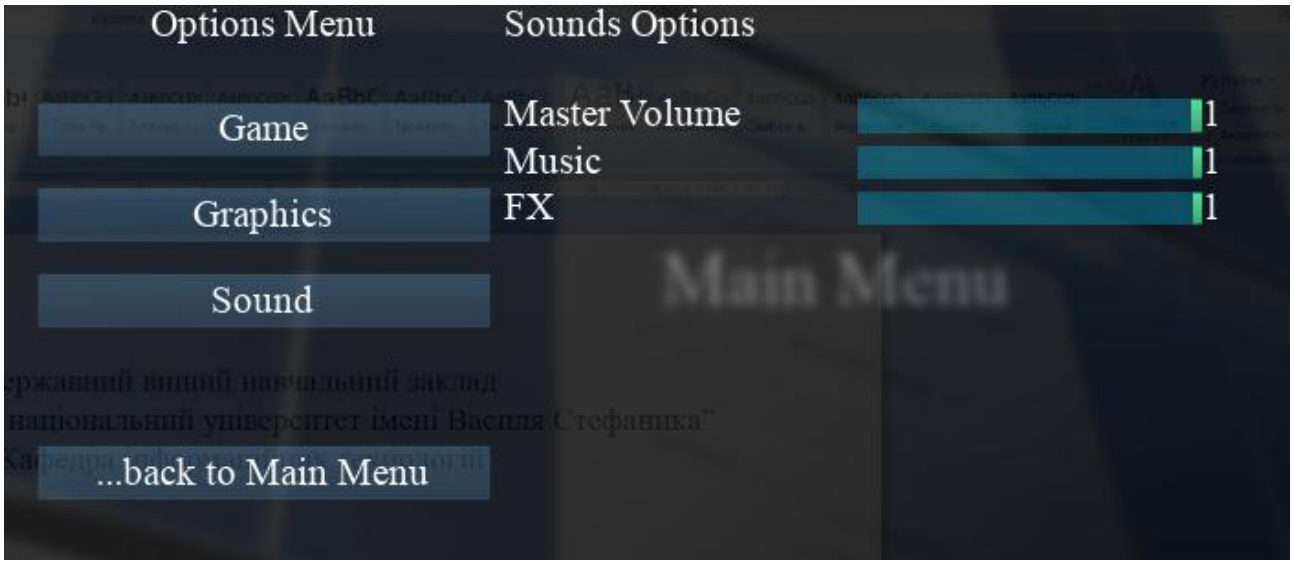

Рисунок 3.14 – Налаштування гучності звуків

При виборі параметрів налаштувань я опирався на тенденції ігрової індустрії. Віджети головного меню знаходиться на окремому рівні, який запускається першим при включенні гри, і звідси ми буде переходити на рівень

з генерацією карти. Шрифт тексту меню – Times New Roman. Для перегляду програмної реалізації, звертайтесь до додатку Б, пункту 10 або до додатку Г.

3.2.3 Переміщення гравця та його зброя

Перед розробкою безпосередньо функціональної частини гри, я вирішив повністю переключитись на візуальну мову програмування для оцінки і порівняння її з більш традиційними мовами програмування. Також на цьому етапі я добавив цю оцінку до свого персонального списку завдань цієї наукової роботи. До такого рішення я прийшов саме в цей момент із-за того, що попередні етапи розробки не передбачали використання C++ в принципі.

В рушії вже присутня база персонажа від першого лиця, і тому мені потрібно її налаштувати і добавити до неї функціонал. Я створив дочірній клас від класу стандартного персонажа і для початку я вказав такі параметри, як швидкість пересування, висота стрибку, максимальний кут нахилу поверхні пересування та зчеплення з поверхнею пересування. Далі я створив додаткові параметри, в яких вказується тип зброї в руці, реалізував механіку подвійного стрибка (гравець може додатковий раз стрибнути в повітрі) і ще я добавив анімацію хитання камери при приземленні гравця на землю після стрибка. Також я добавив запуск циклічної функції колихання камери при ходьбі, яка позволяє імітувати більш реалістичне пересування гравця. Ця функція включається тільки тоді, коли гравець натискає на кнопки пересування і знаходиться на землі.

Подвійний стрибок дозволяє змінити напрямок польоту в повітрі в залежності від керування гравця або зберігати стару інерцію якщо жодна кнопка не натиснута. Також, по задумці гравець пересувається дуже повільно дуже слизький підлозі, але стрибки позволяють прискорювати гравця. Для

перегляду програмної реалізації, звертайтесь до додатку Б, пункту 2 та до додатку В.

Наступним етапом стало створення зброї, та об’єктів з якими взаємодіє гравець. Для зброї ближнього бою я не став шукати унікальної моделі і просто використав модель дробовика, для її використання потрібно нажати клавішу “1” (при цьому гравець починає гру з пістолетом в руках, який іде другим по прорядку). При замаху, зброя чекає на перетин іншого об’єкта з її уявним кубом зіткнення, і якщо такий перетин стається, то цей об’єкт отримує пошкодження в розмірі 50 одиниць життя. Об’єкт не може отримувати пошкодження, якщо у нього не запрограмована така можливість, проте він може отримувати фізичний удар і реагувати на нього (якщо ударити по манекену, який лежить на землі, то він колихнеться). Анімацію замаху було взято з вільного в доступі набору[9]. Зброю ближнього бою продемонстровано на наступному рисунку[13,22] (див. рис. 3.15):

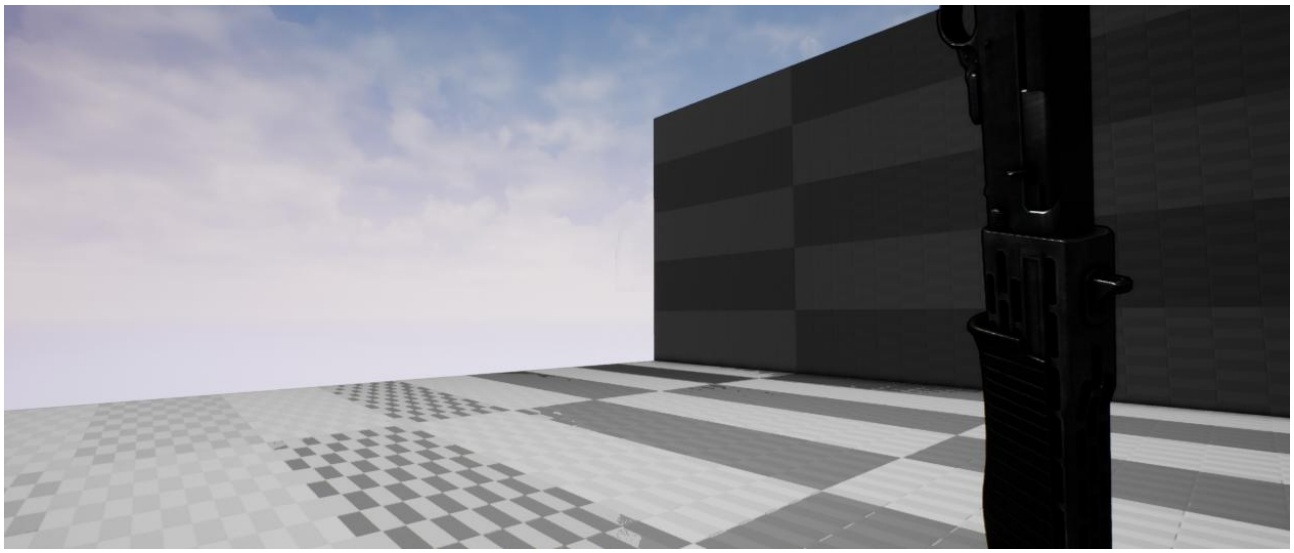

Рисунок 3.15 – Зброя ближнього бою

Кожен вид зброї є підкласом суперкласу зброї, гравець може носити з собою до 4 видів зброї (цю кількість можна збільшити, якщо розробити більшу кількість зброї). Далі я почав роботі над пістолетом. Знову, анімацію було взято з ресурсу з вільним доступом, а моделі зброї була скачані з безплатного

ресурсу, щоб вибрати пістолет в грі, потрібно нажати клавішу “2”. Було розроблено 2 типи стрільби для пістолету.

При вистрілі пістолета в основному режимі: викидається використана гільза, програється анімація, відтворюється звук, запускається снаряд і симулюється система часток з набору. Снаряд летить зі швидкістю 9000 умовних одиниць рушія прямо в точку, яка знаходиться в центрі екрану і наносить 25 одиниць пошкоджень, кожен вистріл тратить 1 патрон. В обоймі може одночасно знаходитись 17 патронів, і при старті гравець появляється з 25 патронами в запасі.

Стрільба є не автоматичною, перед кожним наступним вистрілом відбувається затримка в розмірі 0,08 секунд. Гравець може стріляти зі швидкістю до 750 вистрілів на хвилину, якщо буде достатньо швидко натискати на курсор. Час перезарядки зброї займає 1,55 секунди. Кількість пошкоджень на секунду (DPS) становить 312.5 одиниць

Альтернативний режим тратить 15 патронів (і випускає 15 гільз), і не буде працювати якщо в обоймі їх є недостатня кількість. Другий режим стрільби запускає ракету з пістолета, яка летить зі швидкістю 5000 умовних одиниць та наносить пошкодження розміром 120 в радіусі 500 умовних одиниць. Якщо у гравця є повний заряд пістолету, то після вистрілу в альтернативному режимі, він ще зможе вистрілити 2 рази в основному режимі до вичерпання запасу. Ткож гравець може вистрілити під себе ракетою і отримати імпульс, який відкине його в сторону (цю механіку в іграх називають рокетджампом). Кількість пошкоджень на секунду в альтернативному режимі становить 129 одиниць пошкоджень.

Для перегляду програмної реалізації стрільби, звертайтесь до додатку Б, пункту 4. Для перегляду програмної реалізації перезарядки, звертайтесь до додатку Б, пункту 5. Використання зброї в основному режимі показано на наступному рисунку[11,18,22](див. рис. 3.16):

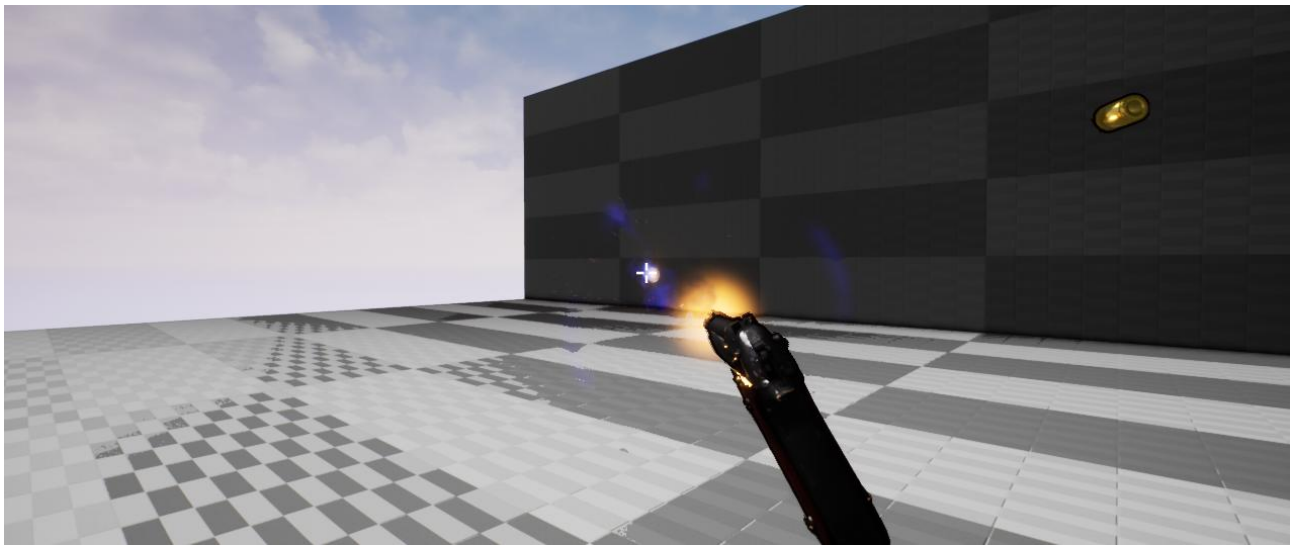

Рисунок 3.16 – Робота пістолета

Всі снаряди вилітають з дула зброї і летять в тому самому напрямку, що і дивиться гравець: технічно це не правильно, оскільки снаряд має летіти в центр поля зору, ви можете побачити відхилення в наступному рисунку(див. рис. 3.17):

Рисунок 3.17 – Стала похибка стрільби

Так, відхилення є, проте вони є незначними, чим дальше ціль від гравця, тим менше похибка з йог точки зору. Проте технічно похибка є сталою на любій відстані, оскільки напрям польоту снаряду є точним і похибка виникає тільки із-за зміщення точки вильоту снаряду трохи в право і в низ. Є 2 способи вирішення цієї проблеми:

- Використовувати систему прорахування влучань через промені
- Прорахувати нову початкову траєкторію снаряду, опираючись на наступну формулу(див. формулу 3.3),

$$Angle = \text{Ang}(A - B)\ , \qquad (3.3)$$

де *Angle* – це новий кут польоту снаряду,

Ang – функція конвертації вектора в кут рушія “Unreal” ,

A – точка в тривимірному просторі, на яку дивиться гравець,

B – точка в тривимірному просторі, з якої вилітає снаряд.

Я вирішив не виправляти цей недолік обґрунтовують тим, що похибка є невеликою, і також із-за того, що невідомо кути снаряд буде летіти, якщо гравець дивиться в небо, оскільки небо в рушії є абстрактним і не може мати на собі точку спостерігання гравця.

Далі я почави робити дробовик, у нього є 8 патронів на старті 0 в запасі. Затримка між вистрілами становить 0.5 секунди (30 вистрілів в хвилину) і час перезарядки становить 0.9 секунди (на відміну від інших шутерів, в цій грі дробовик перезаряджає всю обойму зразу). При попаданні па поверхню, появляється ефект горіння. Щоб вибрати дробовик, потрібно нажати на клавішу “3”. Кожен вистріл пипускає 7 снарядів, і кожен наносить 10 одиниць пошкоджень. Кількість пошкоджень на секунду становить 140 одиниць. Така висока різниця в порівнянні з пістолетом, пояснюється тим, що дробовик моментально наносить 70 пошкоджень, а пістолет поступово наносить свої пошкодження. Існує 2 таких поняття як “burst damage” та “sustained damage” розподілу пошкоджень за час, дробовик належить до першого, а пістолет до другого. Прийнято вважати, що “burst damage” є більш надійним і ефективним.

У дробовика присутній випадковий розкид пуль. Кожна пуля вилітає з таким же початковим відхиленням як і у пістолета, проте додатково добавляється додаткові зміни, унікальні для кожної кулі. Прораховування розкиду пуль відбувається з допомогою наступної формули(див. формулу 3.4),

$$Direction = (x + \mathrm{R}_1(-r, r), y + \mathrm{R}_2(-r, r), z + \mathrm{R}_3(-r, r), \qquad (3.4)$$

де $Direction$ – це новий вектор польоту дробинки,

R(a,b) – це функція генерації випадкових чисел між a та b,

x – це кут камери гравця по осі x,

y – це кут камери гравця по осі y,

z – це кут камери гравця по осі z.

Максимальне відхилення снарядів становить 5 градусів. Також при кожному вистрілі викликається функція колихання камери, яка імітує реакцію гравця на вистріл зброї в руках. Результати вище описаної роботи можна побачити на наступному скріншотіх[13](див. рис. 3.18):

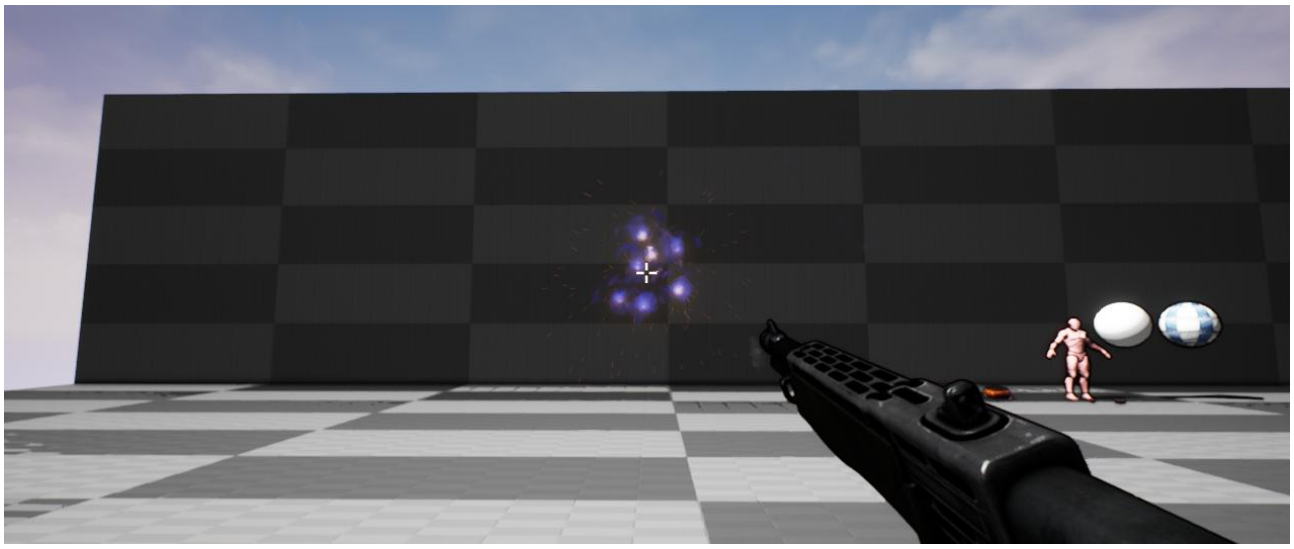

Рисунок 3.18 – Демонстрація розкиду дробинок

Далі я почав роботу на гвинтівкою. У неї є обойна на 54 патрони і 0 патронів в запасі на початку. Час перезарядки становить 0.9 секунди, а режим здійснення стрільби є автоматичним. Кожен снаряд наносить 15 одиниць пошкоджень, при цьому швидкість снарядів,швидкість стрільби та розкид пуль залежать від того, на скільки довго гравець стріляє зі зброї без зупинки. Початкова затримка між вистрілами становить 0.3 секунди (200 вистрілів в

хвилину). З кожним вистрілом, частота збільшується на 15%, при цьому затримка між вистрілами не може стати нижче 0.1 секунди (600 вистрілів на хвилину). Далі показану формулу, за допомогою якої обчислюється час, за який зброя досягає максимальної кількості вистрілів на хвилину(див. формулу 3.5),

$$T(A,S) = \sum_{S*A\geq 0.1}^{I=1} A * (\prod_{I}^{1} S), \quad (3.5)$$

де *T(A,S)* – це функція обчислення часу “розжарювання” зброї,
A – це початкова затримка між вистрілами,
S – це коефіцієнт зниження затримки між вистрілами.

Використовуючи цю формулу, я визначив що знадобиться 1.455 секунди для повного розгону зброї, також за цей проміжок відбудеться 8 вистрілів. Зброя стартує стрільбу з показником 50 пошкоджень на секунду, при цьому за першу секунду стрільби, зброя наносить 90 одиниць пошкоджень. Після повного розгону зброя починає наносити 150 одиниць пошкоджень на секунду.

Початкова швидкість снаряду становить 3000 умовних одиниць. При збільшенні швидкості снаряду використовується той самий коефіцієнт, що і при визначенні швидкості стрільби, тільки у вигляд дільника а не множника. Швидкість польоту зростає аж до 12000 умовних одиниць, що потенційно робить цю зброю власником самих швидких куль.

Точність зброї також збільшується в залежності від часу стрільби, початкову точність стрільби можна описати конусом з радіусом 4.5 градусів. Кінцева похибка точності становить 1.5 градуса. Для перегляду програмної реалізації, звертайтесь до додатку Б, пункту 3.

Також будо зроблено альтернативний режим стрільби. При натисканні на праву клавішу миші, відбувається затримка довжиною 0.25, після якої здійснюється черга пострілів довжиною в 10 снарядів. Кожен такий снаряд наносить 20 одиниць пошкоджень, швидкість польоту становить 6000 умовних

одиниць, а довжина затримки між пострілами становить 0.05 секунди. Також кожен снаряд може відстрибнути від будь якої поверхні, крім ворога, після першого влучання в стіну запускається таймер який через певний проміж уток часу знищує снаряд. В теорії такий залп може нанести до 200 умовних одиниць пошкоджень за 0.75 секунди, якщо всі кулі попадуть в ціль. Кількість нанесених пошкоджень в секунду становить 266.67 умовних одиниць. Використання зброї показано на наступному рисунку[21](див. рис. 3.19):

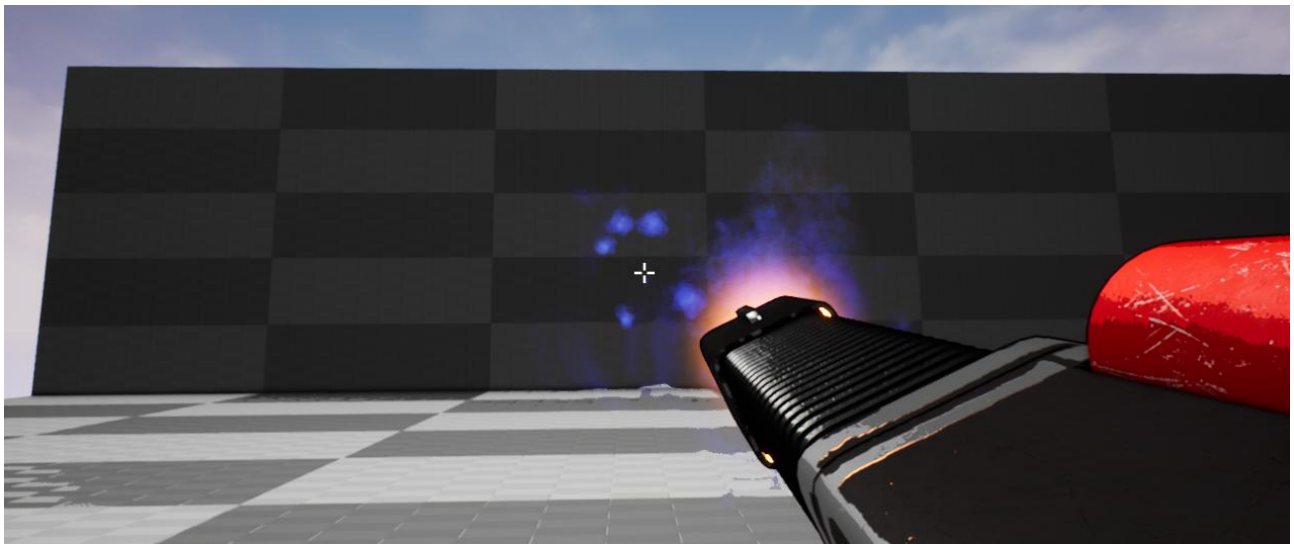

Рисунок 3.19 – Демонстрація гвинтівки

Я розробив систему поповнення патронів, коли гравець вбиває однією зброєю, то він отримує патрони для іншої, таким чино гравець буде змушений постійно міняти свою зброю.

