МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет
Центр заочної, дистанційної та вечірньої форм навчання
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
Ігор ШЕЛЕХОВ
(підпис)
11 грудня 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня магістр

зі спеціальності 122 - Комп'ютерних наук,
освітньо-професійної програми «Інформатика»
на тему: «Інформаційна технологія проектування мережевого ігрового програмного забезпечення»
здобувача групи ІН.мз-21с Сизик Аліні Анатоліївні

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

Аліна СИЗИК
(підпис)

Керівник, Ігор ШЕЛЕХОВ
В.о. завідувача кафедри
кандидат технічних наук/кандидат
(підпис)

Суми – 2023

Сумський державний університет
Центр заочної, дистанційної та вечірньої форм навчання
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН.мз-21с Сизик Аліні Анатоліївні

1. Тема роботи: «Інформаційна технологія проектування мережевого ігрового програмного забезпечення»
затверджую наказом по СумДУ від *«20» листопада 2023 р. № 1308-VI*
2. Термін здачі здобувачем кваліфікаційної роботи *до 14 грудня 2023 року*
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Аналіз та розробка концепції гри. 2) Проєктування ігрової системи. 3) Програмна реалізація ігрової системи. 4) Тестування ігрової системи.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 20 ___ р.

Завдання прийняв до виконання _____ Керівник _____
(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз та розробка концепції гри*	*25.09.2023*	
2	*Проєктування ігрової системи*	29.09.2023	
3	*Програмна реалізація ігрової системи*	25.10.2023	
4	*Тестування ігрової системи*	15.11.2023	
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*	25.11.2023	

Здобувач вищої освіти _____ Керівник _____
(підпис) (підпис)

АНОТАЦІЯ

Записка: 58 стор., 31 рис., 3 додатків, 20 джерел.

Обгрунтування актуальності теми роботи – великий потенціал використання гейміфікаційного підходу у різних сферах професійної діяльності зумовлює актуальність роботи.

Об'єкт дослідження — процеси проєктування сучасної ігрової системи

Предмет дослідження — моделі, методи та засоби інформаційної технології проєктування сучасної ігрової системи.

Мета роботи — дослідження технології проектування мережевого ігрового програмного забезпечення та її використання для розробки мережевого шутера від першої особи

Результати — При написанні роботи були досліджені інформаційні технології проектування мережевого ігрового програмного забезпечення спираючись на які визначені етапи виробництва гри та застосовані для створення мережевого шутера від першої особи: від розробки концепції до поствиробництва, що відповідає сучасній інформаційній технології проектування мережевого ігрового програмного забезпечення.

ІГРОВИЙ РУШІЙ, КЛІЄНТ-СЕРВЕР, КОНЦЕПЦІЯ ГРИ, МЕРЕЖЕВИЙ ШУТЕР, ПОРТРЕТ ГРАВЦЯ,

ЗМІСТ

ВСТУП

Обґрунтування вибору теми роботи, її актуальності.

Розробка комп'ютерних ігор стає все більш прибутковою. Пандемія, спричинила великі збитки багатьом індустріям світу, і вони досі намагаються оговтатися, але сфера ігрової індустрії за цей час зросла і продовжує зростати, її обсяг у світі на даний момент складає 218,7 мільярди доларів, а за прогнозами на 2027 рік має скласти 521 мільярдів доларів. США. Ігрова сфера у порівнянні, з кіно та спортивною індустрією значно більша тому не дивно що багато IT-компаній працюють у цій галузі. Спираючись на цифри можна стверджувати, що ігрова індустрія є одним з рушіїв IT галузі. Також супутній розвиток завдяки їй отримали такі галузі, як кіберспорт та ринок віртуальних речей(блокчейн). Блокчейн в ігровій індустрії дає геймерам змогу торгувати ігровими активами на блокчейн-біржах, таких як Bazaar Marketplace та Binance NFT market. Прогнозується, що до 2030 року обсяг світового ринку блокчейну в іграх досягне приблизно 301,53 млрд дол. США. Кількість глядачів під час трансляції кіберспортивного чемпіонату світу з League of Legends, співставна з аудиторією глядачів чемпіонату світу по футболу.