Також для кожного типу боєприпасів та для одиниць здоров’я, було створено інтерактивні об’єкти, було створено декілька різновидів однакових інтерактивних предметів з різною кількістю припасів. Також можна окремо створювати інтерактивну версії зброї, при взаємодії з якою користувач або отримує патрони від неї, або отримує саму зброю безпосередньо. Ще можна включити циклічну анімацію обертання зброї в повітрі (див. рис. 3.20):

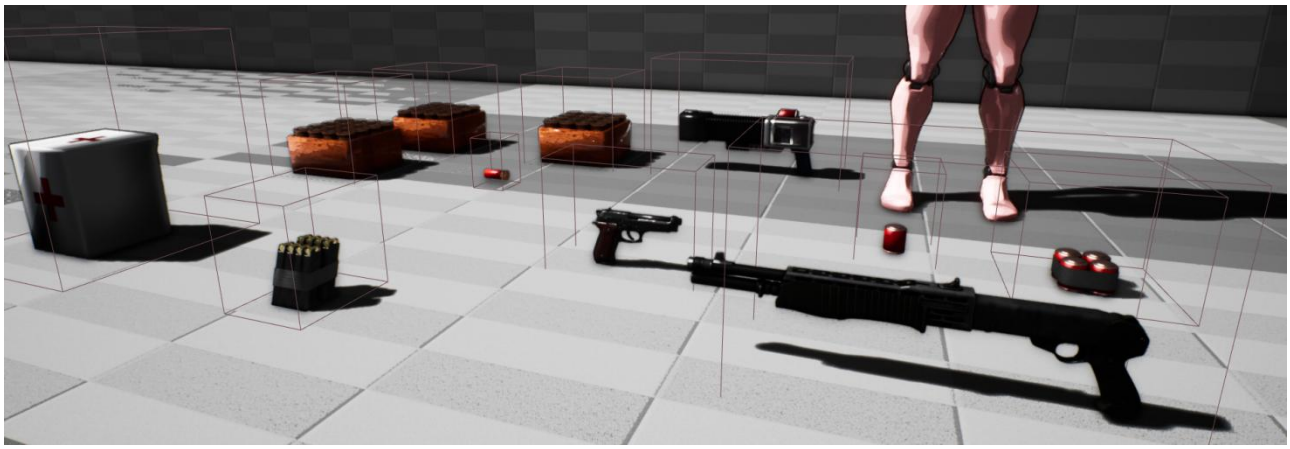

Рисунок 3.20 – Інтерактивні предмети

Далі я розробив систему, яка заміняє покадрову анімацію від першого лиця. При повороті гравця за допомогою пересування курсору, також здійснюється пересування зброї на крані та здійснюється нахил камери в ліво або вправо.

Було розміщено 3 точки руху, перша і друга: A1 та A2 відповідно, в камері і третя A3 в зброї. Я розмістив предмети інтерфейсу предмети в ієрархії переміщення (дочірній “прикріпляється” та слідує за батьківським, хоча він також може пересуватись в своїх відносних координатах) наступним чином: A1 далі камера, далі A2, далі A3 і на сам кінець зброя. При русі курсору, прораховується змінна R1, значення якої постійно прямує з певною швидкістю до, значення напрямку курсору R1Z. Також, чим більша різниця між реальним кутом погляду та анімацією, тим більша швидкість прямування R2 до R2Z.

Точка руху A2 повертається в протилежну напрямку від R2: таким чином A2 повністю компенсує горизонтальний поворот камери (якщо виключити наближення R2 до R2Z) і в результаті ми зможемо замітити, що зброя в інтерфейсі рухається разом з гравцем, проте при поверненні камери в ліво або в право, зброя буде залишатися в тій самій точці в просторі.

При включенні постійного наближення R2 до R2Z ми замітимо що при повороті камери, зброя слідує за нею але з запізненням – це саме те що нам

потрібно і навіть цього могло бути достатньо для імітації покадрової анімації, але я вирішу модифікувати систему.

Для точки руху A3 було створено свої R3 до R3Z які діяли по принципу R2 до R2Z. A3 в свою чергу замість того щоб компенсувати R3 стала рухатись в тому самому напрямку з множником 1.5.

Результатом цієї маніпуляцію стала математична анімація в якій зброя запізнювалась за поворотом камери але, напрямок її дула повертався в ту сторону, в яку вона рухається. Також A1 повертається виходячи зі значень R2, що в свою чергу створює анімацію нахилу камеру на той бік на який вона повертається. Для перегляду програмної реалізації, звертайтесь до додатку Б, пункту 1. На наступному рисунку демонтується приклад анімації, коли гравець різко повертає камеру в право(див. рис. 3.21):

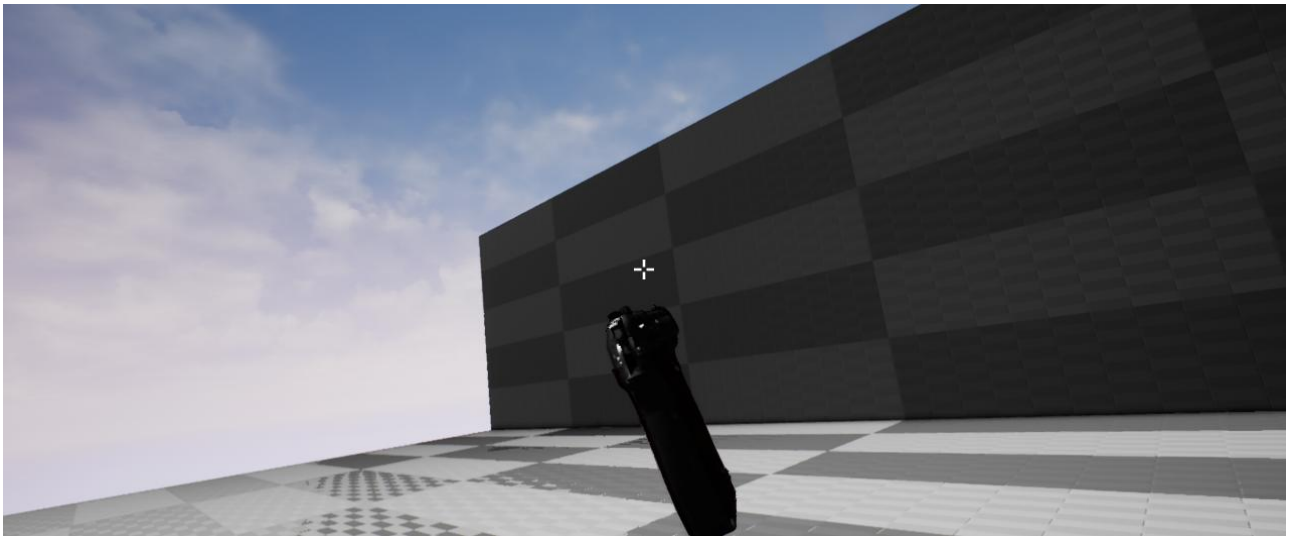

Рисунок 3.21 – Демонстрація повернення зброї та камери в залежності від руху гравця

3.2.2 Інтерфейс гравця

Я розробив інтерфейс на якому показано кількість патронів, життя та кількість балів. Я також вирішив добавити “кінематографічні чорні полоси”.

Кількість балів демонструються за допомогою анімованого шрифта, здоров’я та патрони відображаються за допомогою шрифта Times New Roman. Кількість балів буде збільшуватись при вбивстві ворогів. Параметри перехрестя в центрі екрану можна модифікувати через меню налаштувань гри. Для перегляду програмної реалізації, звертайтесь до додатку Б, пункту 6. Сам інтерфейс продемонстровано на наступному скріншоті(див. рис. 3.22):

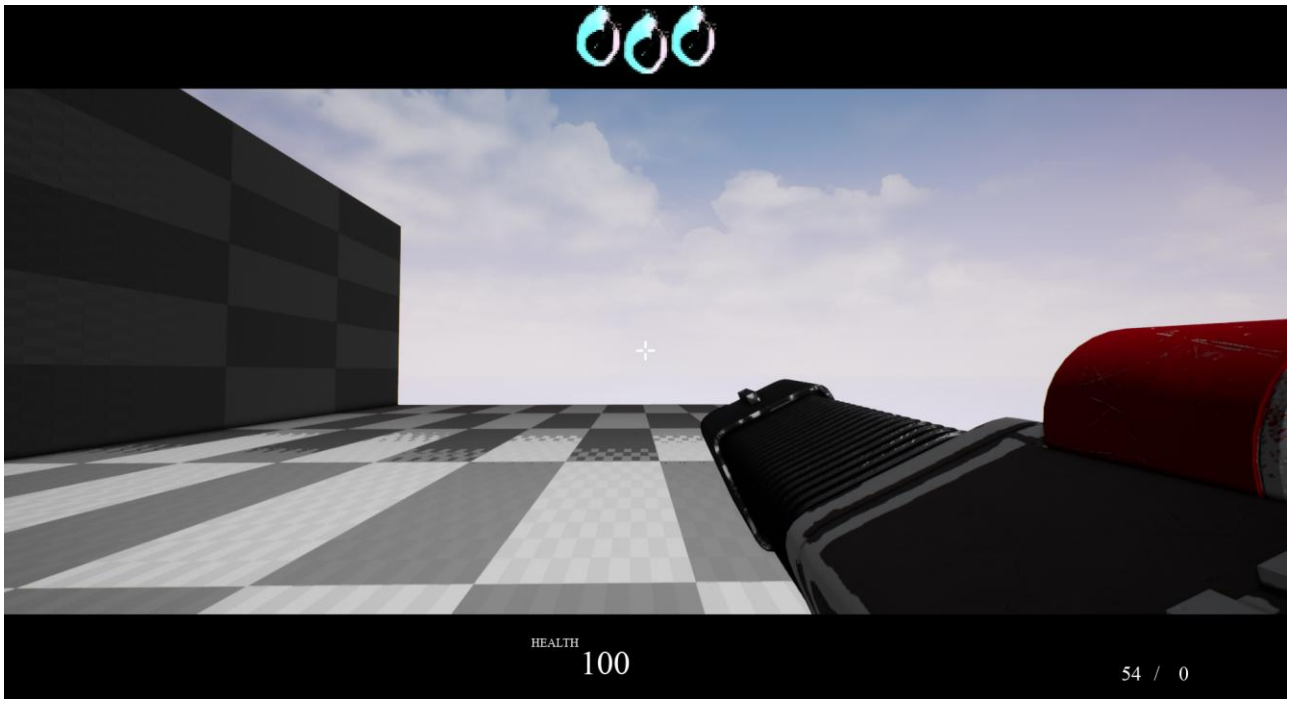

Рисунок 3.22 – Інтерфейс гравця

3.3 Прописування неігрових персонажів

Остаточним рішенням при розробці неігрових персонажів стала розробка трьох різних персонажів з різними механіками. Всі персонажі є підкласами нового суперкласу з функціями отримання пошкоджень, атаки, смерті, патрулювання та переслідування, кожна з них пояснюється далі[28]

- Функція переслідування заставляє персонажа іти до точки в радіусі 5 умовних одиниць рядом з ціллю, при цьому, функція рекурсивно викликає

сама себе і переключається на функцію атаки, коли умова відстані задовільна.

- Функція атаки для кожного наступного персонажа буде модифікованою, проте у суперкласа налаштовано наносити гравцю 10 одиниць пошкоджень кожні 0.95 плюс мінус 0.25 секунди.
- Функція патрулювання вибирає випадкову точку на навігаційному меші в певному радіусі і відправляє персонажа в неї.
- Функція отримання пошкоджень зменшує кількість життя персонажа. Якщо це значення стає меншим або рівним нулю, то викликається функція смерті.
- Функція смерті зупиняє всю дії персонажа, і в залежності від типу пошкоджень, від якої він помер, виконується один з двох сценаріїв. Якщо персонаж помер від звичайного типу пошкоджень, то включається фізика скелету персонажа і йому задається імпульс від попадання. Якщо, в свою чергу, він помер від вибухового типу пошкоджень, то його розриває на шматки (цей сценарій може відбутися звичайної смерті).

Також до цього суперкласу було добавлено клас чуття, який дозволяє отримувати різну інформацію з навколишнього середовища, таку як реакція на абстрактний звук та список об’єктів в полі зору. Коли гравець потрапляє в поле зору, викликається функція, яка зберігає гравця як ціль і запускає функцію переслідування. Якщо гравець видає своє місцезнаходження шумом, то персонаж іде в місце з якого лунав звук.

Першим з розроблених ворогів став персонаж ближнього бою, якого можна бачити на середині представленого рисунку[26] (див. рис. 3.23). Персонаж біжить до гравця зі швидкістю 1000 умовних одиниць і має таку саму функцію пошкоджень, за винятком того, що він наносить 25 одиниць пошкоджень. Було використано геометричну модель та анімацію з “Unreal Marketlapce”. За вбивство дається 1 бал. Для перегляду програмної реалізації ближнього бою, звертайтесь до додатку Б, пункту 7.

Наступним було уроблено персонажа з дробовиком. Він пересувається зі швидкістю 400 умовних одиниць. При отриманні зорового контакту він буде стріляти з допомогою прорахування променів (тобто без снарядів): з дула зброї буде випущено 6 променів відхиленнями в 2 градуси. Якщо промень попадає в тіло гравця, то він отримує 7 одиниць пошкоджень (що в сумі може становити до 42 одиниць за вистріл). В наведенні зброї присутня затримка, тому попасти в гравця буде складніше якщо він буде рухатись. Було використано геометричну модель та анімацію з “Unreal Marketlapce”. За вбивство дається 2 балів. Для перегляду програмної реалізації логіки персонажа, звертайтесь до додатку Б, пункту 8. Його можна бачити на праворуч на представленому рисунку[15] (див. рис. 3.23).

Останні персонаж схожий по параметрам на другого, проте в нього є 2 моделі поведінки[31]:

- Якщо гравець знаходиться на відстані більше 500 одиниць, то він стріляє в нього чегргами з вогняних куль, кожна з яких наносить 20 одиниць пошкоджень, затримка між вистрілами становить 0.4 секунди, а затримка між чергами.
- Якщо гравець знаходиться в радіусі 500 одиниць, то персонаж використовує вогнемети, які наносять 60 пошкоджень в секунду.

За вбивство дається 5 балів. Персонаж знаходиться ліворуч на представленому рисунку[15, 26, 27] (див. рис. 3.23).

Рисунок 3.23 – Неігрові персонажі

Також при я реалізував систему поповнення життя гравця через вбивство ворогів, кожна ліквідація відновлює 10 одиниць життя. Далі я налаштував навігацій меш, який використовує штучний інтелект для пошуку шляху. Персонажі можуть пересуватись на зеленій зоні(див. рис. 3.23):

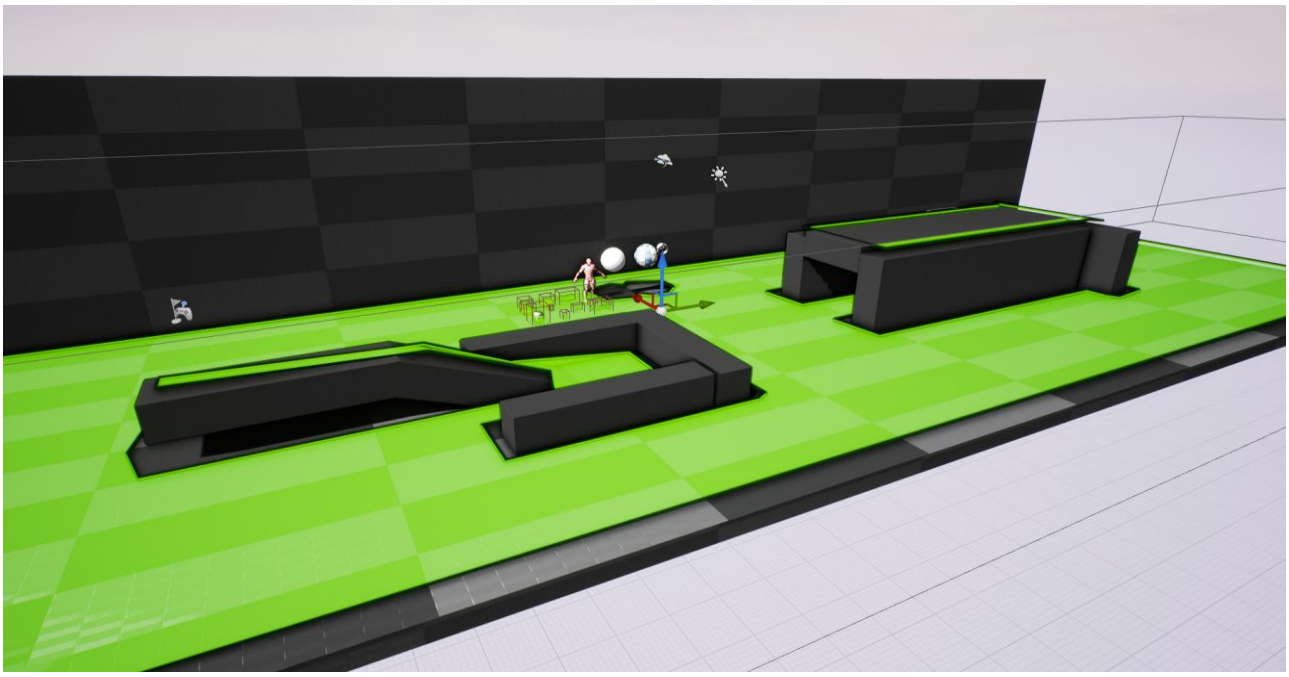

Рисунок 3.24 – Налаштований навігаційний меш

3.4 Розробка карт

Після закінчення підготовки елементів гри, я приступив до побудови карти гри, на яких будуть відбуватись усі дії. Спочатку потрібно розробити елементи, яких будується карта, а потім треба розробити алгоритм конструювання.

3.4.1 Побудова окремих елементів для генерації

Після вибору елементів з базового набору рушія, я почав їх упаковувати в спеціальні рівні наступним чином: один спеціальний рівень містить один макро-елемен (стіну, кімнату, тощо), декілька (кількість залежить від елемента) точок стикування та ймовірно різні динамічні об’єкти, такі як джерела світла, випадкові вороги з предметами або інші інтерактивні предмети. Також рівень зберігає в собі свою ширину та висоту, яка знадобиться в майбутньому.

Всього я було створено 13 таких спеціальних рівнів, а саме: кут, закручений поворот, тупик, широкий тупик, кімната зі спуском в іншу кімнату, перехід з вузького коридору в широкий підйом, прямий коридор, т-видне перехрестя, кімната, широкий коридор, х-видне перехрестя та широке х-видне перехрестя. На наступному рисунку показано х-видне перехрестя на якому є 4 точки стикування (групи стрілок) та елемент в якому зберігаються дані про кімнату [10](див. рис. 3.24):

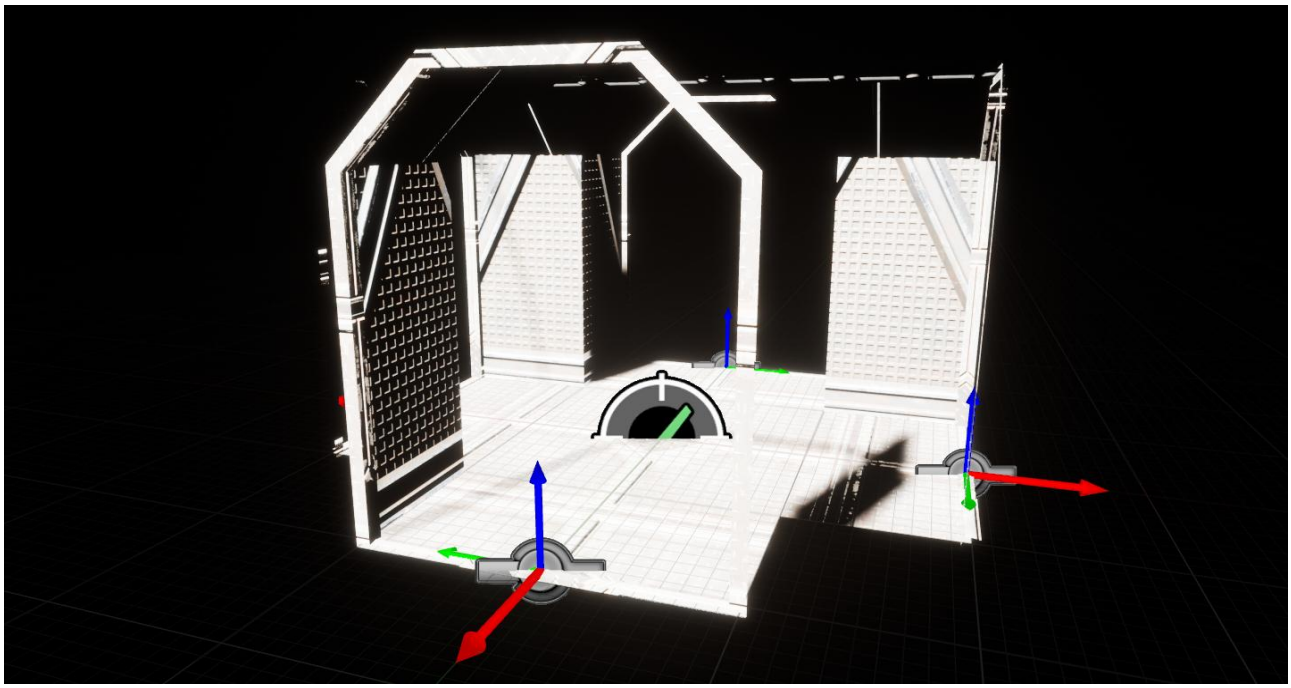

Рисунок 3.24 – Один із сегментів карти

Також я створив стрибкову платформу, яка буде підкидувати гравця з силою 100 одиниць вгору, якщо той захоче піднятись на верхній поверх в одній з кімнат (див. рис. 3.24). Ще було створено список з’єднань, в якому вказано які елементи можна приєднувати до інших (виходячи з думки, що коридори можуть бути вузькими або широкими. Для перегляду програмної реалізації, звертайтесь до додатку Б, пункту 9.

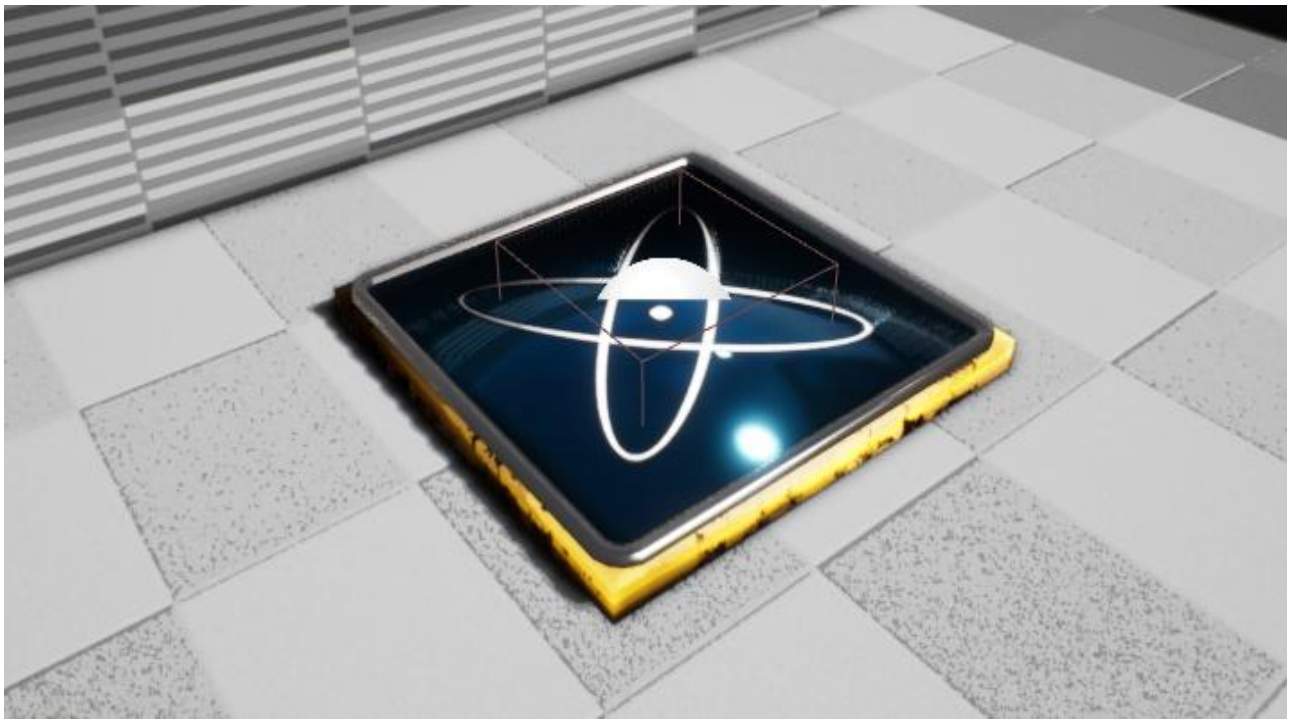

Рисунок 3.25 – Стрибкова платформа

3.4.2 Генерування середовища

Під час розробки системи генерації, я вирішив не опиратись на будь-які математичні алгоритми розглянути раніше. Я спочатку побудував її опираючись на більш примітивні принципи вирішив і оцінити результат такої генерації.

Спроектований мною простий алгоритм працює наступним чином. В параметрах конструктора вказується максимальний розмір карти, після чого в центрі ставиться випадковий об’єкт, з якого починається безпосередньо сама генерація. Далі до цього об’єкта пристиковується інший, який випадковим чином вибирається з списку можливих, об’єкт викреслюється зі списку, якщо він конфліктує з можливими сусідніми об’єктами (відбувається перевірка на перетин об’єкта між існуючими, також видаляються геометрично несумісні об’єкти). Далі у мене було 2 вибори розроби алгоритму: карта генеруються або в глибину або в ширину. Якщо карта генерується в глибину: то нові деталі

карти приєднують до самих нових слотів, якщо в ширину: то вони приєднуються до самих старих. Перший метод генерував кімнату з явним домінантним коридором, в якому інші коридори розгалужувались від нього. Другий метод генерував карту, що більше походила на лабіринт. Після опрацювання отриманої інформації, я вирішив використовувати другий метод. На цій частині роботи алгоритму у нас є каркас з точками, в яких мають бути розміщені безпосередньо самі елементи карт. Після розміщення самих елементів, ми отримуємо такий результат (див. рис. 3.26):

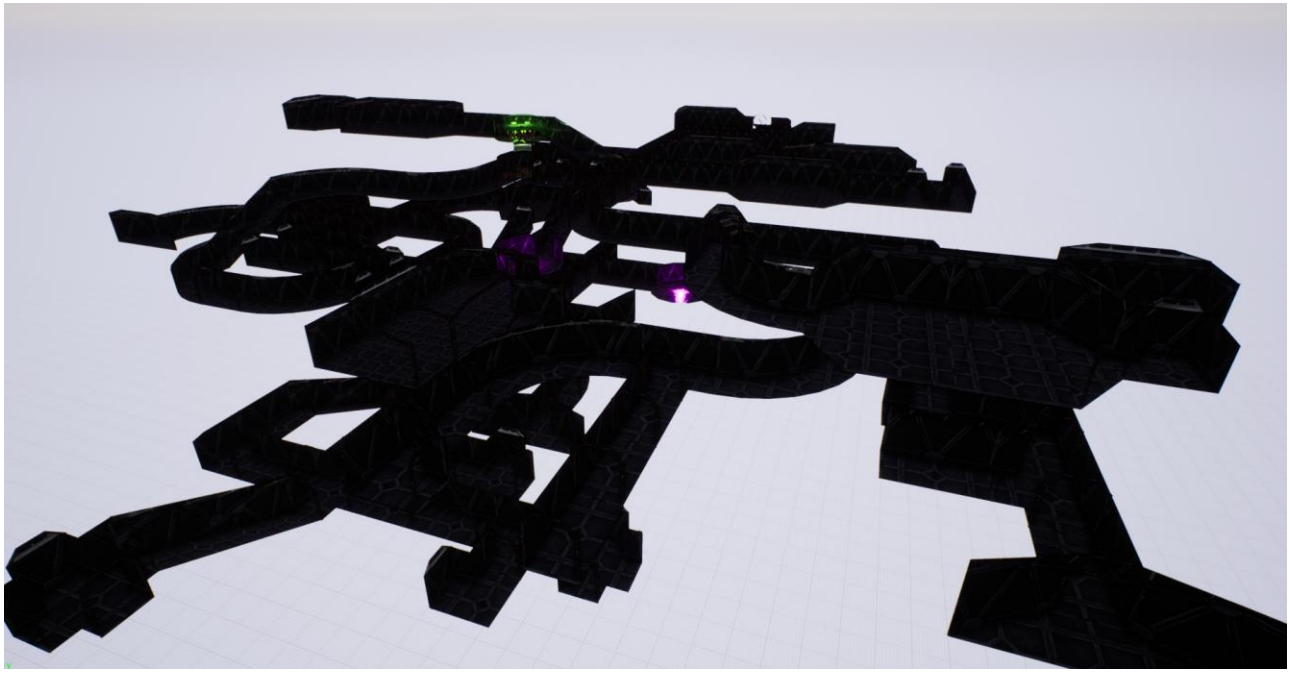

Рисунок 3.26 – Результат алгоритму генерації карти

Після опрацювання отриманого результату і порівняння його з ймовірним результатом алгоритму дерева квадратів, я дійшов висновку, що у моєї схеми є 1 важливий плюс і 1 мінус. Ця модель може працювати не тільки з кубічними деталями, щоб дерево квадратів працювало не з квадратами а з, наприклад, шестигранними кімнатами потрібно було-б зробити ще складніший алгоритм. Мій алгоритм може працювати з будь якими конструкціями, не залежно від їх структури і це є плюсом. Мінусом же можна назвати відсутність будь-якої структуризації, єдиний контроль над генерацією, який ми можемо

задати, це максимальна кількість контрактних елементів та краї карти. Самі карти є хаотичним і в їх будові не можна спостерігати жодної логіки.

Все-таки я вирішив залишити цю модель і не заміняти її на модель з використанням дерева квадратів. Із інших особливостей системи, можна виділити високу продуктивність завдяки налаштованій ієрархії з instanced static mesh. Ще можна виділити наступні недолік: генерація рівня займає достатньо довгий час.

3.4.3 Тестовий рівень

Ще я добавив демонстарційний рівень, на який можна потрапити через головне меню. На цьому рівні знаходяться всі інтерактивні об'єкти, всі вороги, демонстрація пошуку шляху неігрових персонажів, демонстрація ефектів освітлення а також кімната за допомогою якої можна перевірети ефективність відстрибуючих від поверхні снарядів (див. рис. 3.27).

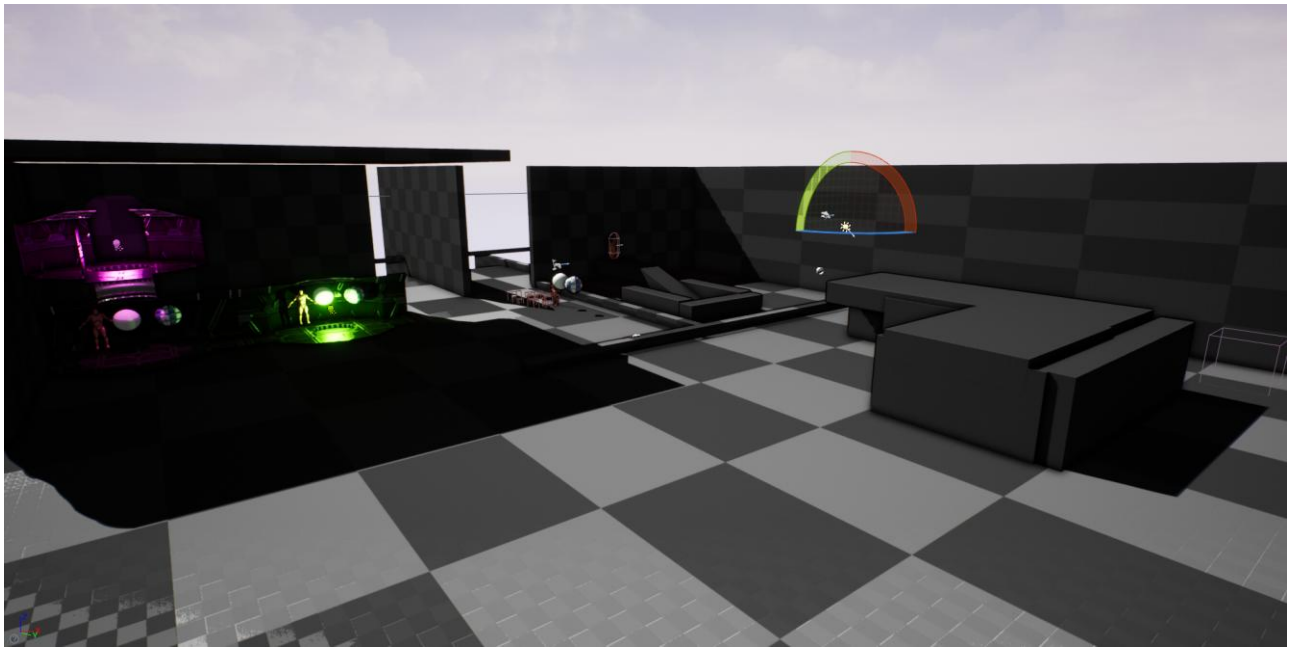

Рисунок 3.27 – Скріншот з тестового рівня

4 МЕТОДИ РОЗПОВСЮДЖЕННЯ, ЕКОНОМІЧНЕ ОБГРУНТУВАННЯ

4.1 Собівартість розробки проекту

При підрахунку ціни розробки я буду починати саме з оплати за роботу. Середня зарплата junior developer в “Unreal” в США починається з 70,000$ на рік (оскільки інформації про ринок в Україні я не зміг знайти, то мені прийдеться опиратись на іноземний ринок). Опираючись на інформацію, що середня річна зарплат зарплата junior developer в “JavaScript” в США становить 60,000$, а середня місячна зарплата на цю саму посаду в Україні становить 650$, то за наступною формулою можна підрахувати приблизну місячну вартість junior developer в “Unreal” в Україні(див. формулу 4.1),

$$U_{UaM} = \frac{JS_{UaM}}{JS_{UsaY}} * U_{UsaY,} \quad (4.1)$$

де U_{UaM}– це місячна зарплата розробника junior developer в “Unreal” в Україні
U_{UsaY} – це річна зарплата розробника junior developer в “ Unreal ” в США
JS_{UaM} – це місячна зарплата розробника junior developer в “JavaScript” в Україні
JS_{UsaY} – це річна зарплата розробника junior developer в “JavaScript” в США

Таким чином, було підраховано що плата junior розробнику в “Unreal” на території України буде коштувати близько 758$ на місяць. Враховуючи, що на початку я міг не тратити на проект взагалі жодного часу а під кінець я мг витрачати аж на 6 годин на день, я побудував лінійну функцію, через яку можна порахувати скільки я приблизно працював над проектом кожен день[29] (див. формулу 4.2),

$$Hours = Day * \frac{3}{100}, \quad (4.2)$$

де $Hours$ – це скільки часу я потратив в день

Day – це день в який я тратив цей час

Скориставшись наступною формулою я підрахував скільки саме часу я потратив(див. формулу 4.3),

$$Hours = \sum_{i=1}^{180} i * \frac{3}{100} = 488.7. \quad (4.3)$$

Беручи до уваги, що нормальна тривалість робочого часу на тиждень за Кодексом законів про працю України становить 40 годин, було підраховано що відробив 47.5% від норми. Також, враховуючи що роботу над проектом було почато в жовтні 2019, і закінчено в березні, то можна зробити висновок то фактично було потрачено 4548$, а з урахуванням проценту відробленої норми, то 2160.86$

Враховуючи, що в середньому мій комп'ютер споживав 180 ват на годину в звичайному режимі 3.20 грн і 440 ват в режимі навантаження (оскільки "Unreal" є дуже ресурсозатратним). В середньому, половина робочого часу тратилась в звичайному режимі і половина в навантаженому, також у мене завди працювало два монітора, кожен з яких приблизно споживає 80 ват на годину[30]. Далі наведено формулу, яка підраховує кількість витрачених грошей на електроенергію, беручи до уваги, що 1 кіловат вартує 3.20 гривень(див. формулу 4.4),

$$\left(\frac{420*488.7}{2} + \frac{180*488.7}{2} + 488.7 * 80 * 2\right) * 0.00320 = 719.3664 \text{ грн.} \quad (4.4)$$

Загалом, якщо брати до уваги факт що курс долара становить 27.10 грн, то на проект було потрачено 13963.14 грн. Також, деякі з використаних ресурсів "Unreal Marketplace" були платними, проте я за них не платив завдяки щомісячній роздачі платних комплектів за безплатно.

4.2 Способи комерціалізації

Способів комерціалізації ігрових продуків є не так багато, окрім представлених нижче, також можна намагатись самостійно просувати та продавати гру, хоча це буде складно і дорого.

Одним із перших способів що спав мені на думку став саме цей. Його суть полягає не в тому, щоб продавати гру, а в тому щоб продавати елементи з яких вона побудована. В “Unreal Marketplace” можна знайти елементи, з дуже широкою класифікацією, а саме там можна знайти: двовимірні елементи, анімацію, архітектурні моделі, готовий код, персонажів, плагіни, середовища, матеріали, музику, різні об’єкти, текстури, візульні ефекти, зброю тощо. Ми, звісно, будемо продавати тільки ті елементи, які и розробили, а саме я можу продавати систему озброєння, генератор карт та шейдери. Якщо шукати аналоги генератора, то в середньому генератори карт продається за 1500 гривень. Шедери схожі на мої продаються в діапазоні від 250 гривень до кількох тисяч (в залежності від кількості функцій). Системи озброєнь продаються в середньому по 900 гривень. Складно визначити ціну мої елементів, оскільки інші розробники явно мають більше знать про рушій і у них товар міг набагато кращий, кат що я хоч і можу продавати свій по мінімальній ціні, але робити цього скоріше за все не буду.

Наступним способом комерціалізації є “Steam”. Ігри добавляються в цей дистрибутив через “Steamworks”. Для того щоб почати продавати гру в необхідно заповнити потрібну інформацію, таку як адрес, місто, країна, електронна пошта та інші. Також потрібно вказати назву компанії (юридичної особи) яка є власником гри, оскільки “Steam” не публікує ігри від імені фізичних осіб не з громадянством США. Також має бути надана вся інформація для оподаткування і відбутись податкове опитування. Також потрібно мати Ідентифікаційні номери платників податків США (TIN).

Щоб добавить свій продукт в дистрибутив, необхідно заплатити податок 100 доларів США, також потрібно сплачувати податок на прибуток США. Ще “Steam” буде привласнювати 30% від доходу. Також потрібно сплачувати VAT за кожну продану гру, її розмір залежить від країни в якій відбулась транзакція(наприклад 5% в Арабський еміратах і 25 в Данії). Також варто враховувати що існує регіональна ціна, гра що коштує 20 доларів в США може коштувати в Україні 360 гривень.

Ще є “Epic games store”, але фізичним особам туди потрапити ще складніше ніж в “Steam” і там жорсткіший контроль якості. Також у них більш вигідніше оподаткування продуктів (12% а не 30%), але там менше покупців.

4.3 Краудфандингова компанія і подальший розвиток проекту

Краудфандинг – це громадське фінансування, в якому вкладники надають гроші на розвиток проекту в якому вони зацікавлені.

Мій продукт може не досягати потрібного рівня, якщо порівнювати з іншими платними іграми, проте я все-ще можу виставити свою гру на одну з краудфандингових мереж і зібрати кошти на розробку повноцінної гри. Така практика часто використовується в нинішньому відео-ігровому ринку.

Розробка вдалої краудфандигової компанії полягає в наступних прийомах: потрібно вдало представити свій продукт використовуючи різні відео та зображення, потрібно розкрити особливості мого проекту та історію за його появленням, треба розробити технологічну дорожню карту, яка буде співставляти прогрес розробки з кількістю залучених грошей і також можна розробити особливі винагороди для вкладників, що сильно виділяються.

Для розробки потрібно буде заснувати компанію (що я не можу зробити без вищої освіти), зібрати команду спеціалістів, а також потрібно можна

найняти менеджера по соціальних мережах, який буде вести бесіду від імені юридичної особи (це дуже часта практика в індустрії).

ВИСНОВКИ

Під час виконання дипломної роботи, було розроблено без бюджетну інді-гру, основними особливостями якої є генерація рівнів, цел шейдинг та механіка стрільби. Було розглянуто особливості рушія “Unreal”, його аналог візуальної мови програмування “Bluprints” і його актуальність в нинішній ситуації на ринку. Було отримано навички для роботи з рушієм і зроблено висновки про гострі питання в процесі розробки.

Сам рушій зародився в кінці 90-х років і його нинішні ітерації часто застосовуються в індустрії. Він розвивався з іншими рушіями і йшов в ногу з розвитком процесуальних можливостей комп’ютерів та в деяких випадках вводив революційні інновації в сферу розробки відеоігор та кінематографії.

Даний рушій є, на мою думку, одним із самих кращих (якщо не найкращих) виборів для розробки інді-ігор, в любому випадку у цій сфері вибір не великий: окрім нашого варіанта є тільки “GoDot”, “Unity” та “CryEngine”. Якщо розглядати рушій з точки зору ААА ігор, то тут питання трохи складніше розглянути, оскільки очевидно що у мене немає досвіду в розробці ігор такого ґатунку, проте беручи до уваги свої знання можу сказати що у “Unreal” є досить серйозні конкуренти: є такі гіганти як “Frostbite”, “id Tech”, і вже відносно скоро стане доступною нова ітерація “Source”, яка може бути беслптною. У мене є сумніви стосовно дружелюбності ігор на рушії до користувацьких модифікацій: по замовчуванню для гравців не доступний інструментарій для розробки доповнень, проте думаю його розробка є можливою і цю проблему можна вирішити.

Що до оцінки візуальної мови програмування “Bluprints”: технічно він може виглядати біль читабельним і при правильному використанні розміщеня коду в двух площинах (а не в одній як з класичним кодом, в якому по факту є тільки рядки). Також цей код є дружнім до нових користувачів, оскільки інтерфейс може пояснювати як працюють блоки і ми бачимо послідовність в

якій програма виконується. Візуально можна зразу замітити схожість з “Simulink” із-за схожості принципу роботи.

Із мінусів можна визначити що по суті як правило великий код “Bluprints” з безліччю з’єднань є менш читабельним, для його форматування потрібно тратити набагато більше тілесних рухів і вам також потрібно багато переміщатись по полю щоб за ним слідкувати, ще й до того, такий код без форматування, відносно класичного програмування, може виглядати дуже нечитабельним і об’єктивно не красивим. Також варто виділити обмежений функціонал цієї мови, з самого першого що кидається на очі, код є дуже громістким, особливо це видно на математичних перетвореннях, в яких кожна арифметична дія це окремий блок, в простих формулах це не є так вже і критично і навіть досить зручно, проте при більш комплексних обчисленнях тратиться багато площини поля редактора на дії, ще й до того зв’язки починають між собою переплітатись і створювати менш читабельний код. Проте в класичних мовах, виклик функції з великою кількістю параметрів може виглядати набагато заплутаніше.

Функціональних недоліків на низькому рівні як таких мало: ті чи інші дії, які доступні на класичних мовах програмування, можна виконати в “Bluprints” але по інакшому, можна виділили відсутність багатовимірних масивів та дата-класів, проте першу проблему можна обійти через остачу від ділення, а другу через використання акторів без візуалізації.

Значним недоліком можна виділити оптимізацію такого коду: він є досить стабільним проте банальне копіювання коду відбувається не моментально і копіювання тільки одного блоку може зайняти цілу секунду (хоча при копіюванні більшої кількості блоків, час не зростає в арифметичній або геометричній прогресії).

Моїм фінальним вердиктом в оцінці “Bluprints” є наступним: попри всі мінуси, ця мова все-таки має свої позитивні сторони і її краще використовувати в тандемі з “C++”, де перший буде виконувати дії на високому рівні, а другий

на низькому. Також є функція конвертації “Bluprints” в “C++”, але вона працює не дуже адекватно.

Після фінального тестування і налагодження помилок роботи програми було зібрано всі доступні дані. В грі можна бачити оригінальну задумку, проте якщо розбирати проект як відео-ігровий продукт, то він є досить примітивним: ігрова механіка є, але вона не збалансована і не є сильно унікальною, сюжету немає в принципі, ігрові рівні дуже прості а варіативність зброї і неігрових персонажів дуже низька. Проте я вважаю що розроблений мною алгоритм генерації рівнів є інтересним рішенням проблеми процедурної генерації. Також, реалізація cell-shading та Outline Враховуючи що у мене не було ні бюджету, ні побічних стимулів для роботи (окрім необхідності розробки проекту для здачі диплому), я вважаю що обсяг виконаної роботи, з врахуванням загальної громісткості проекту, є задовільним.

В грі є 3 противники, 3 види озброєння (з яких 2 мають альтернативний режим), генерація карти, стилізовані рейдери, повний користувацький інтерфейс та близько 15 інтерактивних об'єктів. Також було продумано баланс використання зброї і поповнення життя через ліквідацію ворогів. Гру можна продавати через ігрові дестрибутиви, а проект та його рішення можна продавати в “Unreal Marketplace”.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

REFERENCES

1. Документація до “Unreal engine 2”: веб ресурс. URL: https://docs.unrealengine.com/udk/Two/WebHome.html (дата звернення: 15.9.2019).
2. Документація Unreal engine: технічна документація. URL: https://docs.unrealengine.com/en-US/ (дата звернення: 1.10.2019).
3. Портфоліо 3Д аніматора Баліцького Олега: відеоролик. URL: https://www.youtube.com/watch?v=50a8KC98D4c (дата звернення: 13.1.2020).
4. Список ігор з використанням технології Cell Shading: відеоролик. URL: https://en.wikipedia.org/wiki/List_of_cel-shaded_video_games (дата звернення: 1.1.2020).
5. Стаття про обробку зображень: наукова робота. URL: http://aircconline.com/sipij/V4N3/4313sipij06.pdf (дата звернення: 20.1.2020).
6. Стаття про “Unreal Engine”: Стаття на Вікіпедії. URL: https://en.wikipedia.org/wiki/Unreal_Engine (дата звернення: 20.1.2020).
7. Четверта стіна: Стаття на Вікіпедії. URL: https://uk.wikipedia.org/wiki/четверта_стіна (дата звернення: 2.2.2020).
8. Шейдер: Стаття на Вікіпедії. URL: https://uk.wikipedia.org/wiki/шейдер (дата звернення: 26.12.2019).
9. Animation Starter Pack Unreal engine Marketplace: 3Д анімація. URL: https://www.unrealengine.com/marketplace/en-US/product/animation-starter-pack (дата звернення: 15.2.2020).