Виходячи з вищенаведених цифр сміливо можна стверджувати, що ігрова індустрія є рушієм IT галузі.

Визначення предмета та об'єкта дослідження.

Мета і завдання дослідження відповідно до предмета та об'єкта дослідження.

Мета роботи – дослідження інформаційних технологій проектування мережевого ігрового програмного забезпечення. На основі досліджених технологій створити мережеву гру.

Основні задачі, які потрібно виконати у ході виконання роботи:

– провести аналіз та розробити концепцію гри;

– провести підготовку у виробництві гри;

– здійснити розробку та скомпілювати робочу модель гри;

– отримавши зворотній зв’язок від гравців, виправити можливі помилки та випустити новий реліз.

Методи дослідження.

Методи проєктування клієнт-серверних додатків, методи графічного проєктування, методи забезпечення мережевої взаємодії, методи тестування програмного забезпечення

Наукова новизна отриманих результатів.

В роботі запропоновано комплекс інформаційних, алгоритмічних і програмних засобів інформаційної технології проєктування мережевого ігрового програмного забезпечення та застосуванню їх для створення мережевої гри.

Структура та обсяг роботи.

Пояснювальна записка складається зі вступу, 4 розділів, висновків, списку використаних джерел. Загальний обсяг роботи − 73 сторінки, у тому числі 70 сторінок основного тексту, 2 сторінки списку використаних джерел з 18 пунктів, 59 рисунків.

1 ПРОВЕДЕННЯ АНАЛІЗУ ТА РОЗРОБКА КОНЦЕПЦІЇ ГРИ

Розробки концепції, складається з:

- визначення цільової аудиторії,
- визначення жанру,
- створення приблизного сюжету,

1.1 Цільова аудиторія

Визначення цільової аудиторії складається з наступних етапів:

1) Визначення статистики популярних продажів
2) Формування портрета гравця

1.1.1 Визначення популярних продажів на основі статистичних даних:

Світові тенденції ігрової індустрії:

Google і Newzoo постійно проводять дослідження відносно розвитку ігрової індустрії. Ці дослідження проводяться для чотирьох регіонів та 16 країн, серед яких є Україна.

Рисунок 1.1 – Результати аналізу світового ринку ігор

При аналізі діаграми (Рис. 1.1) видно, що у 2020 році світовий ринок ігор зріс на 23,1%. За останнє десятиліття це найвищий показник. Також легко

помітити зростання світової галузі розробки ігор і до 2024 року вона досягне 218,7 мільярди доларів.

Згідно дослідженням Google Рисунок відсоткове розподілення гравців по регіонам виглядає наступним чином:

Європа – 14%,

Близький Схід та Африка — 15%,

Латинська Америка — 10%,

Північна Америка — 7%.

Азійсько-Тихоокеанський регіон — 54%.

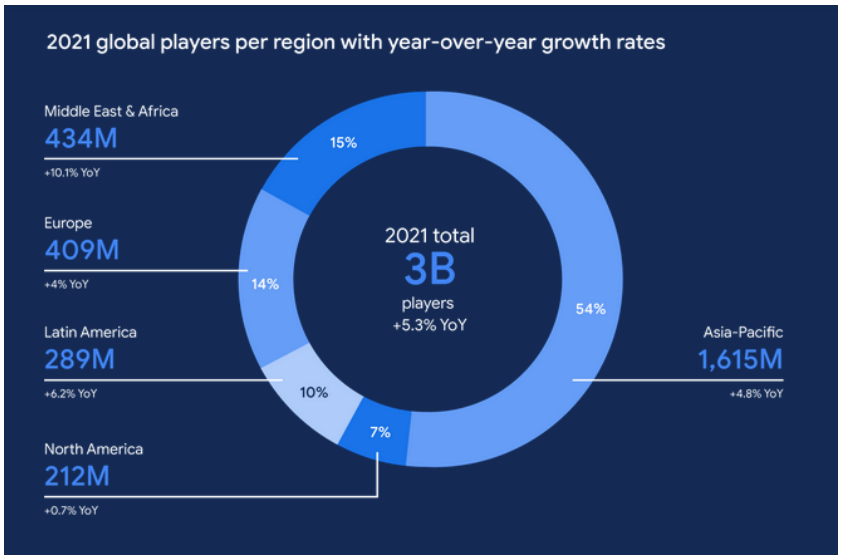

Рисунок 1.2 - Розподілення гравців по регіонам

Нові тенденції на світовому ігровому ринку:

- Поява ігор як соціальних платформ та метавсесвіту.
- Готовність масштабної аудиторії платити за підписки на ігри.
- Популярність стримінгу як форми соціального залучення.
- Перехід до мультиплатформенності, коли контент не пов’язаний з однією платформою.
- Швидше впровадження хмар.
- Зменшення витрат на ПК й консолі на багатьох ринках.
- Затримки в розробці, що вплинули на рівень пропозиції ігор.

Враховуючи те, що тенденції ігрових ринків – світового та українського в багатьох позиціях збігаються, то можна створити гру для українського ігрового ринку, який є набагато зрозумілішим з точки зору розробника для запуску стартового проекту. У подальшому гру потрібно вдосконалити і адаптувати або для масового Азійсько-Тихоокеанський регіону, або не таких масових але багатших ринків Європи та Північної Америки.

1.1.2 Формування портрета гравця:

При формуванні портрету гравця дається відповіді на питання про: локацію, стать, вік, соціальний статус, явні або не явні «болі», платоспроможність.

Ігровий ринок в Україні

Статистичні дані по українському ринку ігор проводилися до повномасштабного вторгнення в Україну. На даний момент ігровий ринок в Україні значно просів.

NielsenIQ на замовлення Wargaming до повномасштабного вторгнення проводила в Україні дослідження гравців відео/комп'ютерних ігор. На основі дослідження можна скласти портрет середньостатистичного гравця в Україні

Діаграма першої п’ятірки найпопулярніших ігор в Україні (рис. 3.1)Рисунок має наступний вигляд:

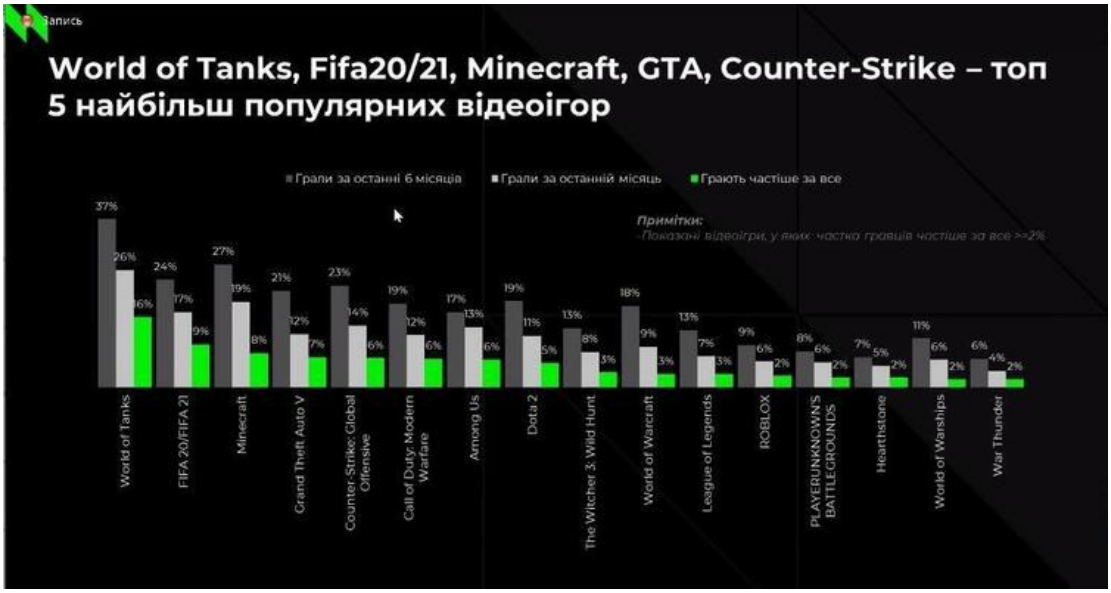

Рисунок 1.3 - Найпопулярніші ігри в Україні

Найбільш поширеними іграми серед мешканців України є World of Tanks, Fifa20/21, Minecraft, GTA, Counter-Strike.