10. A GameTextures.com 2019 FREE Material Pack: текстури. URL: https://www.unrealengine.com/marketplace/en-US/product/phoenyx-anim-pack-02 (дата звернення: 26.3.2020).
11. Beretta 9mm free3d: 3Д модель. URL: https://free3d.com/3d-model/beretta-9mm-14366.html (дата звернення: 10.11.2019).
12. Blueprint Generating Procedural URL: https://youtu.be/mI7eYXMJ5eI (дата звернення: 17.11.2019).
13. Franchi-SPAS 12: 3Д модель. URL:https://sketchfab.com/3d-models/franchi-spas-12-3375e411d98f43639c753494557666af (дата звернення: 20.11.2019).
14. Gears of War: Сайт продукту. URL: https://gearsofwar.com/ (дата звернення: 15.11.2019).
15. Grunt Soldier" Unreal engine Marketplace: 3Д модель. URL: https://www.unrealengine.com/marketplace/en-US/product/grunt-soldier (дата звернення: 26.11.2019).
16. How Many Guns are in Borderlands 2? відеоролик. URL: https://www.youtube.com/watch?v=yueSPkzBa1M (дата звернення: 18.11.2019).
17. How The Wind Waker Defined Cel Shading: відеоролик. URL: https://www.youtube.com/watch?v=mnxs6CR6Zrk (дата звернення: 17.1.2020).
18. Infinity Blade: Effects Unreal engine Marketplace: 3Д модель. URL: https://www.unrealengine.com/marketplace/en-US/product/infinity-blade-effects (дата звернення: 1.2.2020).
19. Source Filmmaker Сайт продукту. URL: https://store.steampowered.com/app/1840/Source_Filmmaker/ (дата звернення: 16.9.2019).

20. The Legend of Zelda: The Wind Waker: Сайт продукту. URL: https://www.nintendo.com/games/detail/the-legend-of-zelda-the-wind-waker-hd-wii-u/ (дата звернення: 13.1.2020).
21. UAC Plasma Rifle Sketchfab: 3Д модель. URL: https://sketchfab.com/3d-models/uac-plasma-rifle-3cd9bb5eeda54a10bb13d21015b29723 (дата звернення: 4.2.2020).
22. Unreal Engine: Сайт продукту. URL: https://www.unrealengine.com/en-US/ (дата звернення: 1.9.2019).
23. Unreal Gold: Сайт продукту. URL: https://store.steampowered.com/app/13250/Unreal_Gold/ (дата звернення: 11.9.2019).
24. Unreal Tournament 2004: Сайт продукту. URL: https://store.steampowered.com/app/13230/Unreal_Tournament_2004_Editors_Choice_Edition/ (дата звернення: 15.11.2019).
25. Unreal is free to celebrate its 20th birthday: стаття. URL: https://www.rockpapershotgun.com/2018/05/22/unreal-free-for-20th-birthday/ (дата звернення: 12.10.2019).
26. Cyberprisoner Unreal engine Marketplace: 3Д модель. URL: https://unrealengine.com/marketplace/en-US/product/cyberprisoner (дата звернення: 25.11.2019).
27. Fighters Pack Unreal engine Marketplace: 3Д модель. URL: https://unrealengine.com/marketplace/en-US/product/fighters-pack (дата звернення: 17.3.2020).
28. Quadtree. Стаття на Вікіпедії. URL: https://en.wikipedia.org/wiki/Quadtree(дата звернення: 1.2.2020).
29. Kuz M., Kozlenko M., Lazarovych I., Rysniuk O., Novak V., Novak M. Method of weights determination based on ratings of software quality metrics. Applied scientific and technical research: Proceedings of the IV International

Scientific and Practical Conference, Ivano-Frankivsk, April 1-3, 2020, Ivano-Frankivsk, V.2. 2020. P. 37-39.

30. Kuz M., Shevliuk L., Ostafiichuk T., Mishagin R. Modeling of software time characteristics. Applied scientific and technical research: Proceedings of the IV International Scientific and Practical Conference, Ivano-Frankivsk, April 1-3, 2020, Ivano-Frankivsk, V.2. 2020. P. 39-42.
31. M. Kozlenko, A. Bosyi, O. Simkiv, and N. Savchenko, "Artificial intelligence in e-commerce," in Proc. 3rd International Conference "Applied Information Systems and Technologies in the Information Society" (AISTIS), V. Pleskach and V. Mironova, Eds. Taras Shevchenko National University of Kyiv, Kyiv, Ukraine, Sep. 30, 2019, pp. 207-211.

ДОДАТОК А

Специфікація вимог до “Epic PNU game” на “Unreal Engine”

(рекомендований)

СПЕЦИФІКАЦІЯ ВИМОГ ДО “EPIC PNU GAME” НА “UNREAL ENGINE”

За авторством Баліцького Олега (ІПЗ-41)

А.1 ВВЕДЕННЯ

А.1.1 Призначення

Цей документ призначений для полегшення розробки продукту. Тут структуровано всю необхідну інформацію, починаючи від системних вимог і закінчуючи вимогами до дизайну.

А.1.2 Угоди, прийняті в документах

При оформленні документації було звернено увагу на вимоги до оформлення дипломного проекту за 2018 рік. При розробці продукту було прийнято End User License Agreement рушія “Unreal” та умови ліцензії на програмне забезпечення “Microsoft”

А.1.2 Передбачена аудиторія

Передбачається що цією документацію будуть користуватись програмні інженери з ймовірною неповною вищою освітою та рівнем знань

програмування Junior або Middle. В цьому документі буде описано поведінку розробки системи і всю іншу необхідну інформацію.

А.1.3 Границі проекту

Границями проекту вважається вважаться сфера застосування продукту в прямому призначенні. Продукт можна розповсюджувати за гроші через ігрові дистрибутиви Любі інші способи використання не передбачені. Продукт не належить до ніякого підприємства і є власністю фізичної особи. Основною ціллю проекту (окрім розвитку умінь) є отримання будь-яких корисних надбань.

А.1.4 Посилання

Нижче було вказано посилання на інформацію на яку посилається цей документі, та на яку потрібно опиратись при розробці проекту:

1 Документація Unreal engine: технічна документація. URL: https://docs.unrealengine.com/en-US/
2 End User License Agreement рушія “Unreal”: угода використання рушія. URL: unrealengine.com/en-US/eula/bullet-train
3 Умови ліцензії на програмне забезпечення “Microsoft”. URL: https://visualstudio.microsoft.com/ru/license-terms/mlt031819/

А.2 ЗАГАЛЬНИЙ ОПИС

А.2.1 Загальний погляд на продукт

Продукт застосовується в розважальних цілях. Продукт являє собою відеогру жанру “шутер від першого лиця”. Розробка проекту в свою чергу є засобом для підняття навичок розробника та способом досягнення різних корисних цілей. Продукт не має бути версією іншого продукту.

Передбачено аудиторією продукту є люди переважно віком від 15 до 35, серед яких 65% є чоловіки.

А.2.2 Особливості продукту

Головними особливостями продукту мають стати динамічний геймплей, баланс ігрових механік та обмежений але стилізований візуальний стиль. Потоки даних мають виглядати наступним чином (див. рисунок 2.1):

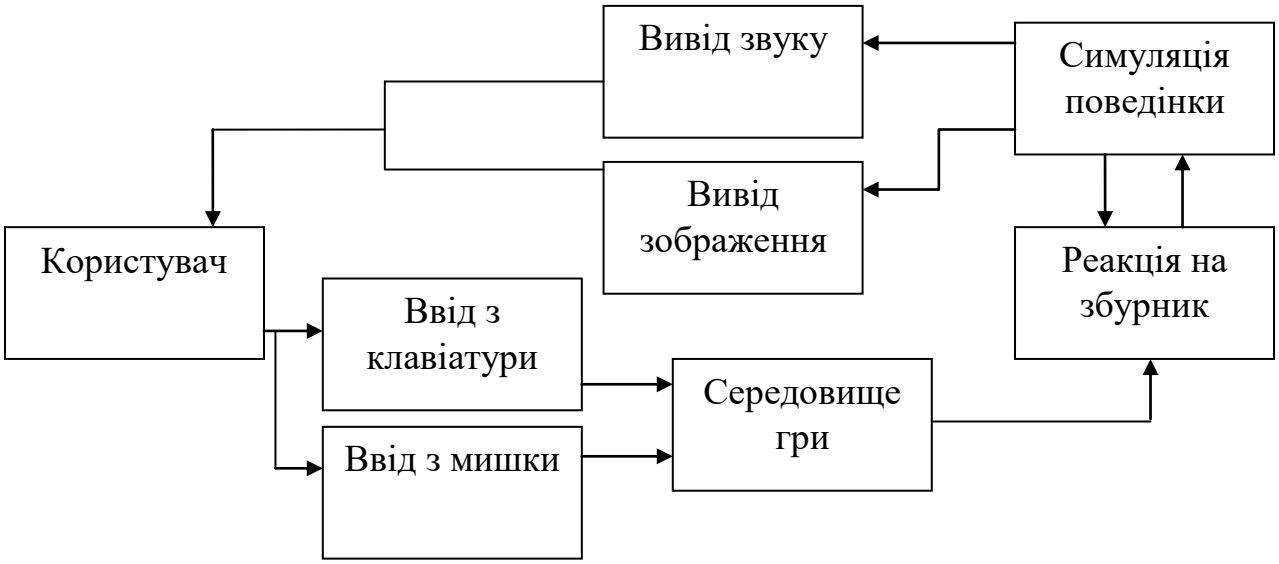

Рисунок А.1 – потоки даних проекту

А.2.3 Класи і характеристики користувачів

Продукт не передбачає різних рівнів користувачів: всі користувачі рівні між собою. Більше повноважень знаходиться не у власників програмного забезпечення а у його розробників, оскільки вони мають доступ до коду продукту і до всіх інших його частин.

А.2.4 Операційне середовище

При розробці проекту, рекомендується процесор з 4-ма ядрами, 16 гігабайтів оперативної пам’яті, мінімум 70 гігабайтів льног омісця на жорсткому диску та сучасну не інтегровану відео-карту. Рекомендується використання Windows 10, проте підійде будь яка ніша операційна система на 64-біній базі. Мінімальними вимогами для гравців є відео-карта Nvidia GTX-1050 або інші аналоги, процесор Intel I3-7300 або інші аналоги, 8Gb RAM та 6 гігабайтів вільного місця на жорсткому диску.

А.2.5 Обмеження дизайну і реалізації

На розробиках лежать обмеження в вигляді необхідності дотримуватись маркам вимог продуктивності, розробники обмежені процесуальними властивостями комп’ютерів користувачів, за умов якщо ті відповідають програмним вимогам. Розробники обмежені в розробці системи збору даних про користувачів.

Розробники обмежені жанром розробленої гри. Жодниз інших обмежень немає.

А.2.6 Документація для користувачів

Документації для користувачів в комплекті з продуктом не іде. Пояснюється це тим, що продукт є розважальним і його інтерфейс є достатньо дружнім до користувача. Будь яка необхідна інформація вбудована в інтерфейс програми. Деякі данні в продукті не описані, оскільки вони належать до загально розповсюджених знань.

А.2.7 Припущення і залежності

Припускається використання інших мов програмування окрім “C++”. Можливість перегляду коду залежить від наявності потрібного програмного забезпечення а саме “Unreal engine”

А.3 ФУНКЦІЇ СИСТЕМИ

А.3.1 Опис і пріоритети

Даний продукт має представляти собою гру в жані “Шутер від першого лиця” з усіма витікаючими особливостями. Крім цього, в розпорядженні гравця має бути доступно як мінімум 2 типи озброєння та можливість поповнювати боєкомплект та здоровя. Мають бути неігрові персонажі - вороги, які взаємодіють з гравцем. Пріорітетними задачами є розробка шейдерів та реалізація збалансованої гейплейної задумки.

А.3.2 Послідовності «вплив-реакція»

В програмному забезпеченні мають бути наступні взаємозв’язки. Продукт має правильно реагувати на введення користувача. Налаштування гри має правильно вносити зміну до роботи коду. При використанні зброх гравця має виконуватись унікальна для кожної зброї дія, яка тим чи іншим методом впливає на неігрових персонажів, які в свою чергу взаємодіють з гравцем шляхом пониження його рівня життя.

А.3.3 Функціональні вимоги

Функціональні вимоги будуть вважатись виконаними, в тому випадку, якщо у гравця є можливість міняти налаштування гри, генерувати карту, протидіяти неігровим персонажам та відновлювати свій боєкомплект та життя.

А.4 ВИМОГИ ДО ЗВОНІШНІХ ІНТЕРФЕЙСІВ

А.4.1 Інтерфейси користувача

Інтерфейс має наслідувати основні поняття інтерфейсу ігор такого жанру: гравець має дізнаватись звідкись про свій рівень життя та боєкомплект, також гравець має знати коли ворог наносить йому пошкодження і навпаки. Крім цього має бути стандартне меню, через яке, гравець має переключатись між основними функціями гри. Весь інший інтерфейс, представляє собою периферійне обладнання комп’ютера. Запуск продукту має здійснюватись через ярлик exe файлу.

А.4.2 Програмні інтерфейси

Жоден програмний інтерфейс не передбачений, оскільки ніхто крім виробника не має доступу до коду. Присутня консоль розробника в іграх на вбраному рушії не потребує будь-яких пояснень із-за спільних команд.

А.4.3 Інтерфейси обладнання

Має бути присутня можливість використання клавіатури та миші. Вивід зображення має відбуватись через монітор, а вивід звуку через пристрої звуку.

А.4.4 Інтерфейси зв’язку і комунікації

Ніякі мережеві методи зв’язку не передбачені, оскільки гра відбувається в офлайновому режимі. Комунікація між розробником та споживачем допускається через соціальні мережі. Передача коду і готового продукту допускається через вільного розміщення на Google disc.

А.4.5 Нефункціональні вимоги

Вимог що-до безпеки та конфіденційності з точки зору готовго продукту майже нема. Єдине вимога – це недопущення по сторонніх осіб до коду продукту. Передбачуються несанкціоновані методи доступу до деяких елементів гри (а саме текстур, звуків та 3д моделей) шляхом реверсивної інженерії в зв’язку з особливостями рушія, але це не критично.

Основною вимогою до надійності продукту є відсутність критичних помилок, які заважають користувачу грати в гру або створюють графічні артефакти. Основною вимогою до відновлюваності є реалізаціє повної функції паузи гри. Перевірка продукту здійснюється через безпосередню гру і через тестову карту.

А.5 НЕФУНКЦІОНАЛЬНІ ВИМОГИ

А.5.1 Вимоги до продуктивності

Основними вимогами до продуктивності є стабільна робота гри в 60 фпс за умови дотримань оптимальних конфігурацій комп’ютера.

Також рекомендується опиратись на дане джерело інформації: https://uk.wikipedia.org/wiki/Тестування_продуктивності

А.5.2 Вимоги збереженості даних

Основною вимогою до збереження даних це збереження налаштувань.

А.5.3 Критерії якості програмного забезпечення

При оцінці якості програмного забезпечення потрібно користуватись даними з наступного посилання: https://uk.wikipedia.org/wiki/Якість_програмного_забезпечення

А.5.4 Вимоги до безпеки системи

Особливих вимог до безпеки готового продукту нема. Основні вимоги до безпеки можна знайти тут: https://uk.wikipedia.org/wiki/Безпека_прикладних_програм

ДОДАТОК Б

Представлення візуального коду

(рекомендований)

1. Механіка Mouse sway. URL: https://blueprintue.com/blueprint/4w379te4/
2. Механіка стрибка. URL: https://blueprintue.com/blueprint/z-xejyc_/
3. Механіка стрільби. URL: https://blueprintue.com/blueprint/bm0qbdzy/
4. Механіка альтернативної стрільби. URL: https://blueprintue.com/blueprint/dc4p3arb/
5. Механіка перезарядки. URL: https://blueprintue.com/blueprint/iux4-ju4/
6. Механіка запуску інтерфейсу. URL: https://blueprintue.com/blueprint/bnicapkj/
7. Механіка ближнього бою неігрового персонажа. URL: https://blueprintue.com/blueprint/8yk2xzd7/
8. Основна механіка одного з неігрових персонажів. URL: https://blueprintue.com/blueprint/dx_bu4so/
9. Виклик функцій генератора карти. URL: https://blueprintue.com/blueprint/6kh2uzw0/
10. Механіка інтерфейсу https://blueprintue.com/blueprint/9dpb4_1n/

ДОДАТОК В

Програмний код продукту (механіка стрибка)

(рекомендований)

```
Begin Object Class=/Script/BlueprintGraph.K2Node_InputAction
Name="K2Node_InputAction_60"
   InputActionName="Jump"
   bOverrideParentBinding=False
   NodePosX=-20832
   NodePosY=-848
   NodeGuid=85B233A843C15F3ACC54228E364AB796
   CustomProperties Pin
(PinId=48D2032843DFE6DE228F03BDF5329C9F,PinName="Pressed",Directio
n="EGPD_Output",PinType.PinCategory="exec",PinType.PinSubCategory=
"",PinType.PinSubCategoryObject=None,PinType.PinSubCategoryMemberR
eference=(),PinType.PinValueType=(),PinType.ContainerType=None,Pin
Type.bIsReference=False,PinType.bIsConst=False,PinType.bIsWeakPoin
ter=False,LinkedTo=(K2Node_IfThenElse_3
99932CD24052765193C000A28B3C5607,),PersistentGuid=0000000000000000
0000000000000000,bHidden=False,bNotConnectable=False,bDefaultValue
IsReadOnly=False,bDefaultValueIsIgnored=False,bAdvancedView=False,
bOrphanedPin=False,)
   CustomProperties Pin
(PinId=E923AC324D959037CB7E5D891F543940,PinName="Released",Directi
on="EGPD_Output",PinType.PinCategory="exec",PinType.PinSubCategory
="",PinType.PinSubCategoryObject=None,PinType.PinSubCategoryMember
Reference=(),PinType.PinValueType=(),PinType.ContainerType=None,Pi
nType.bIsReference=False,PinType.bIsConst=False,PinType.bIsWeakPoi
nter=False,LinkedTo=(K2Node_CallFunction_9464
28E34A524B8051E782E333AE340F0F65,),PersistentGuid=0000000000000000
0000000000000000,bHidden=False,bNotConnectable=False,bDefaultValue
IsReadOnly=False,bDefaultValueIsIgnored=False,bAdvancedView=False,
bOrphanedPin=False,)
```

Продовження додатку В

```
   CustomProperties Pin
(PinId=FECDFF44412356652A8CD499F5469CA0,PinName="Key",Direction="E
GPD_Output",PinType.PinCategory="struct",PinType.PinSubCategory=""
,PinType.PinSubCategoryObject=ScriptStruct'"/Script/InputCore.Key"
',PinType.PinSubCategoryMemberReference=(),PinType.PinValueType=()
,PinType.ContainerType=None,PinType.bIsReference=False,PinType.bIs
Const=False,PinType.bIsWeakPointer=False,DefaultValue="None",Autog
eneratedDefaultValue="None",PersistentGuid=00000000000000000000000
000000000,bHidden=False,bNotConnectable=False,bDefaultValueIsReadO
nly=False,bDefaultValueIsIgnored=False,bAdvancedView=False,bOrphan
edPin=False,)
End Object
Begin Object Class=/Script/UnrealEd.EdGraphNode_Comment
Name="EdGraphNode_Comment_10"
   NodePosX=-20928
   NodePosY=-960
   NodeWidth=2096
   NodeHeight=720
   NodeComment="Jump"
   NodeGuid=B5B1DB0D407B8CFA1BFC82A5E207C9F5
End Object
Begin Object Class=/Script/BlueprintGraph.K2Node_CallFunction
Name="K2Node_CallFunction_9464"
   FunctionReference=(MemberName="StopJumping",bSelfContext=True)
   NodePosX=-20832
   NodePosY=-720
   NodeGuid=87A062894FD2D141EDA0A6924E2F6EF2
   CustomProperties Pin
(PinId=28E34A524B8051E782E333AE340F0F65,PinName="execute",PinToolT
ip="\nExec",PinType.PinCategory="exec",PinType.PinSubCategory="",P
inType.PinSubCategoryObject=None,PinType.PinSubCategoryMemberRefer
ence=(),PinType.PinValueType=(),PinType.ContainerType=None,PinType
.bIsReference=False,PinType.bIsConst=False,PinType.bIsWeakPointer=
```

```
False,LinkedTo=(K2Node_InputAction_60
E923AC324D959037CB7E5D891F543940,),PersistentGuid=0000000000000000
0000000000000000,bHidden=False,bNotConnectable=False,bDefaultValue
IsReadOnly=False,bDefaultValueIsIgnored=False,bAdvancedView=False,
bOrphanedPin=False,)
   CustomProperties Pin
(PinId=94DD177042C36EC780ECE58FF177CD47,PinName="then",PinToolTip=
"\nExec",Direction="EGPD_Output",PinType.PinCategory="exec",PinTyp
e.PinSubCategory="",PinType.PinSubCategoryObject=None,PinType.PinS
ubCategoryMemberReference=(),PinType.PinValueType=(),PinType.Conta
inerType=None,PinType.bIsReference=False,PinType.bIsConst=False,Pi
nType.bIsWeakPointer=False,PersistentGuid=0000000000000000000000000
00000000,bHidden=False,bNotConnectable=False,bDefaultValueIsReadOn
ly=False,bDefaultValueIsIgnored=False,bAdvancedView=False,bOrphane
dPin=False,)
   CustomProperties Pin
(PinId=E59C000141B335641FAD73B143C2DD33,PinName="self",PinFriendly
Name=NSLOCTEXT("K2Node", "Target",
"Target"),PinToolTip="Target\nCharacter Object
Reference",PinType.PinCategory="object",PinType.PinSubCategory="",
PinType.PinSubCategoryObject=Class'"/Script/Engine.Character"',Pin
Type.PinSubCategoryMemberReference=(),PinType.PinValueType=(),PinT
ype.ContainerType=None,PinType.bIsReference=False,PinType.bIsConst
=False,PinType.bIsWeakPointer=False,PersistentGuid=000000000000000
00000000000000000,bHidden=False,bNotConnectable=False,bDefaultValu
eIsReadOnly=False,bDefaultValueIsIgnored=False,bAdvancedView=False
,bOrphanedPin=False,)
End Object
Begin Object Class=/Script/BlueprintGraph.K2Node_CallFunction
Name="K2Node_CallFunction_48"

FunctionReference=(MemberName="LaunchCharacter",bSelfContext=True)
   NodePosX=-19088
```

Продовження додатку В

```
   NodePosY=-880
   NodeGuid=8065133548D6CB77255770A886A48EBF
   CustomProperties Pin
(PinId=8FEE1967465FAB8455E70A9546C401F9,PinName="execute",PinToolT
ip="\nExec",PinType.PinCategory="exec",PinType.PinSubCategory="",P
inType.PinSubCategoryObject=None,PinType.PinSubCategoryMemberRefer
ence=(),PinType.PinValueType=(),PinType.ContainerType=None,PinType
.bIsReference=False,PinType.bIsConst=False,PinType.bIsWeakPointer=
False,LinkedTo=(K2Node_IfThenElse_20
8D2804034864525FFABF66AC278AB431,),PersistentGuid=0000000000000000
0000000000000000,bHidden=False,bNotConnectable=False,bDefaultValue
IsReadOnly=False,bDefaultValueIsIgnored=False,bAdvancedView=False,
bOrphanedPin=False,)
   CustomProperties Pin
(PinId=BE6B05F840E6D047EE12D5924F080631,PinName="then",PinToolTip=
"\nExec",Direction="EGPD_Output",PinType.PinCategory="exec",PinTyp
e.PinSubCategory="",PinType.PinSubCategoryObject=None,PinType.PinS
ubCategoryMemberReference=(),PinType.PinValueType=(),PinType.Conta
inerType=None,PinType.bIsReference=False,PinType.bIsConst=False,Pi
nType.bIsWeakPointer=False,PersistentGuid=0000000000000000000000000
00000000,bHidden=False,bNotConnectable=False,bDefaultValueIsReadOn
ly=False,bDefaultValueIsIgnored=False,bAdvancedView=False,bOrphane
dPin=False,)
   CustomProperties Pin
(PinId=D28A5C094C9E2718ACEFB1BBA7CFDD33,PinName="self",PinFriendly
Name=NSLOCTEXT("K2Node", "Target",
"Target"),PinToolTip="Target\nCharacter Object
Reference",PinType.PinCategory="object",PinType.PinSubCategory="",
PinType.PinSubCategoryObject=Class'"/Script/Engine.Character"',Pin
Type.PinSubCategoryMemberReference=(),PinType.PinValueType=(),PinT
ype.ContainerType=None,PinType.bIsReference=False,PinType.bIsConst
=False,PinType.bIsWeakPointer=False,PersistentGuid=000000000000000
00000000000000000,bHidden=False,bNotConnectable=False,bDefaultValu
```

eIsReadOnly=False,bDefaultValueIsIgnored=False,bAdvancedView=False
,bOrphanedPin=False,)
   CustomProperties Pin
(PinId=464755E94D36D197DE62F0836D1E551C,PinName="LaunchVelocity",P
inToolTip="Launch Velocity\nVector\n\nis the velocity to impart to
the
Character",PinType.PinCategory="struct",PinType.PinSubCategory="",
PinType.PinSubCategoryObject=ScriptStruct'"/Script/CoreUObject.Vec
tor"',PinType.PinSubCategoryMemberReference=(),PinType.PinValueTyp
e=(),PinType.ContainerType=None,PinType.bIsReference=False,PinType
.bIsConst=False,PinType.bIsWeakPointer=False,DefaultValue="0, 0,
0",AutogeneratedDefaultValue="0, 0,
0",LinkedTo=(K2Node_CommutativeAssociativeBinaryOperator_14
FF1A7F0E4CB732A9293A1C9EF5D41A0F,),PersistentGuid=0000000000000000
0000000000000000,bHidden=False,bNotConnectable=False,bDefaultValue
IsReadOnly=False,bDefaultValueIsIgnored=False,bAdvancedView=False,
bOrphanedPin=False,)
   CustomProperties Pin
(PinId=C4A8F6DA40546C5B12309BA846F5E68F,PinName="bXYOverride",PinT
oolTip="XYOverride\nBoolean\n\nif true replace the XY part of the
Character\'s velocity instead of adding to
it.",PinType.PinCategory="bool",PinType.PinSubCategory="",PinType.
PinSubCategoryObject=None,PinType.PinSubCategoryMemberReference=()
,PinType.PinValueType=(),PinType.ContainerType=None,PinType.bIsRef
erence=False,PinType.bIsConst=False,PinType.bIsWeakPointer=False,D
efaultValue="false",AutogeneratedDefaultValue="false",PersistentGu
id=00000000000000000000000000000000,bHidden=False,bNotConnectable=
False,bDefaultValueIsReadOnly=False,bDefaultValueIsIgnored=False,b
AdvancedView=False,bOrphanedPin=False,)
   CustomProperties Pin
(PinId=FEC28E614CEF9809927BC2BF2C1BAA19,PinName="bZOverride",PinTo
olTip="ZOverride\nBoolean\n\nif true replace the Z component of
the Character\'s velocity instead of adding to

it.",PinType.PinCategory="bool",PinType.PinSubCategory="",PinType.
PinSubCategoryObject=None,PinType.PinSubCategoryMemberReference=()
,PinType.PinValueType=(),PinType.ContainerType=None,PinType.bIsRef
erence=False,PinType.bIsConst=False,PinType.bIsWeakPointer=False,D
efaultValue="false",AutogeneratedDefaultValue="false",PersistentGu
id=00000000000000000000000000000000,bHidden=False,bNotConnectable=
False,bDefaultValueIsReadOnly=False,bDefaultValueIsIgnored=False,b
AdvancedView=False,bOrphanedPin=False,)
End Object
Begin Object Class=/Script/BlueprintGraph.K2Node_IfThenElse
Name="K2Node_IfThenElse_3"
   NodePosX=-20448
   NodePosY=-848
   NodeGuid=06A6E98C4175951C607A1BB9B82B5B28
   CustomProperties Pin
(PinId=99932CD24052765193C000A28B3C5607,PinName="execute",PinType.
PinCategory="exec",PinType.PinSubCategory="",PinType.PinSubCategor
yObject=None,PinType.PinSubCategoryMemberReference=(),PinType.PinV
alueType=(),PinType.ContainerType=None,PinType.bIsReference=False,
PinType.bIsConst=False,PinType.bIsWeakPointer=False,LinkedTo=(K2No
de_InputAction_60
48D2032843DFE6DE228F03BDF5329C9F,),PersistentGuid=0000000000000000
0000000000000000,bHidden=False,bNotConnectable=False,bDefaultValue
IsReadOnly=False,bDefaultValueIsIgnored=False,bAdvancedView=False,
bOrphanedPin=False,)
   CustomProperties Pin
(PinId=B0AD42334AB9EA0F00E4E999D8269A66,PinName="Condition",PinTyp
e.PinCategory="bool",PinType.PinSubCategory="",PinType.PinSubCateg
oryObject=None,PinType.PinSubCategoryMemberReference=(),PinType.Pi
nValueType=(),PinType.ContainerType=None,PinType.bIsReference=Fals
e,PinType.bIsConst=False,PinType.bIsWeakPointer=False,DefaultValue
="true",AutogeneratedDefaultValue="true",LinkedTo=(K2Node_CallFunc
tion_46

```
B22C87A64F7750CB39CAC69C5FFCCBD5,),PersistentGuid=0000000000000000
0000000000000000,bHidden=False,bNotConnectable=False,bDefaultValue
IsReadOnly=False,bDefaultValueIsIgnored=False,bAdvancedView=False,
bOrphanedPin=False,)
   CustomProperties Pin
(PinId=08F5667E42BC272B30CC3084D310EF21,PinName="then",PinFriendly
Name=NSLOCTEXT("K2Node", "true",
"true"),Direction="EGPD_Output",PinType.PinCategory="exec",PinType
.PinSubCategory="",PinType.PinSubCategoryObject=None,PinType.PinSu
bCategoryMemberReference=(),PinType.PinValueType=(),PinType.Contai
nerType=None,PinType.bIsReference=False,PinType.bIsConst=False,Pin
Type.bIsWeakPointer=False,LinkedTo=(K2Node_VariableSet_14
1527D75E4B5045460E9C43A89B32E4AD,),PersistentGuid=0000000000000000
0000000000000000,bHidden=False,bNotConnectable=False,bDefaultValue
IsReadOnly=False,bDefaultValueIsIgnored=False,bAdvancedView=False,
bOrphanedPin=False,)
   CustomProperties Pin
(PinId=AEF9A40A4AC66A4DD8E4B38EEC05C53C,PinName="else",PinFriendly
Name=NSLOCTEXT("K2Node", "false",
"false"),Direction="EGPD_Output",PinType.PinCategory="exec",PinTyp
e.PinSubCategory="",PinType.PinSubCategoryObject=None,PinType.PinS
ubCategoryMemberReference=(),PinType.PinValueType=(),PinType.Conta
inerType=None,PinType.bIsReference=False,PinType.bIsConst=False,Pi
nType.bIsWeakPointer=False,PersistentGuid=0000000000000000000000000
00000000,bHidden=False,bNotConnectable=False,bDefaultValueIsReadOn
ly=False,bDefaultValueIsIgnored=False,bAdvancedView=False,bOrphane
dPin=False,)
End Object
Begin Object Class=/Script/BlueprintGraph.K2Node_VariableGet
Name="K2Node_VariableGet_0"

VariableReference=(MemberName="JumpsLeft",MemberGuid=6D41254346CB3
861C858AF86CDF84533,bSelfContext=True)
```

Продовження додатку В

```
   NodePosX=-20800
   NodePosY=-896
   NodeGuid=A7DF4BF04BCFBFF312D962943A1EB187
   CustomProperties Pin
(PinId=BE54B3924CB1F1E2EAC2958C64B90597,PinName="JumpsLeft",Direct
ion="EGPD_Output",PinType.PinCategory="int",PinType.PinSubCategory
="",PinType.PinSubCategoryObject=None,PinType.PinSubCategoryMember
Reference=(),PinType.PinValueType=(),PinType.ContainerType=None,Pi
nType.bIsReference=False,PinType.bIsConst=False,PinType.bIsWeakPoi
nter=False,DefaultValue="0",AutogeneratedDefaultValue="0",LinkedTo
=(K2Node_CallFunction_46
E8501313453700D79703D984756BC756,K2Node_CallFunction_43
65C9A3554C22978CE5321A8C4750DCEC,),PersistentGuid=0000000000000000
0000000000000000,bHidden=False,bNotConnectable=False,bDefaultValue
IsReadOnly=False,bDefaultValueIsIgnored=False,bAdvancedView=False,
bOrphanedPin=False,)
   CustomProperties Pin
(PinId=95214B9A4A4902F6B64B41B7152AFCBE,PinName="self",PinFriendly
Name=NSLOCTEXT("K2Node", "Target",
"Target"),PinType.PinCategory="object",PinType.PinSubCategory="",P
inType.PinSubCategoryObject=BlueprintGeneratedClass'"/Game/Classic
_Shooter/FirstPersonCharacter/BP_FirstPersonCharacter.BP_FirstPers
onCharacter_C"',PinType.PinSubCategoryMemberReference=(),PinType.P
inValueType=(),PinType.ContainerType=None,PinType.bIsReference=Fal
se,PinType.bIsConst=False,PinType.bIsWeakPointer=False,PersistentG
uid=00000000000000000000000000000000,bHidden=True,bNotConnectable=
False,bDefaultValueIsReadOnly=False,bDefaultValueIsIgnored=False,b
AdvancedView=False,bOrphanedPin=False,)
End Object
Begin Object Class=/Script/BlueprintGraph.K2Node_CallFunction
Name="K2Node_CallFunction_46"
   bIsPureFunc=True
```

```
FunctionReference=(MemberParent=Class'"/Script/Engine.KismetMathLi
brary"',MemberName="Greater_IntInt")
   NodePosX=-20624
   NodePosY=-784
   NodeGuid=CA95E6B34D3112E77C2BB497A7909228
   CustomProperties Pin
(PinId=4ECD7C334AAFC8230FA97EAC93FDC0F9,PinName="self",PinFriendly
Name=NSLOCTEXT("K2Node", "Target",
"Target"),PinToolTip="Target\nKismet Math Library Object
Reference",PinType.PinCategory="object",PinType.PinSubCategory="",
PinType.PinSubCategoryObject=Class'"/Script/Engine.KismetMathLibra
ry"',PinType.PinSubCategoryMemberReference=(),PinType.PinValueType
=(),PinType.ContainerType=None,PinType.bIsReference=False,PinType.
bIsConst=False,PinType.bIsWeakPointer=False,DefaultObject="/Script
/Engine.Default__KismetMathLibrary",PersistentGuid=000000000000000
00000000000000000,bHidden=True,bNotConnectable=False,bDefaultValue
IsReadOnly=False,bDefaultValueIsIgnored=False,bAdvancedView=False,
bOrphanedPin=False,)
   CustomProperties Pin
(PinId=E8501313453700D79703D984756BC756,PinName="A",PinToolTip="A\
nInteger",PinType.PinCategory="int",PinType.PinSubCategory="",PinT
ype.PinSubCategoryObject=None,PinType.PinSubCategoryMemberReferenc
e=(),PinType.PinValueType=(),PinType.ContainerType=None,PinType.bI
sReference=False,PinType.bIsConst=False,PinType.bIsWeakPointer=Fal
se,DefaultValue="0",AutogeneratedDefaultValue="0",LinkedTo=(K2Node
_VariableGet_0
BE54B3924CB1F1E2EAC2958C64B90597,),PersistentGuid=0000000000000000
0000000000000000,bHidden=False,bNotConnectable=False,bDefaultValue
IsReadOnly=False,bDefaultValueIsIgnored=False,bAdvancedView=False,
bOrphanedPin=False,)
   CustomProperties Pin
(PinId=A85FE014462F4F79BDD216BCA552197C,PinName="B",PinToolTip="B\
```

```
nInteger",PinType.PinCategory="int",PinType.PinSubCategory="",PinT
ype.PinSubCategoryObject=None,PinType.PinSubCategoryMemberReferenc
e=(),PinType.PinValueType=(),PinType.ContainerType=None,PinType.bI
sReference=False,PinType.bIsConst=False,PinType.bIsWeakPointer=Fal
se,DefaultValue="0",AutogeneratedDefaultValue="0",PersistentGuid=0
0000000000000000000000000000000,bHidden=False,bNotConnectable=Fals
e,bDefaultValueIsReadOnly=False,bDefaultValueIsIgnored=False,bAdva
ncedView=False,bOrphanedPin=False,)
   CustomProperties Pin
(PinId=B22C87A64F7750CB39CAC69C5FFCCBD5,PinName="ReturnValue",PinT
oolTip="Return Value\nBoolean\n\nReturns true if A is greater than
B (A >
B)",Direction="EGPD_Output",PinType.PinCategory="bool",PinType.Pin
SubCategory="",PinType.PinSubCategoryObject=None,PinType.PinSubCat
egoryMemberReference=(),PinType.PinValueType=(),PinType.ContainerT
ype=None,PinType.bIsReference=False,PinType.bIsConst=False,PinType
.bIsWeakPointer=False,DefaultValue="false",AutogeneratedDefaultVal
ue="false",LinkedTo=(K2Node_IfThenElse_3
B0AD42334AB9EA0F00E4E999D8269A66,),PersistentGuid=0000000000000000
0000000000000000,bHidden=False,bNotConnectable=False,bDefaultValue
IsReadOnly=False,bDefaultValueIsIgnored=False,bAdvancedView=False,
bOrphanedPin=False,)
End Object
Begin Object Class=/Script/BlueprintGraph.K2Node_VariableSet
Name="K2Node_VariableSet_14"

VariableReference=(MemberName="JumpsLeft",MemberGuid=6D41254346CB3
861C858AF86CDF84533,bSelfContext=True)
   NodePosX=-20240
   NodePosY=-832
   NodeGuid=2D2CD5464A8DE2BA77515AA9BCBB24D6
   CustomProperties Pin
(PinId=1527D75E4B5045460E9C43A89B32E4AD,PinName="execute",PinType.
```

PinCategory="exec",PinType.PinSubCategory="",PinType.PinSubCategoryObject=None,PinType.PinSubCategoryMemberReference=(),PinType.PinValueType=(),PinType.ContainerType=None,PinType.bIsReference=False,PinType.bIsConst=False,PinType.bIsWeakPointer=False,LinkedTo=(K2Node_IfThenElse_3
08F5667E42BC272B30CC3084D310EF21,),PersistentGuid=0000000000000000000000000000000,bHidden=False,bNotConnectable=False,bDefaultValueIsReadOnly=False,bDefaultValueIsIgnored=False,bAdvancedView=False,bOrphanedPin=False,)
CustomProperties Pin
(PinId=778C3CFC4C95004E4B2E8DA78C25B100,PinName="then",Direction="EGPD_Output",PinType.PinCategory="exec",PinType.PinSubCategory="",PinType.PinSubCategoryObject=None,PinType.PinSubCategoryMemberReference=(),PinType.PinValueType=(),PinType.ContainerType=None,PinType.bIsReference=False,PinType.bIsConst=False,PinType.bIsWeakPointer=False,LinkedTo=(K2Node_IfThenElse_20
37EFDB264D327B241811F5B04F5F7286,),PersistentGuid=0000000000000000000000000000000,bHidden=False,bNotConnectable=False,bDefaultValueIsReadOnly=False,bDefaultValueIsIgnored=False,bAdvancedView=False,bOrphanedPin=False,)
CustomProperties Pin
(PinId=E9B9C4E54944D9E332C7BE8BCF8A1C89,PinName="JumpsLeft",PinType.PinCategory="int",PinType.PinSubCategory="",PinType.PinSubCategoryObject=None,PinType.PinSubCategoryMemberReference=(),PinType.PinValueType=(),PinType.ContainerType=None,PinType.bIsReference=False,PinType.bIsConst=False,PinType.bIsWeakPointer=False,DefaultValue="0",AutogeneratedDefaultValue="0",LinkedTo=(K2Node_CallFunction_431418ACD240F31ED8B2780A81D2CB992B,),PersistentGuid=0000000000000000000000000000000,bHidden=False,bNotConnectable=False,bDefaultValueIsReadOnly=False,bDefaultValueIsIgnored=False,bAdvancedView=False,bOrphanedPin=False,)
CustomProperties Pin
(PinId=1063095843788500327831A4E671BAE2,PinName="Output_Get",PinTo

olTip="Retrieves the value of the variable, can use instead of a separate Get
node",Direction="EGPD_Output",PinType.PinCategory="int",PinType.PinSubCategory="",PinType.PinSubCategoryObject=None,PinType.PinSubCategoryMemberReference=(),PinType.PinValueType=(),PinType.ContainerType=None,PinType.bIsReference=False,PinType.bIsConst=False,PinType.bIsWeakPointer=False,DefaultValue="0",AutogeneratedDefaultValue="0",PersistentGuid=00000000000000000000000000000000,bHidden=False,bNotConnectable=False,bDefaultValueIsReadOnly=False,bDefaultValueIsIgnored=False,bAdvancedView=False,bOrphanedPin=False,)
   CustomProperties Pin (PinId=C194D11E400264D931C8DDB77DD76D13,PinName="self",PinFriendlyName=NSLOCTEXT("K2Node", "Target",
"Target"),PinType.PinCategory="object",PinType.PinSubCategory="",PinType.PinSubCategoryObject=BlueprintGeneratedClass'"/Game/Classic_Shooter/FirstPersonCharacter/BP_FirstPersonCharacter.BP_FirstPersonCharacter_C"',PinType.PinSubCategoryMemberReference=(),PinType.PinValueType=(),PinType.ContainerType=None,PinType.bIsReference=False,PinType.bIsConst=False,PinType.bIsWeakPointer=False,PersistentGuid=00000000000000000000000000000000,bHidden=True,bNotConnectable=False,bDefaultValueIsReadOnly=False,bDefaultValueIsIgnored=False,bAdvancedView=False,bOrphanedPin=False,)
End Object
Begin Object Class=/Script/BlueprintGraph.K2Node_CallFunction Name="K2Node_CallFunction_43"
   bIsPureFunc=True

FunctionReference=(MemberParent=Class'"/Script/Engine.KismetMathLibrary"',MemberName="Subtract_IntInt")
   NodePosX=-20416
   NodePosY=-912
   NodeGuid=554806B34168A9ADA7D15680F2CDFA2E

Продовження додатку В

CustomProperties Pin
(PinId=CEF1DADB4E962891593BC7B2E870DF89,PinName="self",PinFriendly
Name=NSLOCTEXT("K2Node", "Target",
"Target"),PinToolTip="Target\nKismet Math Library Object
Reference",PinType.PinCategory="object",PinType.PinSubCategory="",
PinType.PinSubCategoryObject=Class'"/Script/Engine.KismetMathLibra
ry"',PinType.PinSubCategoryMemberReference=(),PinType.PinValueType
=(),PinType.ContainerType=None,PinType.bIsReference=False,PinType.
bIsConst=False,PinType.bIsWeakPointer=False,DefaultObject="/Script
/Engine.Default__KismetMathLibrary",PersistentGuid=000000000000000
00000000000000000,bHidden=True,bNotConnectable=False,bDefaultValue
IsReadOnly=False,bDefaultValueIsIgnored=False,bAdvancedView=False,
bOrphanedPin=False,)
CustomProperties Pin
(PinId=65C9A3554C22978CE5321A8C4750DCEC,PinName="A",PinToolTip="A\
nInteger",PinType.PinCategory="int",PinType.PinSubCategory="",PinT
ype.PinSubCategoryObject=None,PinType.PinSubCategoryMemberReferenc
e=(),PinType.PinValueType=(),PinType.ContainerType=None,PinType.bI
sReference=False,PinType.bIsConst=False,PinType.bIsWeakPointer=Fal
se,DefaultValue="0",AutogeneratedDefaultValue="0",LinkedTo=(K2Node
_VariableGet_0
BE54B3924CB1F1E2EAC2958C64B90597,),PersistentGuid=000000000000000
00000000000000000,bHidden=False,bNotConnectable=False,bDefaultValue
IsReadOnly=False,bDefaultValueIsIgnored=False,bAdvancedView=False,
bOrphanedPin=False,)
CustomProperties Pin
(PinId=9DDD6536466160CF1AFC469609295989,PinName="B",PinToolTip="B\
nInteger",PinType.PinCategory="int",PinType.PinSubCategory="",PinT
ype.PinSubCategoryObject=None,PinType.PinSubCategoryMemberReferenc
e=(),PinType.PinValueType=(),PinType.ContainerType=None,PinType.bI
sReference=False,PinType.bIsConst=False,PinType.bIsWeakPointer=Fal
se,DefaultValue="1",AutogeneratedDefaultValue="1",PersistentGuid=0
00000000000000000000000000000000,bHidden=False,bNotConnectable=Fals

```
e,bDefaultValueIsReadOnly=False,bDefaultValueIsIgnored=False,bAdva
ncedView=False,bOrphanedPin=False,)
   CustomProperties Pin
(PinId=1418ACD240F31ED8B2780A81D2CB992B,PinName="ReturnValue",PinT
oolTip="Return Value\nInteger\n\nSubtraction (A -
B)",Direction="EGPD_Output",PinType.PinCategory="int",PinType.PinS
ubCategory="",PinType.PinSubCategoryObject=None,PinType.PinSubCate
goryMemberReference=(),PinType.PinValueType=(),PinType.ContainerTy
pe=None,PinType.bIsReference=False,PinType.bIsConst=False,PinType.
bIsWeakPointer=False,DefaultValue="0",AutogeneratedDefaultValue="0
",LinkedTo=(K2Node_VariableSet_14
E9B9C4E54944D9E332C7BE8BCF8A1C89,),PersistentGuid=0000000000000000
0000000000000000,bHidden=False,bNotConnectable=False,bDefaultValue
IsReadOnly=False,bDefaultValueIsIgnored=False,bAdvancedView=False,
bOrphanedPin=False,)
End Object
Begin Object Class=/Script/BlueprintGraph.K2Node_IfThenElse
Name="K2Node_IfThenElse_20"
   NodePosX=-19904
   NodePosY=-848
   NodeGuid=D374CB8E4E05DCB38451C7B006BD1DA9
   CustomProperties Pin
(PinId=37EFDB264D327B241811F5B04F5F7286,PinName="execute",PinType.
PinCategory="exec",PinType.PinSubCategory="",PinType.PinSubCategor
yObject=None,PinType.PinSubCategoryMemberReference=(),PinType.PinV
alueType=(),PinType.ContainerType=None,PinType.bIsReference=False,
PinType.bIsConst=False,PinType.bIsWeakPointer=False,LinkedTo=(K2No
de_VariableSet_14
778C3CFC4C95004E4B2E8DA78C25B100,),PersistentGuid=0000000000000000
0000000000000000,bHidden=False,bNotConnectable=False,bDefaultValue
IsReadOnly=False,bDefaultValueIsIgnored=False,bAdvancedView=False,
bOrphanedPin=False,)
```