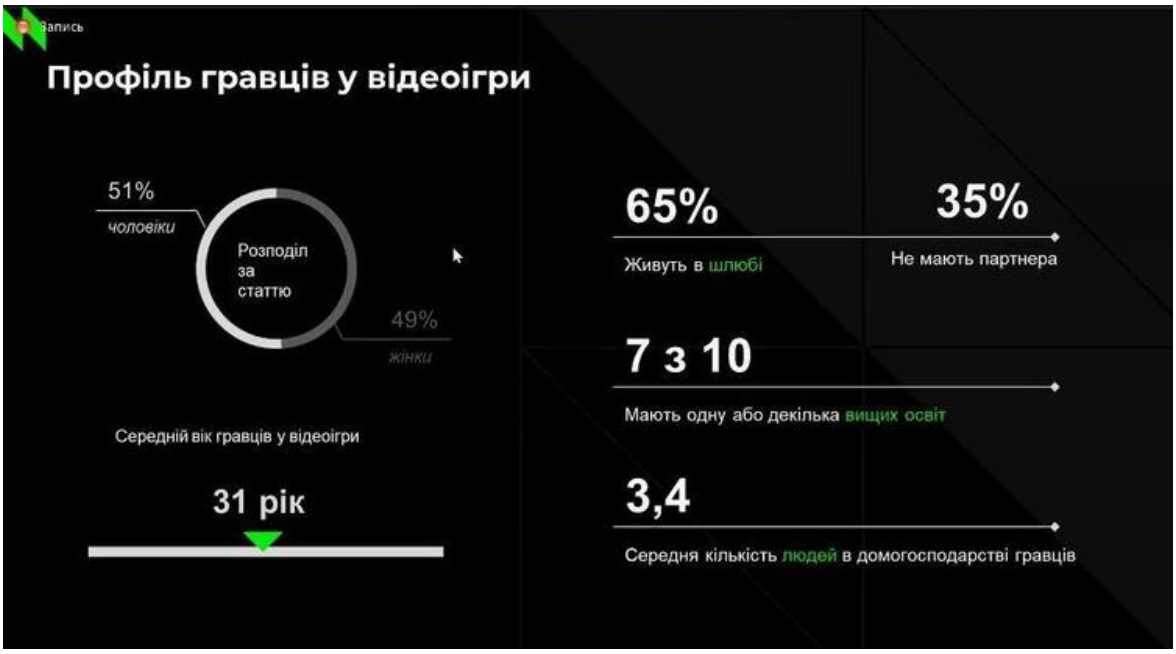

Рисунок 1.4 – Характеристики гравців

З діаграми (рис. 1.4) видно, що середній вік гравця в Україні 31 рік і 70% з них мають вищу освіту і більшість живуть у шлюбі.

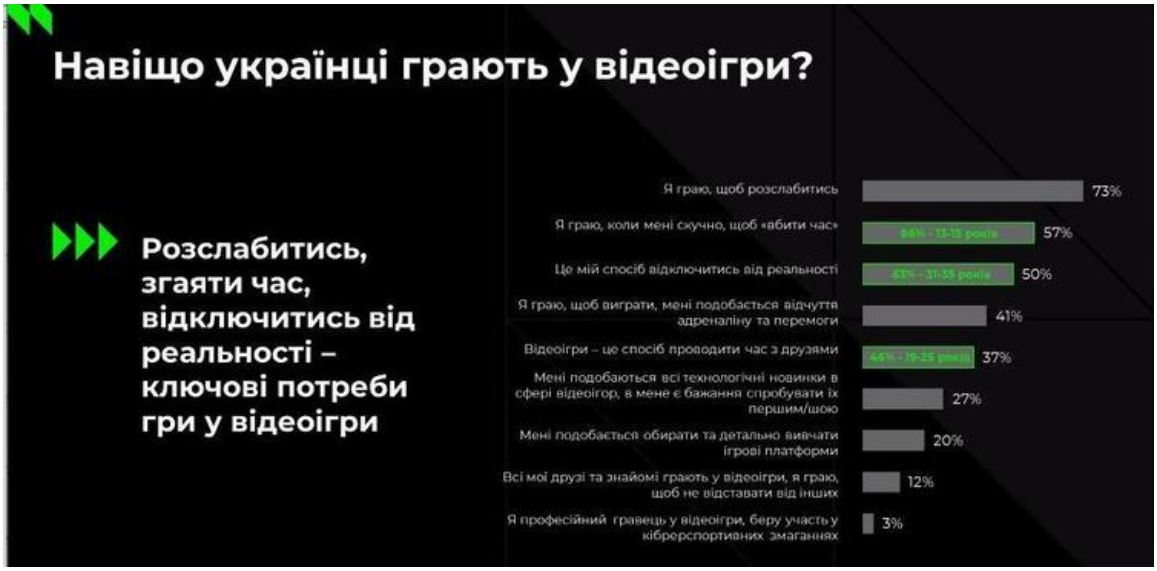

Рисунок 1.5 – Ключові потреби гри у відеоігри

Якщо взяти більшість, а це люди віком 31 рік то вони «занурюються» в ігри, щоб відключитися від реальності (рис. 1.5).

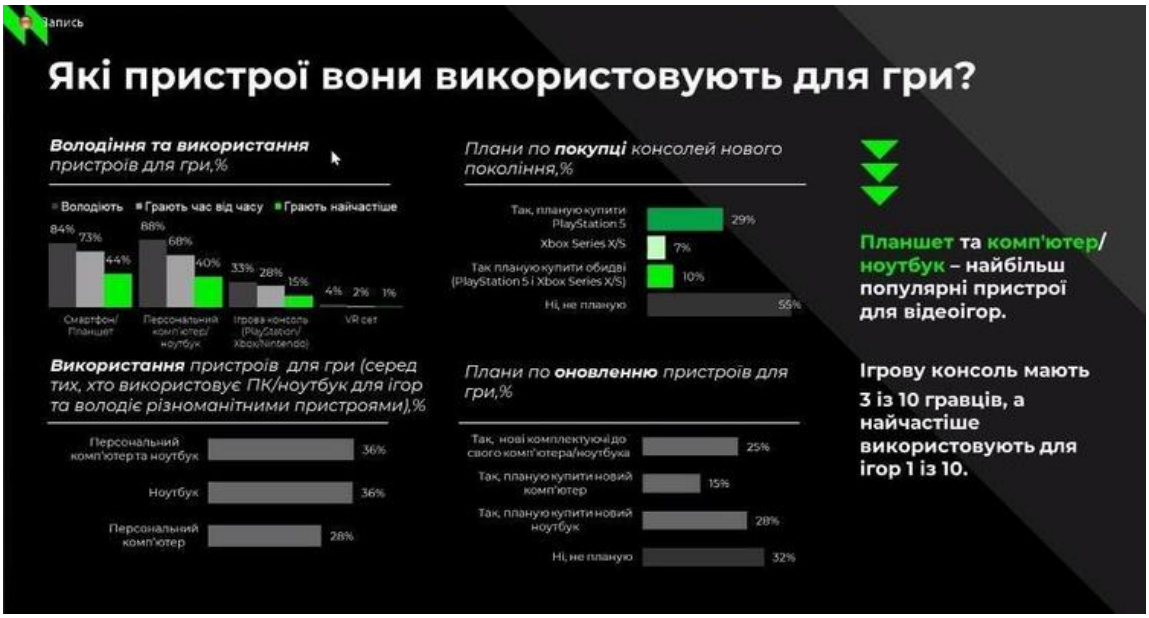

Рисунок 1.6 – Аналіз ігрових пристроїв

Три з десяти респондентів мають ігрові консолі (PlayStation і Xbox), а для гри використовують тільки один. Більшість грає на планшетах та комп’ютерах/ноутбуках (рис. 1.6).

Рисунок 1.7 – Аналіз часу, що витрачається на ігри

Близько 1-3 у будні та 6 годин у вихідні українці можуть «присвятити» іграм. Це досить великий показник, якщо врахувати те, що більшість респондентів працюють і мають сім'ї (Рис. 1.7).