Продовження додатку В

CustomProperties Pin
(PinId=5523F1DC469B7B48FC137D973F61D4AA,PinName="Condition",PinTyp
e.PinCategory="bool",PinType.PinSubCategory="",PinType.PinSubCateg
oryObject=None,PinType.PinSubCategoryMemberReference=(),PinType.Pi
nValueType=(),PinType.ContainerType=None,PinType.bIsReference=Fals
e,PinType.bIsConst=False,PinType.bIsWeakPointer=False,DefaultValue
="true",AutogeneratedDefaultValue="true",LinkedTo=(K2Node_CallFunc
tion_64
91A77FD14586C81EF94D25AC91D72F0E,),PersistentGuid=0000000000000000
0000000000000000,bHidden=False,bNotConnectable=False,bDefaultValue
IsReadOnly=False,bDefaultValueIsIgnored=False,bAdvancedView=False,
bOrphanedPin=False,)
CustomProperties Pin
(PinId=8D2804034864525FFABF66AC278AB431,PinName="then",PinFriendly
Name=NSLOCTEXT("K2Node", "true",
"true"),Direction="EGPD_Output",PinType.PinCategory="exec",PinType
.PinSubCategory="",PinType.PinSubCategoryObject=None,PinType.PinSu
bCategoryMemberReference=(),PinType.PinValueType=(),PinType.Contai
nerType=None,PinType.bIsReference=False,PinType.bIsConst=False,Pin
Type.bIsWeakPointer=False,LinkedTo=(K2Node_CallFunction_48
8FEE1967465FAB8455E70A9546C401F9,),PersistentGuid=0000000000000000
0000000000000000,bHidden=False,bNotConnectable=False,bDefaultValue
IsReadOnly=False,bDefaultValueIsIgnored=False,bAdvancedView=False,
bOrphanedPin=False,)
CustomProperties Pin
(PinId=1CE14CE54FDE7BE5BA91139DCE458DE4,PinName="else",PinFriendly
Name=NSLOCTEXT("K2Node", "false",
"false"),Direction="EGPD_Output",PinType.PinCategory="exec",PinTyp
e.PinSubCategory="",PinType.PinSubCategoryObject=None,PinType.PinS
ubCategoryMemberReference=(),PinType.PinValueType=(),PinType.Conta
inerType=None,PinType.bIsReference=False,PinType.bIsConst=False,Pi
nType.bIsWeakPointer=False,LinkedTo=(K2Node_IfThenElse_21
83D868E34222E3B8C4EA6FBF3F090BF5,),PersistentGuid=0000000000000000

```
0000000000000000,bHidden=False,bNotConnectable=False,bDefaultValue
IsReadOnly=False,bDefaultValueIsIgnored=False,bAdvancedView=False,
bOrphanedPin=False,)
End Object
Begin Object Class=/Script/BlueprintGraph.K2Node_VariableGet
Name="K2Node_VariableGet_19"

VariableReference=(MemberName="JumpsLeft",MemberGuid=6D41254346CB3
861C858AF86CDF84533,bSelfContext=True)
   NodePosX=-20048
   NodePosY=-672
   NodeGuid=ED5783734BD9E7124B053FAA4F4C9BB5
   CustomProperties Pin
(PinId=BE54B3924CB1F1E2EAC2958C64B90597,PinName="JumpsLeft",Direct
ion="EGPD_Output",PinType.PinCategory="int",PinType.PinSubCategory
="",PinType.PinSubCategoryObject=None,PinType.PinSubCategoryMember
Reference=(),PinType.PinValueType=(),PinType.ContainerType=None,Pi
nType.bIsReference=False,PinType.bIsConst=False,PinType.bIsWeakPoi
nter=False,DefaultValue="0",AutogeneratedDefaultValue="0",LinkedTo
=(K2Node_Knot_2
8195999F4E4827EB4E71658B0681664E,),PersistentGuid=0000000000000000
0000000000000000,bHidden=False,bNotConnectable=False,bDefaultValue
IsReadOnly=False,bDefaultValueIsIgnored=False,bAdvancedView=False,
bOrphanedPin=False,)
   CustomProperties Pin
(PinId=95214B9A4A4902F6B64B41B7152AFCBE,PinName="self",PinFriendly
Name=NSLOCTEXT("K2Node", "Target",
"Target"),PinType.PinCategory="object",PinType.PinSubCategory="",P
inType.PinSubCategoryObject=BlueprintGeneratedClass'"/Game/Classic
_Shooter/FirstPersonCharacter/BP_FirstPersonCharacter.BP_FirstPers
onCharacter_C"',PinType.PinSubCategoryMemberReference=(),PinType.P
inValueType=(),PinType.ContainerType=None,PinType.bIsReference=Fal
se,PinType.bIsConst=False,PinType.bIsWeakPointer=False,PersistentG
```

```
uid=00000000000000000000000000000000,bHidden=True,bNotConnectable=
False,bDefaultValueIsReadOnly=False,bDefaultValueIsIgnored=False,b
AdvancedView=False,bOrphanedPin=False,)
End Object
Begin Object Class=/Script/BlueprintGraph.K2Node_CallFunction
Name="K2Node_CallFunction_163"

FunctionReference=(MemberName="LaunchCharacter",bSelfContext=True)
   NodePosX=-19088
   NodePosY=-464
   NodeGuid=CE35959D420BC2BD9536768D913E56E6
   CustomProperties Pin
(PinId=8FEE1967465FAB8455E70A9546C401F9,PinName="execute",PinToolT
ip="\nExec",PinType.PinCategory="exec",PinType.PinSubCategory="",P
inType.PinSubCategoryObject=None,PinType.PinSubCategoryMemberRefer
ence=(),PinType.PinValueType=(),PinType.ContainerType=None,PinType
.bIsReference=False,PinType.bIsConst=False,PinType.bIsWeakPointer=
False,LinkedTo=(K2Node_IfThenElse_21
8A2A89C74763EC9EF691358699D2DB56,),PersistentGuid=0000000000000000
0000000000000000,bHidden=False,bNotConnectable=False,bDefaultValue
IsReadOnly=False,bDefaultValueIsIgnored=False,bAdvancedView=False,
bOrphanedPin=False,)
   CustomProperties Pin
(PinId=BE6B05F840E6D047EE12D5924F080631,PinName="then",PinToolTip=
"\nExec",Direction="EGPD_Output",PinType.PinCategory="exec",PinTyp
e.PinSubCategory="",PinType.PinSubCategoryObject=None,PinType.PinS
ubCategoryMemberReference=(),PinType.PinValueType=(),PinType.Conta
inerType=None,PinType.bIsReference=False,PinType.bIsConst=False,Pi
nType.bIsWeakPointer=False,PersistentGuid=0000000000000000000000000
00000000,bHidden=False,bNotConnectable=False,bDefaultValueIsReadOn
ly=False,bDefaultValueIsIgnored=False,bAdvancedView=False,bOrphane
dPin=False,)
```

Продовження додатку В

```
   CustomProperties Pin
(PinId=D28A5C094C9E2718ACEFB1BBA7CFDD33,PinName="self",PinFriendly
Name=NSLOCTEXT("K2Node", "Target",
"Target"),PinToolTip="Target\nCharacter Object
Reference",PinType.PinCategory="object",PinType.PinSubCategory="",
PinType.PinSubCategoryObject=Class'"/Script/Engine.Character"',Pin
Type.PinSubCategoryMemberReference=(),PinType.PinValueType=(),PinT
ype.ContainerType=None,PinType.bIsReference=False,PinType.bIsConst
=False,PinType.bIsWeakPointer=False,PersistentGuid=000000000000000
00000000000000000,bHidden=False,bNotConnectable=False,bDefaultValu
eIsReadOnly=False,bDefaultValueIsIgnored=False,bAdvancedView=False
,bOrphanedPin=False,)
   CustomProperties Pin
(PinId=464755E94D36D197DE62F0836D1E551C,PinName="LaunchVelocity",P
inToolTip="Launch Velocity\nVector\n\nis the velocity to impart to
the
Character",PinType.PinCategory="struct",PinType.PinSubCategory="",
PinType.PinSubCategoryObject=ScriptStruct'"/Script/CoreUObject.Vec
tor"',PinType.PinSubCategoryMemberReference=(),PinType.PinValueTyp
e=(),PinType.ContainerType=None,PinType.bIsReference=False,PinType
.bIsConst=False,PinType.bIsWeakPointer=False,DefaultValue="0, 0,
0",AutogeneratedDefaultValue="0, 0,
0",LinkedTo=(K2Node_CommutativeAssociativeBinaryOperator_7
FF1A7F0E4CB732A9293A1C9EF5D41A0F,),PersistentGuid=0000000000000000
0000000000000000,bHidden=False,bNotConnectable=False,bDefaultValue
IsReadOnly=False,bDefaultValueIsIgnored=False,bAdvancedView=False,
bOrphanedPin=False,)
   CustomProperties Pin
(PinId=C4A8F6DA40546C5B12309BA846F5E68F,PinName="bXYOverride",PinT
oolTip="XYOverride\nBoolean\n\nif true replace the XY part of the
Character\'s velocity instead of adding to
it.",PinType.PinCategory="bool",PinType.PinSubCategory="",PinType.
PinSubCategoryObject=None,PinType.PinSubCategoryMemberReference=()
```

,PinType.PinValueType=(),PinType.ContainerType=None,PinType.bIsReference=False,PinType.bIsConst=False,PinType.bIsWeakPointer=False,DefaultValue="true",AutogeneratedDefaultValue="false",PersistentGuid=00000000000000000000000000000000,bHidden=False,bNotConnectable=False,bDefaultValueIsReadOnly=False,bDefaultValueIsIgnored=False,bAdvancedView=False,bOrphanedPin=False,)

CustomProperties Pin (PinId=FEC28E614CEF9809927BC2BF2C1BAA19,PinName="bZOverride",PinToolTip="ZOverride\nBoolean\n\nif true replace the Z component of the Character\'s velocity instead of adding to it.",PinType.PinCategory="bool",PinType.PinSubCategory="",PinType.PinSubCategoryObject=None,PinType.PinSubCategoryMemberReference=(),PinType.PinValueType=(),PinType.ContainerType=None,PinType.bIsReference=False,PinType.bIsConst=False,PinType.bIsWeakPointer=False,DefaultValue="true",AutogeneratedDefaultValue="false",PersistentGuid=00000000000000000000000000000000,bHidden=False,bNotConnectable=False,bDefaultValueIsReadOnly=False,bDefaultValueIsIgnored=False,bAdvancedView=False,bOrphanedPin=False,)

End Object

Begin Object Class=/Script/BlueprintGraph.K2Node_CallFunction Name="K2Node_CallFunction_64"

bIsPureFunc=True

FunctionReference=(MemberParent=Class'"/Script/Engine.KismetMathLibrary"',MemberName="EqualEqual_IntInt")

NodePosX=-20032

NodePosY=-752

NodeGuid=B006417C47E514A8E96175BF132D3ABB

CustomProperties Pin (PinId=BBFB359A4FA6F1009481988C169A7522,PinName="self",PinFriendlyName=NSLOCTEXT("K2Node", "Target", "Target"),PinToolTip="Target\nKismet Math Library Object Reference",PinType.PinCategory="object",PinType.PinSubCategory="",

PinType.PinSubCategoryObject=Class'"/Script/Engine.KismetMathLibrary"',PinType.PinSubCategoryMemberReference=(),PinType.PinValueType=(),PinType.ContainerType=None,PinType.bIsReference=False,PinType.bIsConst=False,PinType.bIsWeakPointer=False,DefaultObject="/Script/Engine.Default__KismetMathLibrary",PersistentGuid=00000000000000000000000000000000,bHidden=True,bNotConnectable=False,bDefaultValueIsReadOnly=False,bDefaultValueIsIgnored=False,bAdvancedView=False,bOrphanedPin=False,)

CustomProperties Pin (PinId=EA0EEDF741789968C08BAAA427328EB4,PinName="A",PinToolTip="A\nInteger",PinType.PinCategory="int",PinType.PinSubCategory="",PinType.PinSubCategoryObject=None,PinType.PinSubCategoryMemberReference=(),PinType.PinValueType=(),PinType.ContainerType=None,PinType.bIsReference=False,PinType.bIsConst=False,PinType.bIsWeakPointer=False,DefaultValue="0",AutogeneratedDefaultValue="0",LinkedTo=(K2Node_Knot_2 4F9347224A4B699CE48C0F9C368F4F79,),PersistentGuid=00000000000000000000000000000000,bHidden=False,bNotConnectable=False,bDefaultValueIsReadOnly=False,bDefaultValueIsIgnored=False,bAdvancedView=False,bOrphanedPin=False,)

CustomProperties Pin (PinId=88FD535A41D3D6560910BFA30B8093A4,PinName="B",PinToolTip="B\nInteger",PinType.PinCategory="int",PinType.PinSubCategory="",PinType.PinSubCategoryObject=None,PinType.PinSubCategoryMemberReference=(),PinType.PinValueType=(),PinType.ContainerType=None,PinType.bIsReference=False,PinType.bIsConst=False,PinType.bIsWeakPointer=False,DefaultValue="1",AutogeneratedDefaultValue="0",PersistentGuid=00000000000000000000000000000000,bHidden=False,bNotConnectable=False,bDefaultValueIsReadOnly=False,bDefaultValueIsIgnored=False,bAdvancedView=False,bOrphanedPin=False,)

CustomProperties Pin (PinId=91A77FD14586C81EF94D25AC91D72F0E,PinName="ReturnValue",PinToolTip="Return Value\nBoolean\n\nReturns true if A is equal to B

```
(A ==
B)",Direction="EGPD_Output",PinType.PinCategory="bool",PinType.Pin
SubCategory="",PinType.PinSubCategoryObject=None,PinType.PinSubCat
egoryMemberReference=(),PinType.PinValueType=(),PinType.ContainerT
ype=None,PinType.bIsReference=False,PinType.bIsConst=False,PinType
.bIsWeakPointer=False,DefaultValue="false",AutogeneratedDefaultVal
ue="false",LinkedTo=(K2Node_IfThenElse_20
5523F1DC469B7B48FC137D973F61D4AA,),PersistentGuid=0000000000000000
0000000000000000,bHidden=False,bNotConnectable=False,bDefaultValue
IsReadOnly=False,bDefaultValueIsIgnored=False,bAdvancedView=False,
bOrphanedPin=False,)
End Object
Begin Object Class=/Script/BlueprintGraph.K2Node_VariableGet
Name="K2Node_VariableGet_29"

VariableReference=(MemberName="CharacterMovement",bSelfContext=Tru
e)
   NodePosX=-20064
   NodePosY=-608
   NodeGuid=8F2B8FA449BAE44E20AD66AEFD7B1ACD
   CustomProperties Pin
(PinId=F8E6599F4E4B24E6DFE71B99FBF99545,PinName="CharacterMovement
",Direction="EGPD_Output",PinType.PinCategory="object",PinType.Pin
SubCategory="",PinType.PinSubCategoryObject=Class'"/Script/Engine.
CharacterMovementComponent"',PinType.PinSubCategoryMemberReference
=(),PinType.PinValueType=(),PinType.ContainerType=None,PinType.bIs
Reference=False,PinType.bIsConst=False,PinType.bIsWeakPointer=Fals
e,LinkedTo=(K2Node_CallFunction_122
F682CE7A4EAF12087E17DF9F9A41A8D6,),PersistentGuid=0000000000000000
0000000000000000,bHidden=False,bNotConnectable=False,bDefaultValue
IsReadOnly=False,bDefaultValueIsIgnored=False,bAdvancedView=False,
bOrphanedPin=False,)
```

```
   CustomProperties Pin
(PinId=5F2C55EB499E4CFC3FA8FA80329BDAA0,PinName="self",PinFriendly
Name=NSLOCTEXT("K2Node", "Target",
"Target"),PinType.PinCategory="object",PinType.PinSubCategory="",P
inType.PinSubCategoryObject=Class'"/Script/Engine.Character"',PinT
ype.PinSubCategoryMemberReference=(),PinType.PinValueType=(),PinTy
pe.ContainerType=None,PinType.bIsReference=False,PinType.bIsConst=
False,PinType.bIsWeakPointer=False,PersistentGuid=0000000000000000
0000000000000000,bHidden=True,bNotConnectable=False,bDefaultValueI
sReadOnly=False,bDefaultValueIsIgnored=False,bAdvancedView=False,b
OrphanedPin=False,)
End Object
Begin Object Class=/Script/BlueprintGraph.K2Node_CallFunction
Name="K2Node_CallFunction_122"
   bIsPureFunc=True
   bIsConstFunc=True

FunctionReference=(MemberParent=Class'"/Script/Engine.PawnMovement
Component"',MemberName="GetLastInputVector")
   NodePosX=-19888
   NodePosY=-640
   NodeGuid=FB954DE948F12B2C9506A3804BCDABCF
   CustomProperties Pin
(PinId=F682CE7A4EAF12087E17DF9F9A41A8D6,PinName="self",PinFriendly
Name=NSLOCTEXT("K2Node", "Target",
"Target"),PinToolTip="Target\nPawn Movement Component Object
Reference",PinType.PinCategory="object",PinType.PinSubCategory="",
PinType.PinSubCategoryObject=Class'"/Script/Engine.PawnMovementCom
ponent"',PinType.PinSubCategoryMemberReference=(),PinType.PinValue
Type=(),PinType.ContainerType=None,PinType.bIsReference=False,PinT
ype.bIsConst=False,PinType.bIsWeakPointer=False,LinkedTo=(K2Node_V
ariableGet_29
F8E6599F4E4B24E6DFE71B99FBF99545,),PersistentGuid=0000000000000000
```

```
0000000000000000,bHidden=False,bNotConnectable=False,bDefaultValue
IsReadOnly=False,bDefaultValueIsIgnored=False,bAdvancedView=False,
bOrphanedPin=False,)
   CustomProperties Pin
(PinId=32554D544C70FFB103AA909DEC0DAD7D,PinName="ReturnValue",PinT
oolTip="Return Value\nVector\n\nThe last input vector in world
space that was processed by ConsumeInputVector(). See:
AddInputVector(), ConsumeInputVector(),
GetPendingInputVector()",Direction="EGPD_Output",PinType.PinCatego
ry="struct",PinType.PinSubCategory="",PinType.PinSubCategoryObject
=ScriptStruct'"/Script/CoreUObject.Vector"',PinType.PinSubCategory
MemberReference=(),PinType.PinValueType=(),PinType.ContainerType=N
one,PinType.bIsReference=False,PinType.bIsConst=False,PinType.bIsW
eakPointer=False,DefaultValue="0, 0,
0",AutogeneratedDefaultValue="0, 0,
0",LinkedTo=(K2Node_CallFunction_164
342A734549E7E3F170253D9066C80411,K2Node_CallFunction_161
7D3C981C40A4188388831DB7D349A70B,K2Node_CallFunction_126
342A734549E7E3F170253D9066C80411,),PersistentGuid=0000000000000000
0000000000000000,bHidden=False,bNotConnectable=False,bDefaultValue
IsReadOnly=False,bDefaultValueIsIgnored=False,bAdvancedView=False,
bOrphanedPin=False,)
End Object
Begin Object Class=/Script/BlueprintGraph.K2Node_CallFunction
Name="K2Node_CallFunction_164"
   bIsPureFunc=True

FunctionReference=(MemberParent=Class'"/Script/Engine.KismetMathLi
brary"',MemberName="Multiply_VectorInt")
   NodePosX=-19600
   NodePosY=-592
   NodeGuid=4D1B1EC74D91E7AB745706BE9D794123
```

Продовження додатку В

```
   CustomProperties Pin
(PinId=6A7E0C9644FAD66380060F81045ED174,PinName="self",PinFriendly
Name=NSLOCTEXT("K2Node", "Target",
"Target"),PinToolTip="Target\nKismet Math Library Object
Reference",PinType.PinCategory="object",PinType.PinSubCategory="",
PinType.PinSubCategoryObject=Class'"/Script/Engine.KismetMathLibra
ry"',PinType.PinSubCategoryMemberReference=(),PinType.PinValueType
=(),PinType.ContainerType=None,PinType.bIsReference=False,PinType.
bIsConst=False,PinType.bIsWeakPointer=False,DefaultObject="/Script
/Engine.Default__KismetMathLibrary",PersistentGuid=000000000000000
00000000000000000,bHidden=True,bNotConnectable=False,bDefaultValue
IsReadOnly=False,bDefaultValueIsIgnored=False,bAdvancedView=False,
bOrphanedPin=False,)
   CustomProperties Pin
(PinId=342A734549E7E3F170253D9066C80411,PinName="A",PinToolTip="A\
nVector",PinType.PinCategory="struct",PinType.PinSubCategory="",Pi
nType.PinSubCategoryObject=ScriptStruct'"/Script/CoreUObject.Vecto
r"',PinType.PinSubCategoryMemberReference=(),PinType.PinValueType=
(),PinType.ContainerType=None,PinType.bIsReference=False,PinType.b
IsConst=False,PinType.bIsWeakPointer=False,DefaultValue="0, 0,
0",AutogeneratedDefaultValue="0, 0,
0",LinkedTo=(K2Node_CallFunction_122
32554D544C70FFB103AA909DEC0DAD7D,),PersistentGuid=000000000000000
00000000000000000,bHidden=False,bNotConnectable=False,bDefaultValue
IsReadOnly=False,bDefaultValueIsIgnored=False,bAdvancedView=False,
bOrphanedPin=False,)
   CustomProperties Pin
(PinId=A746BD4F4F7786EC12147D98A9C64EAC,PinName="B",PinToolTip="B\
nInteger",PinType.PinCategory="int",PinType.PinSubCategory="",PinT
ype.PinSubCategoryObject=None,PinType.PinSubCategoryMemberReferenc
e=(),PinType.PinValueType=(),PinType.ContainerType=None,PinType.bI
sReference=False,PinType.bIsConst=False,PinType.bIsWeakPointer=Fal
se,DefaultValue="500",AutogeneratedDefaultValue="0",PersistentGuid
```

```
=00000000000000000000000000000000,bHidden=False,bNotConnectable=Fa
lse,bDefaultValueIsReadOnly=False,bDefaultValueIsIgnored=False,bAd
vancedView=False,bOrphanedPin=False,)
   CustomProperties Pin
(PinId=698840F541A6B3C43B8A479D15E79B1B,PinName="ReturnValue",PinT
oolTip="Return Value\nVector\n\nScales Vector A by
B",Direction="EGPD_Output",PinType.PinCategory="struct",PinType.Pi
nSubCategory="",PinType.PinSubCategoryObject=ScriptStruct'"/Script
/CoreUObject.Vector"',PinType.PinSubCategoryMemberReference=(),Pin
Type.PinValueType=(),PinType.ContainerType=None,PinType.bIsReferen
ce=False,PinType.bIsConst=False,PinType.bIsWeakPointer=False,Defau
ltValue="0, 0, 0",AutogeneratedDefaultValue="0, 0,
0",LinkedTo=(K2Node_CommutativeAssociativeBinaryOperator_14
CEBD2BF5406C180663D0E9850FD2845A,),PersistentGuid=0000000000000000
0000000000000000,bHidden=False,bNotConnectable=False,bDefaultValue
IsReadOnly=False,bDefaultValueIsIgnored=False,bAdvancedView=False,
bOrphanedPin=False,)
End Object
Begin Object
Class=/Script/BlueprintGraph.K2Node_CommutativeAssociativeBinaryOp
erator Name="K2Node_CommutativeAssociativeBinaryOperator_14"
   bIsPureFunc=True

FunctionReference=(MemberParent=Class'"/Script/Engine.KismetMathLi
brary"',MemberName="Add_VectorVector")
   NodePosX=-19456
   NodePosY=-576
   NodeGuid=1CA13A8346ED4CA4763664914798579C
   CustomProperties Pin
(PinId=27862FD643D10B3FA7C58AA7607386C2,PinName="self",PinFriendly
Name=NSLOCTEXT("K2Node", "Target",
"Target"),PinToolTip="Target\nKismet Math Library Object
Reference",PinType.PinCategory="object",PinType.PinSubCategory="",
```

Продовження додатку В

PinType.PinSubCategoryObject=Class'"/Script/Engine.KismetMathLibrary"',PinType.PinSubCategoryMemberReference=(),PinType.PinValueType=(),PinType.ContainerType=None,PinType.bIsReference=False,PinType.bIsConst=False,PinType.bIsWeakPointer=False,DefaultObject="/Script/Engine.Default__KismetMathLibrary",PersistentGuid=00000000000000000000000000000000,bHidden=True,bNotConnectable=False,bDefaultValueIsReadOnly=False,bDefaultValueIsIgnored=False,bAdvancedView=False,bOrphanedPin=False,)

CustomProperties Pin (PinId=CEBD2BF5406C180663D0E9850FD2845A,PinName="A",PinToolTip="A\nVector",PinType.PinCategory="struct",PinType.PinSubCategory="",PinType.PinSubCategoryObject=ScriptStruct'"/Script/CoreUObject.Vector"',PinType.PinSubCategoryMemberReference=(),PinType.PinValueType=(),PinType.ContainerType=None,PinType.bIsReference=False,PinType.bIsConst=False,PinType.bIsWeakPointer=False,DefaultValue="0, 0, 0",AutogeneratedDefaultValue="0, 0, 0",LinkedTo=(K2Node_CallFunction_164 698840F541A6B3C43B8A479D15E79B1B,),PersistentGuid=00000000000000000000000000000000,bHidden=False,bNotConnectable=False,bDefaultValueIsReadOnly=False,bDefaultValueIsIgnored=False,bAdvancedView=False,bOrphanedPin=False,)

CustomProperties Pin (PinId=B0327EFF42626F9F9FBB53B671E31D76,PinName="B",PinToolTip="B\nVector",PinType.PinCategory="struct",PinType.PinSubCategory="",PinType.PinSubCategoryObject=ScriptStruct'"/Script/CoreUObject.Vector"',PinType.PinSubCategoryMemberReference=(),PinType.PinValueType=(),PinType.ContainerType=None,PinType.bIsReference=False,PinType.bIsConst=False,PinType.bIsWeakPointer=False,DefaultValue="0, 0,300.000000",AutogeneratedDefaultValue="0, 0, 0",PersistentGuid=00000000000000000000000000000000,bHidden=False,bNotConnectable=False,bDefaultValueIsReadOnly=False,bDefaultValueIsIgnored=False,bAdvancedView=False,bOrphanedPin=False,)

```
   CustomProperties Pin
(PinId=FF1A7F0E4CB732A9293A1C9EF5D41A0F,PinName="ReturnValue",PinT
oolTip="Return Value\nVector\n\nVector
addition",Direction="EGPD_Output",PinType.PinCategory="struct",Pin
Type.PinSubCategory="",PinType.PinSubCategoryObject=ScriptStruct'"
/Script/CoreUObject.Vector"',PinType.PinSubCategoryMemberReference
=(),PinType.PinValueType=(),PinType.ContainerType=None,PinType.bIs
Reference=False,PinType.bIsConst=False,PinType.bIsWeakPointer=Fals
e,DefaultValue="0, 0, 0",AutogeneratedDefaultValue="0, 0,
0",LinkedTo=(K2Node_CallFunction_48
464755E94D36D197DE62F0836D1E551C,),PersistentGuid=0000000000000000
0000000000000000,bHidden=False,bNotConnectable=False,bDefaultValue
IsReadOnly=False,bDefaultValueIsIgnored=False,bAdvancedView=False,
bOrphanedPin=False,)
End Object
```

ДОДАТОК Г

Візуальний код

(рекомендований)

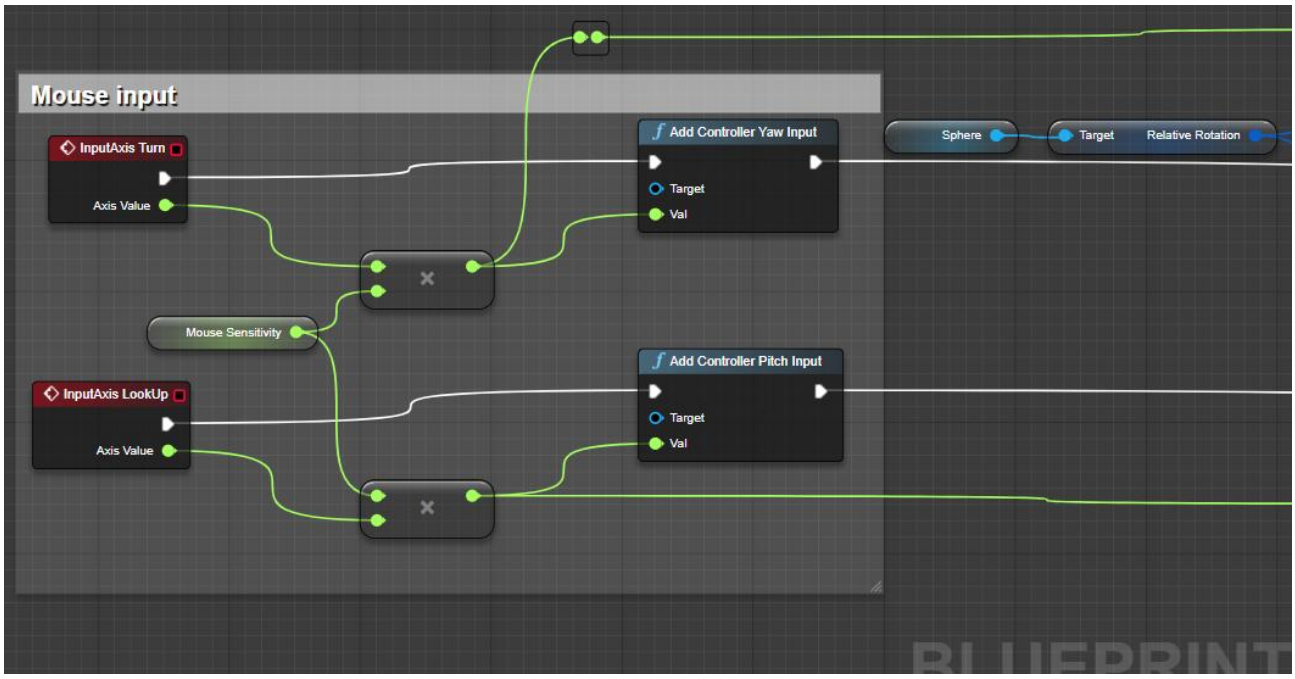

Рисунок Г.1 – механіка анімації руху миші, частина 1

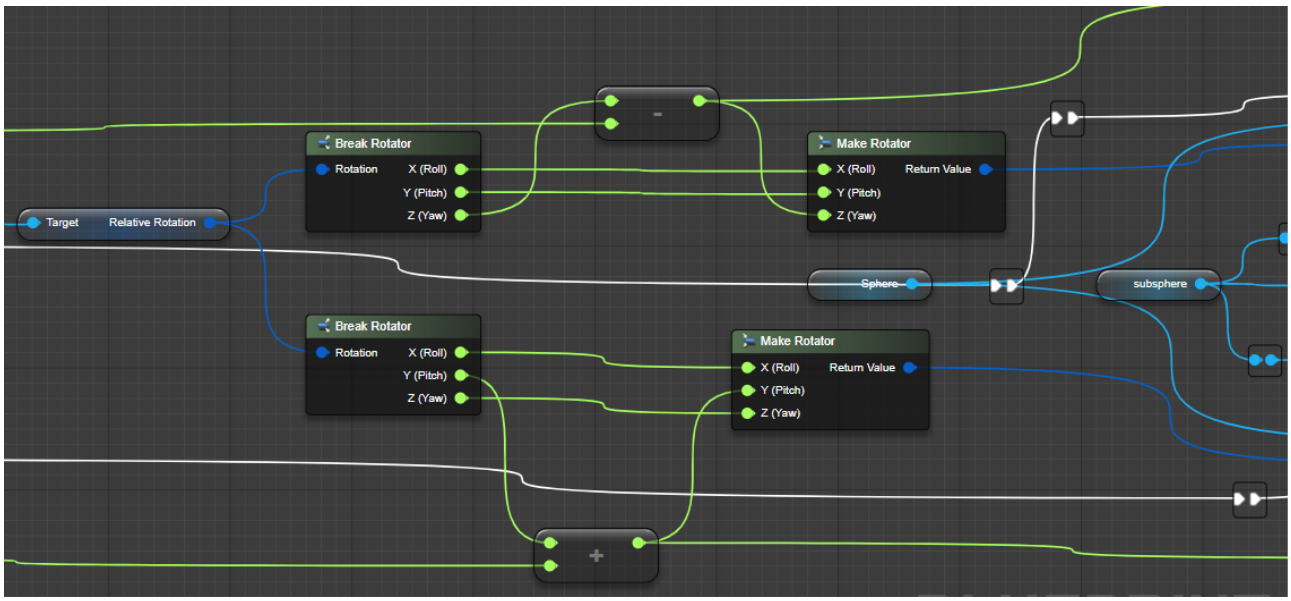

Рисунок Г.2 – механіка анімації руху миші, частина 2

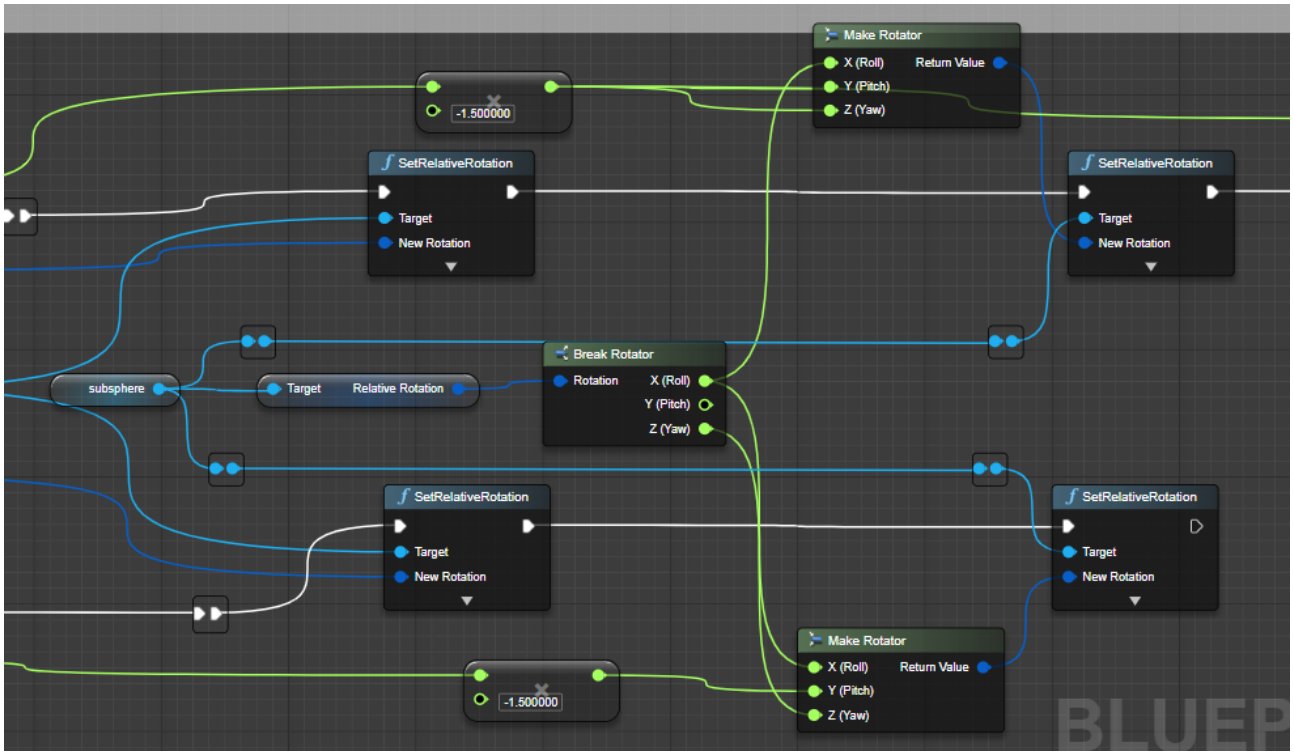

Рисунок Г.3 – механіка анімації руху миші, частина 3

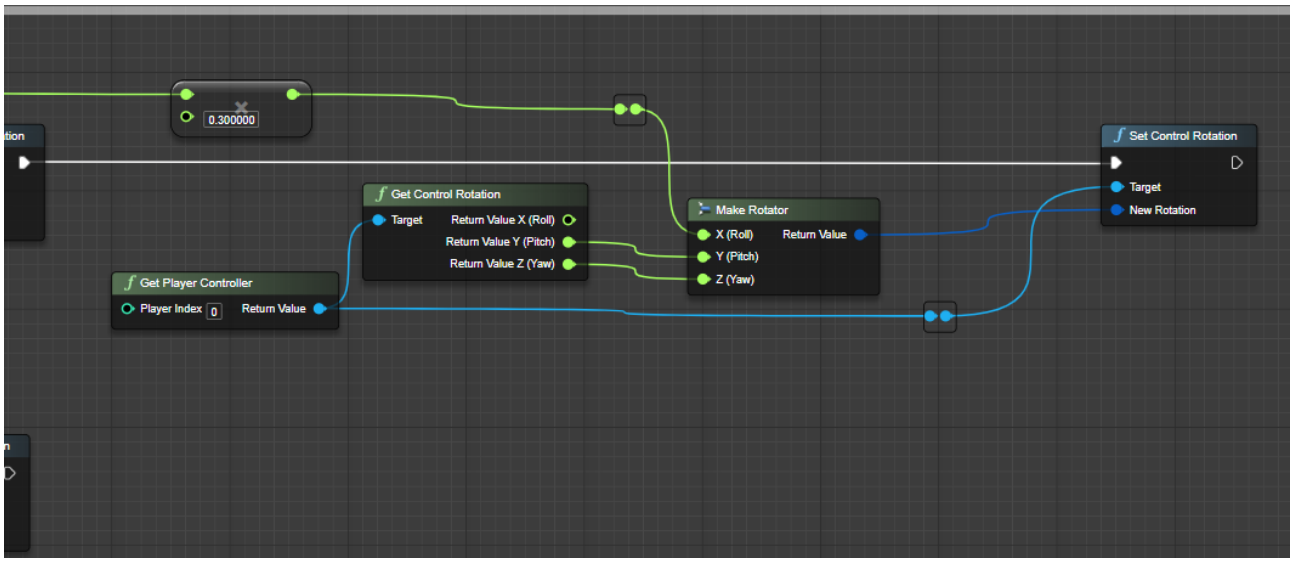

Рисунок Г.4 – механіка анімації руху миші, частина 4

Продовження додатку Г

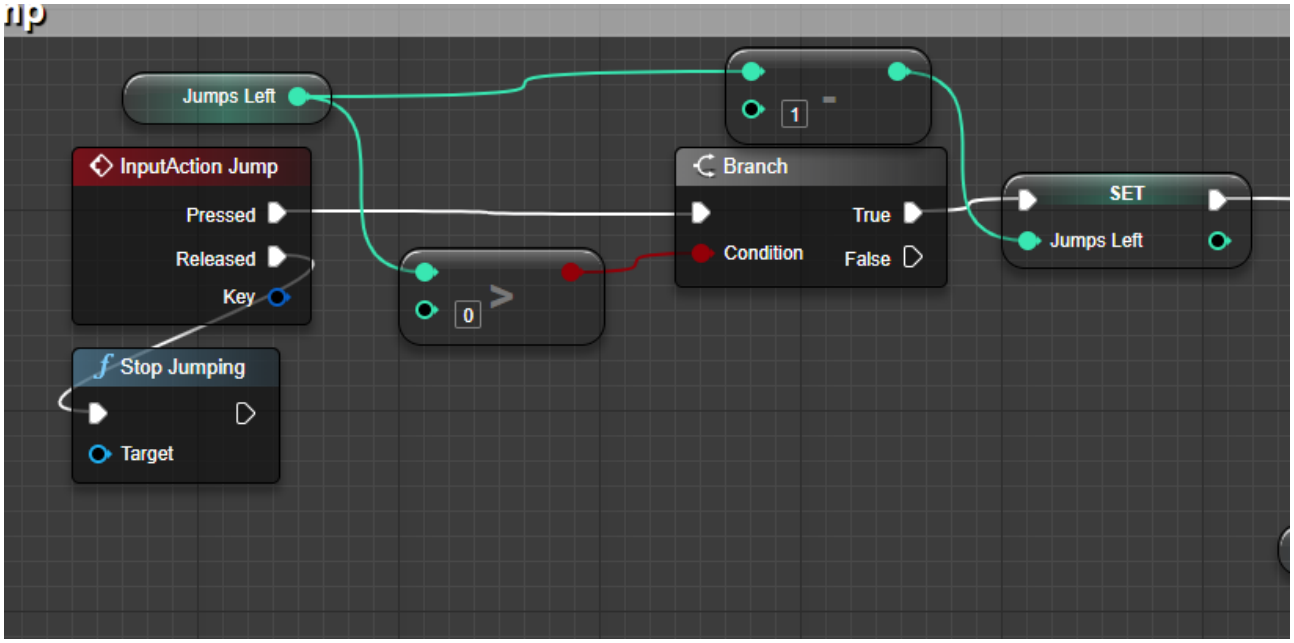

Рисунок Г.5 – механіка стрибку, частина 1

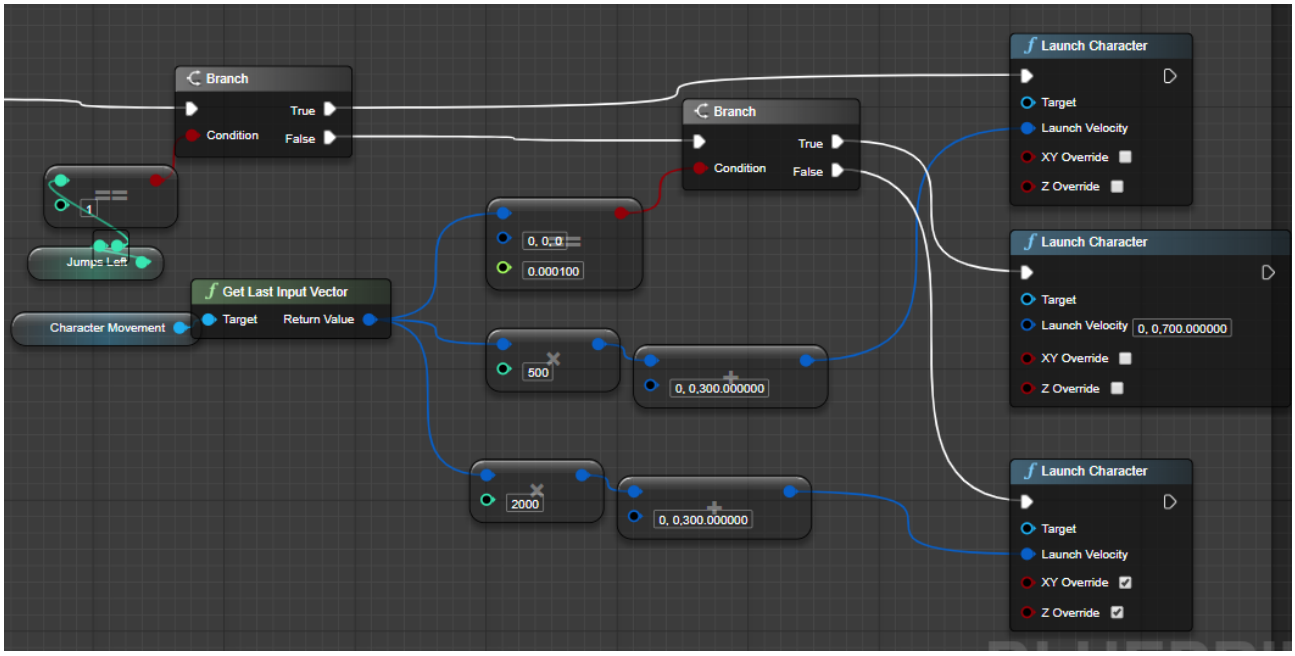

Рисунок Г.6 – механіка стрибку, частина 2

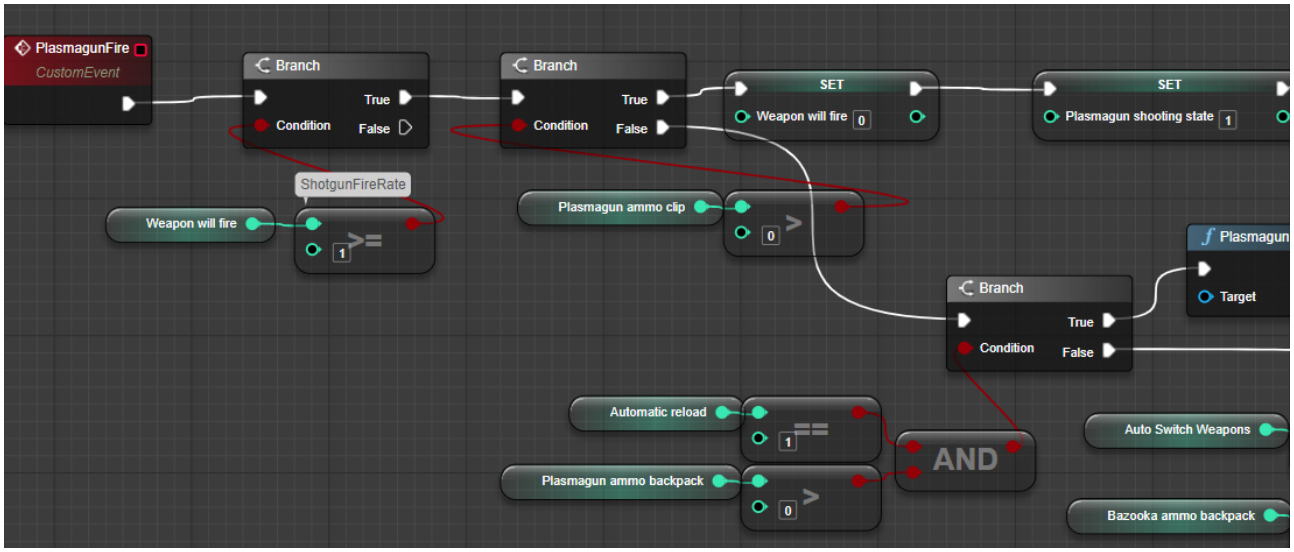

Рисунок Г.7 – механіка стрільби, частина 1

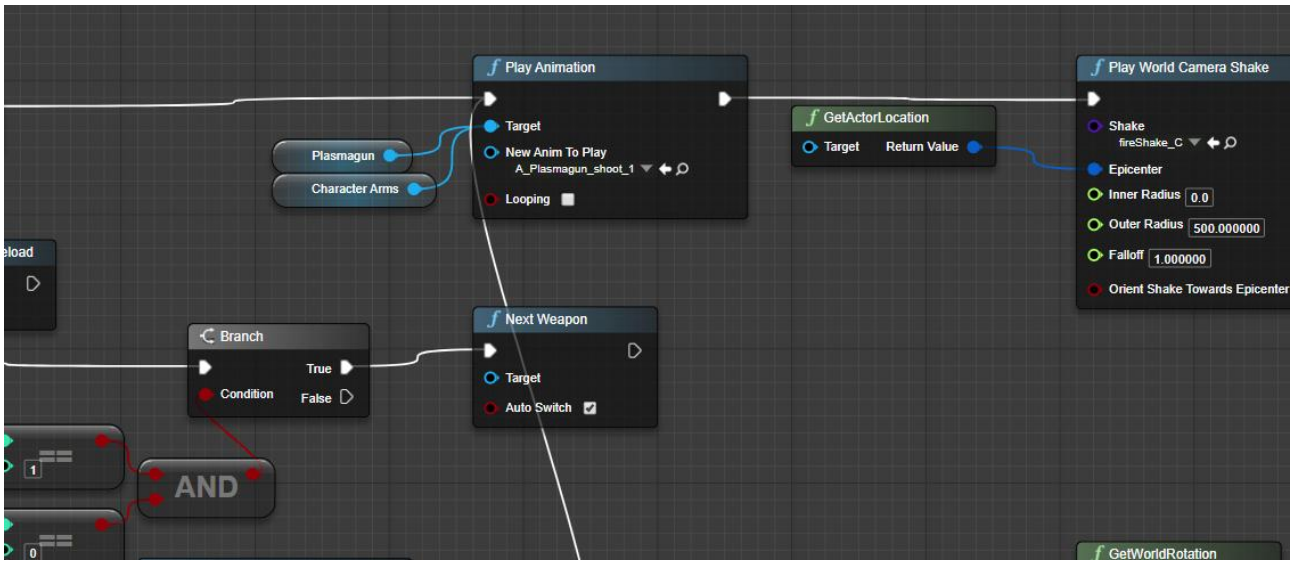

Рисунок Г.8 – механіка стрільби, частина 2

Продовження додатку Г

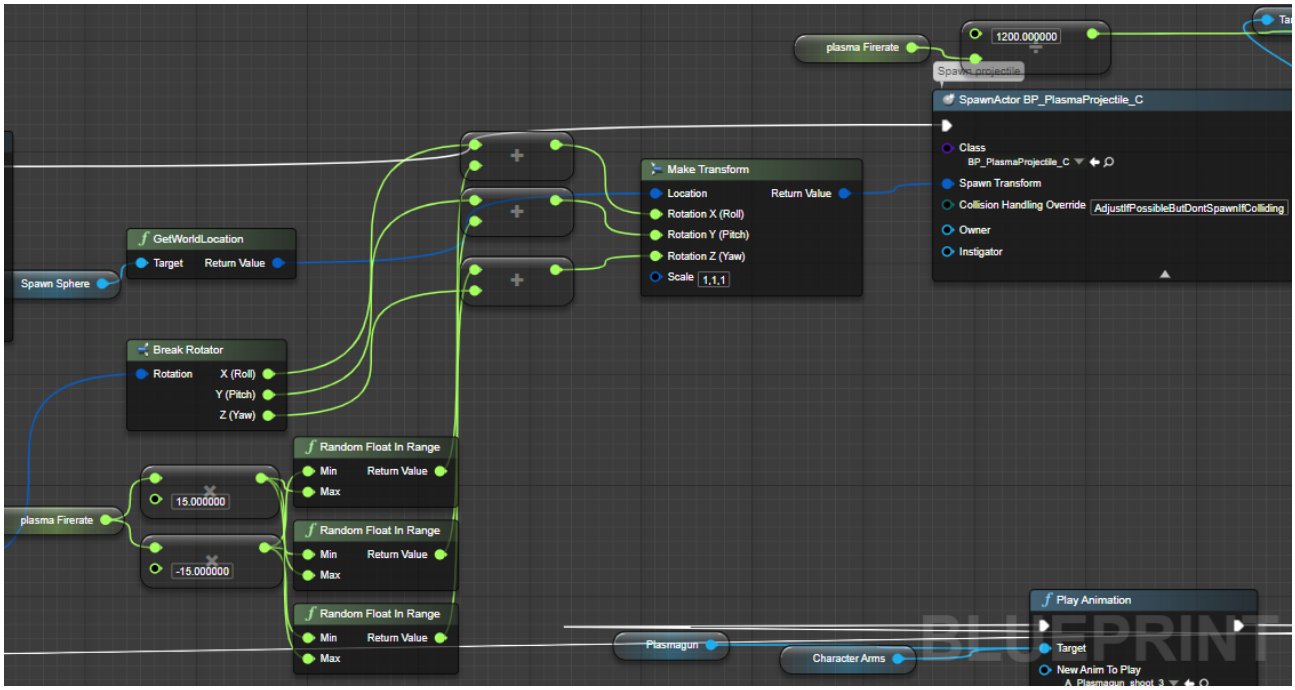

Рисунок Г.9 – механіка стрільби, частина 3

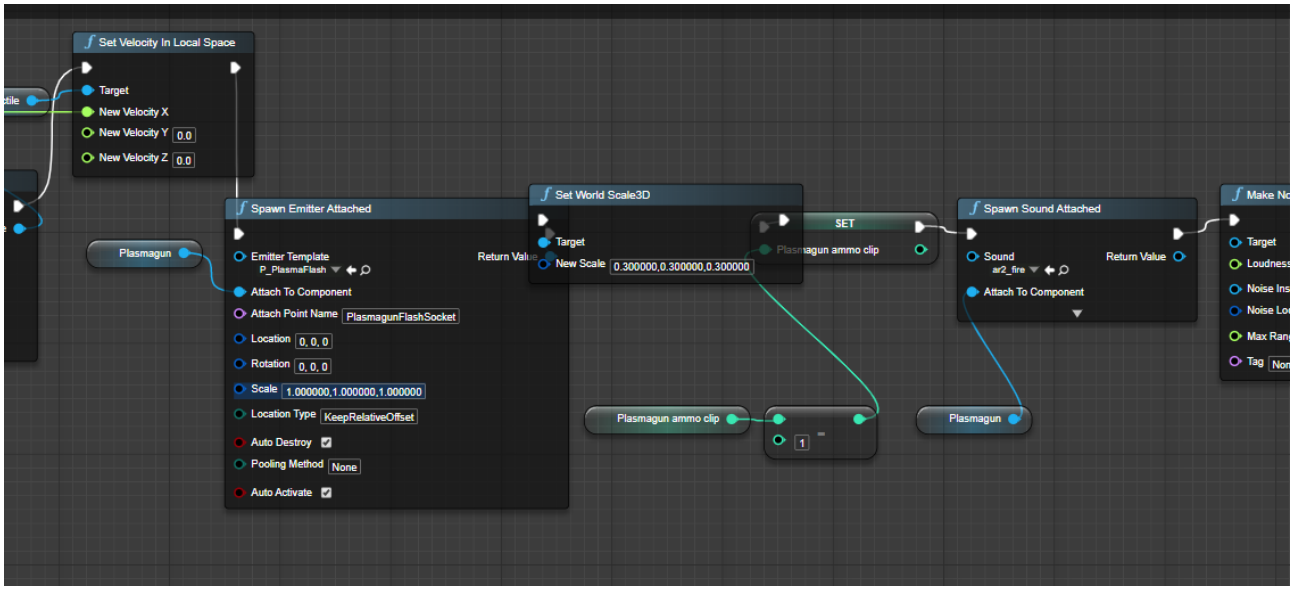

Рисунок Г.10 – механіка стрільби, частина 4

Продовження додатку Г

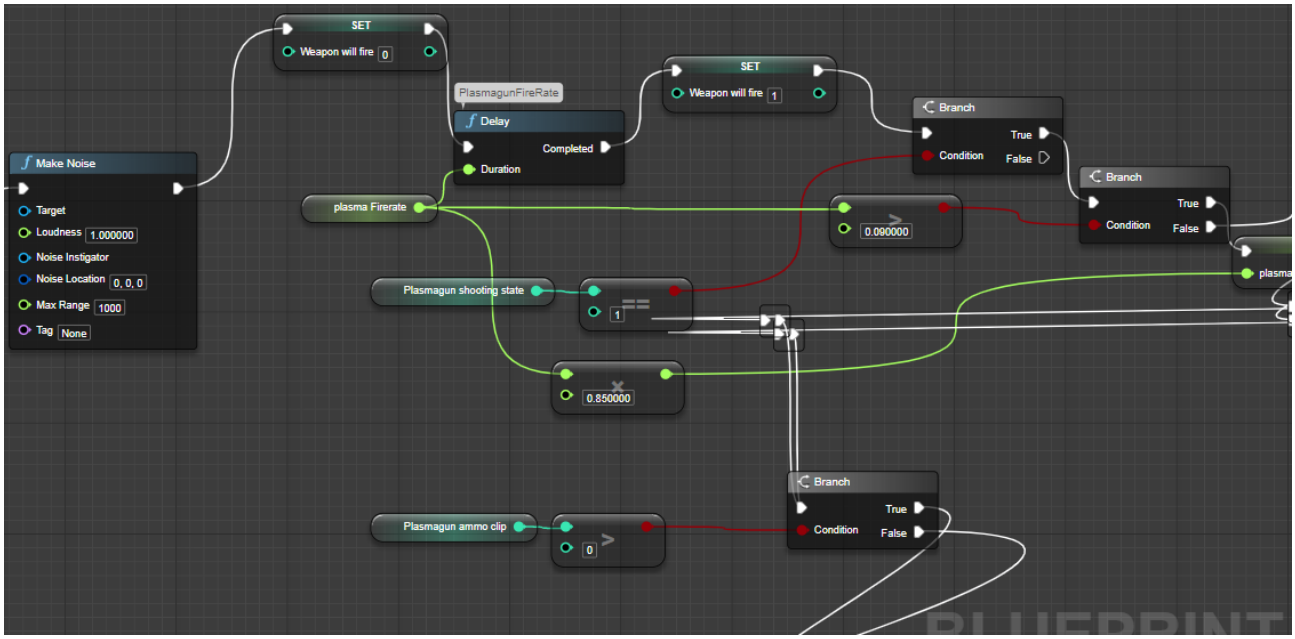

Рисунок Г.11 – механіка стрільби, частина 5

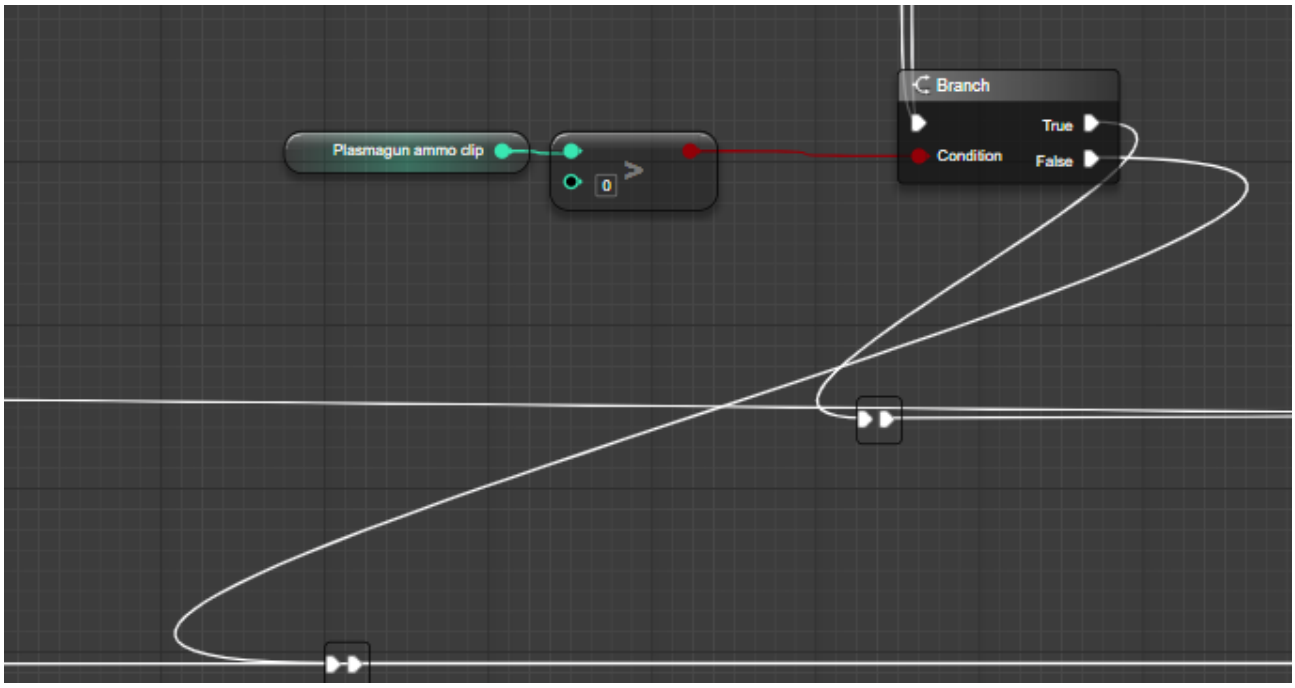

Рисунок Г.12 – механіка стрільби, частина 6

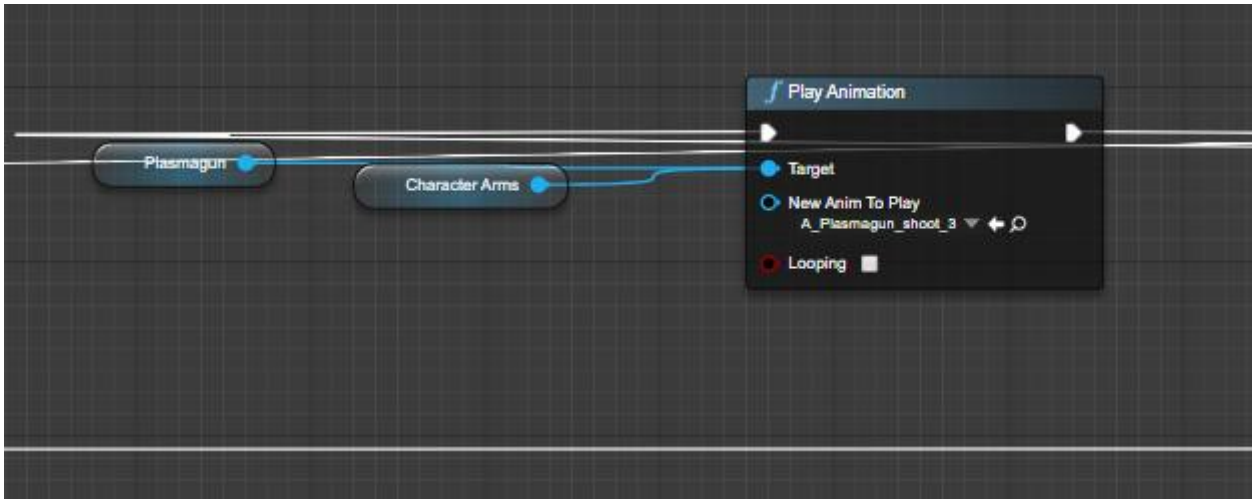

Рисунок Г.13 – механіка стрільби, частина 7

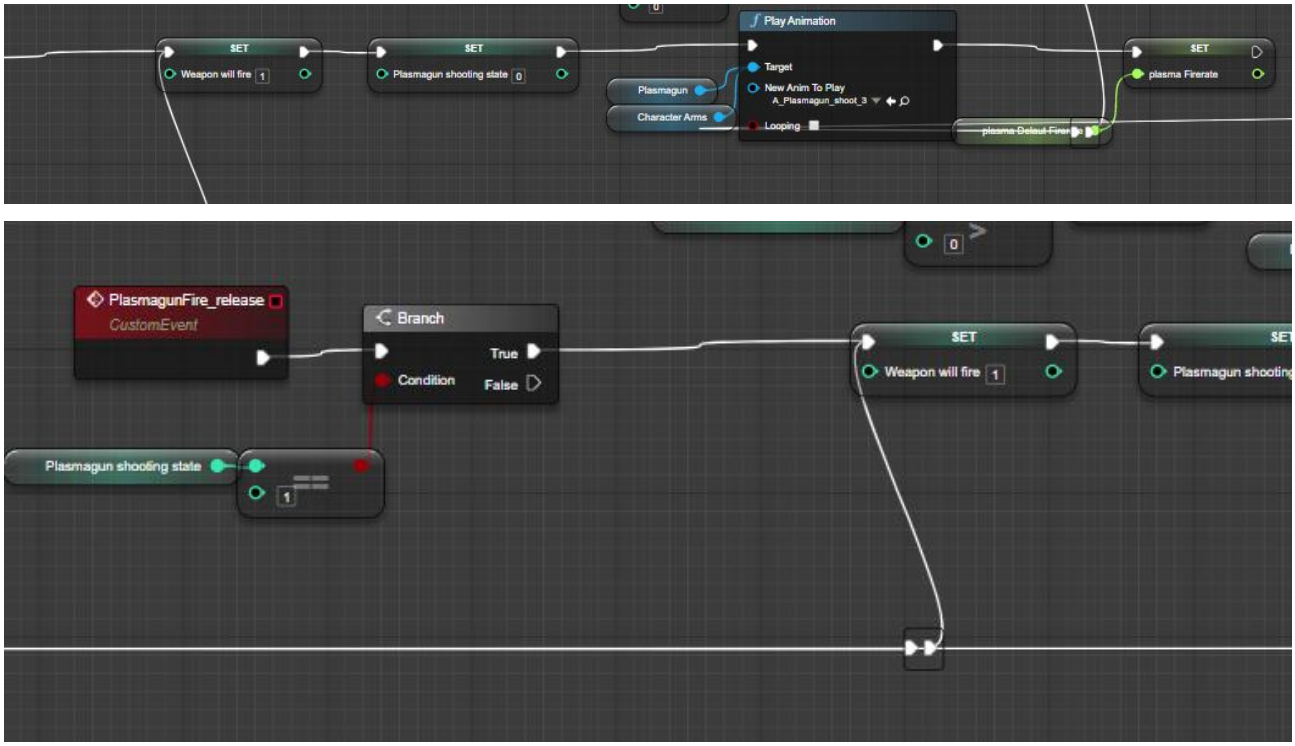

Рисунок Г.12 – механіка стрільби, частина 8

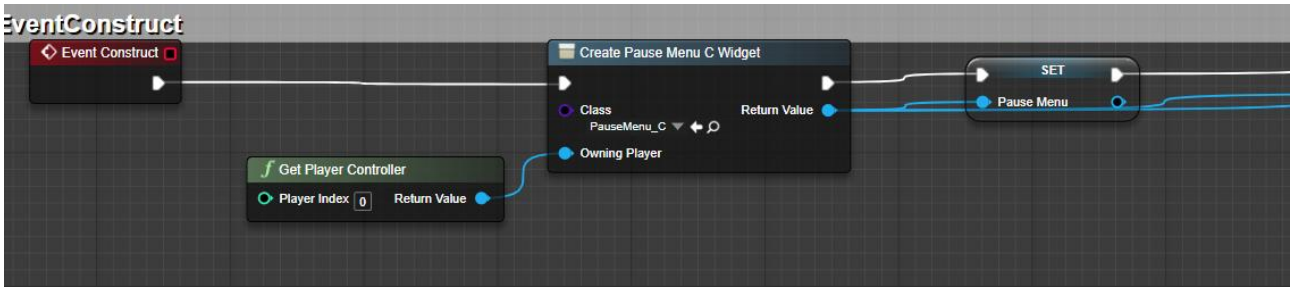

Рисунок Г.13 – механіка меню, частина 1

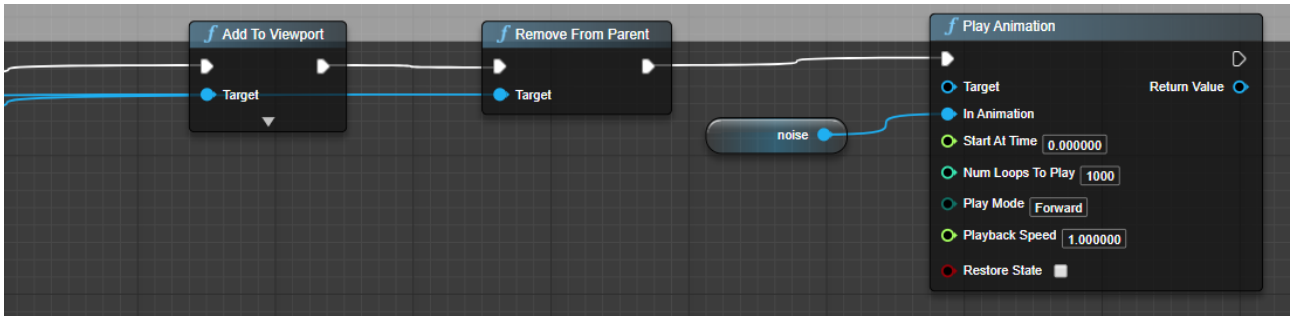

Рисунок Г.14 – механіка меню, частина 2

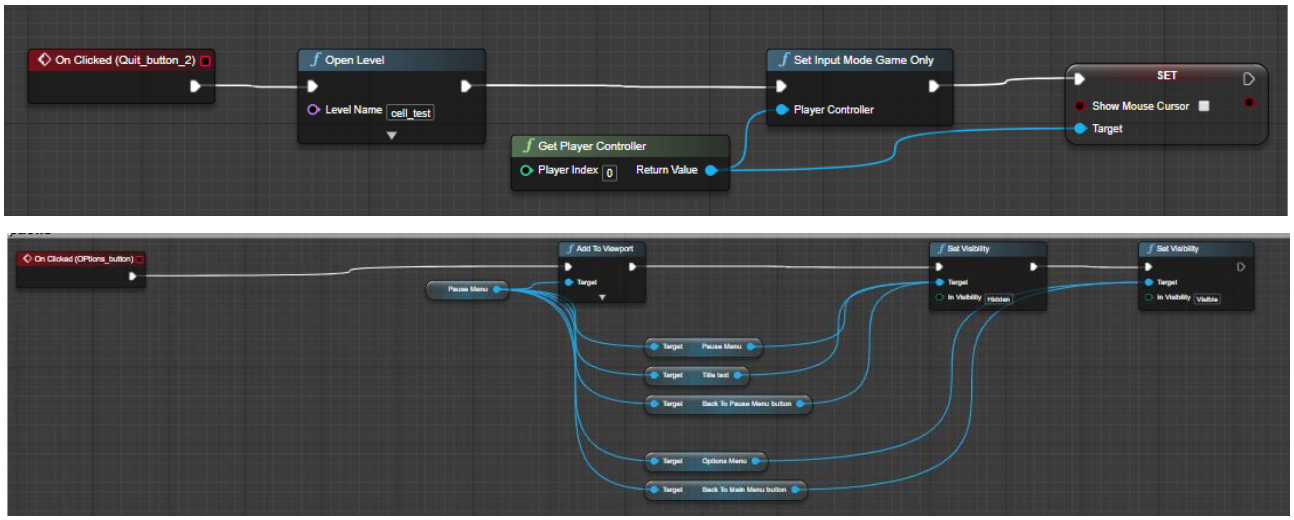

Рисунок Г.15 – механіка меню, частина 3

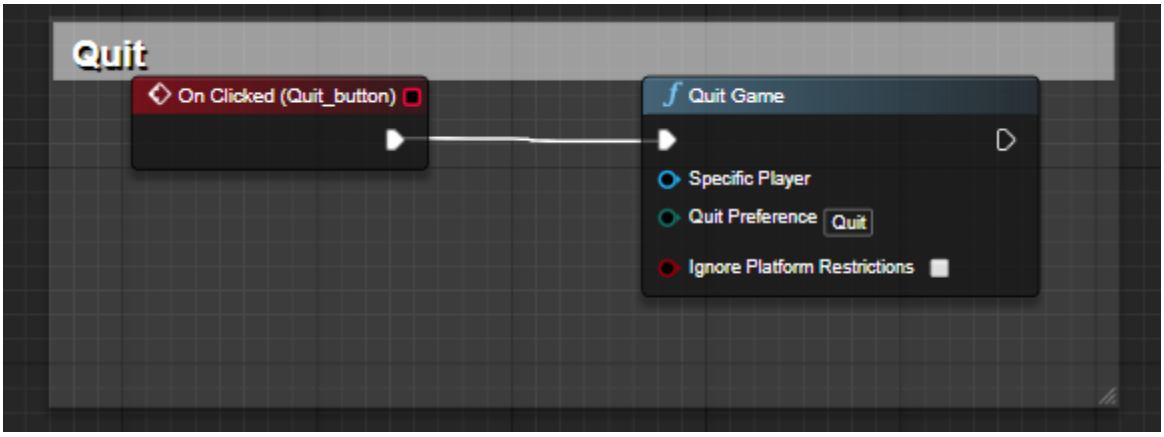

Рисунок Г.16 – механіка меню, частина 4