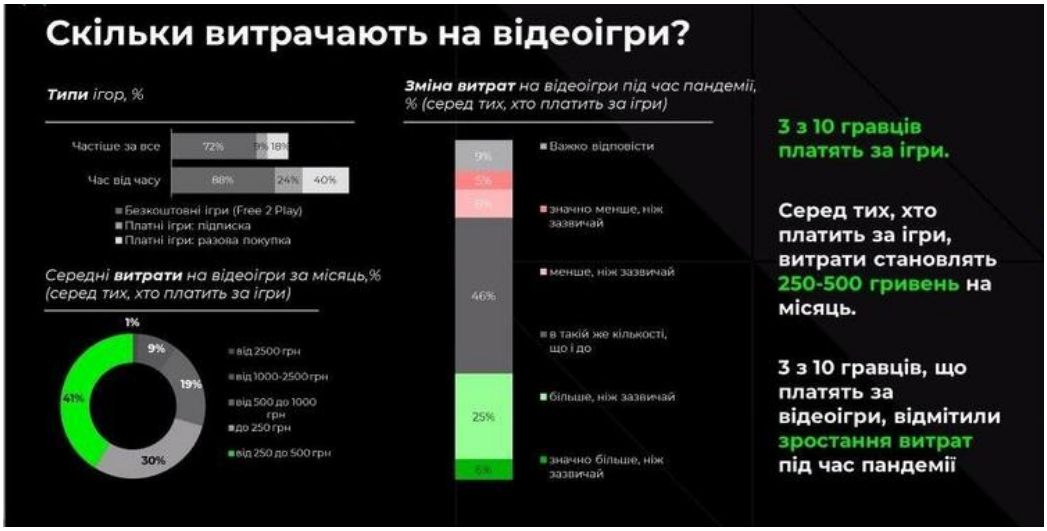

Рисунок 1.8 – Аналіз фінансових витрат на ігри

Від 2500 грн щомісячно витрачають на ігри 9% гравців, вони або оплачують підписку, або разово купують продукт. 46% гравців, користуються платними сервісами, 31% став платити більше, а 13% – менше. Більшість опитаних (близько 73%) використовують безкоштовні ігри (Рис. 1.8).

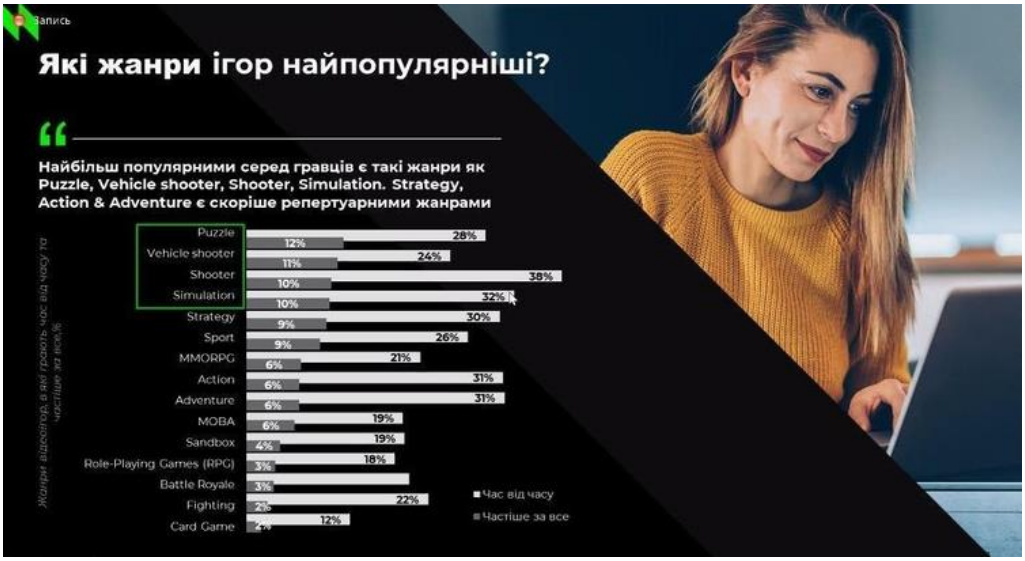

Рисунок 1.9 – Аналіз популярності ігрових жанрів

Найчастіше українці обирають ігри таких жанрів як: Puzzle, Vehicle shooter, Shooter, Simulation і Strategy (рис. 1.9)

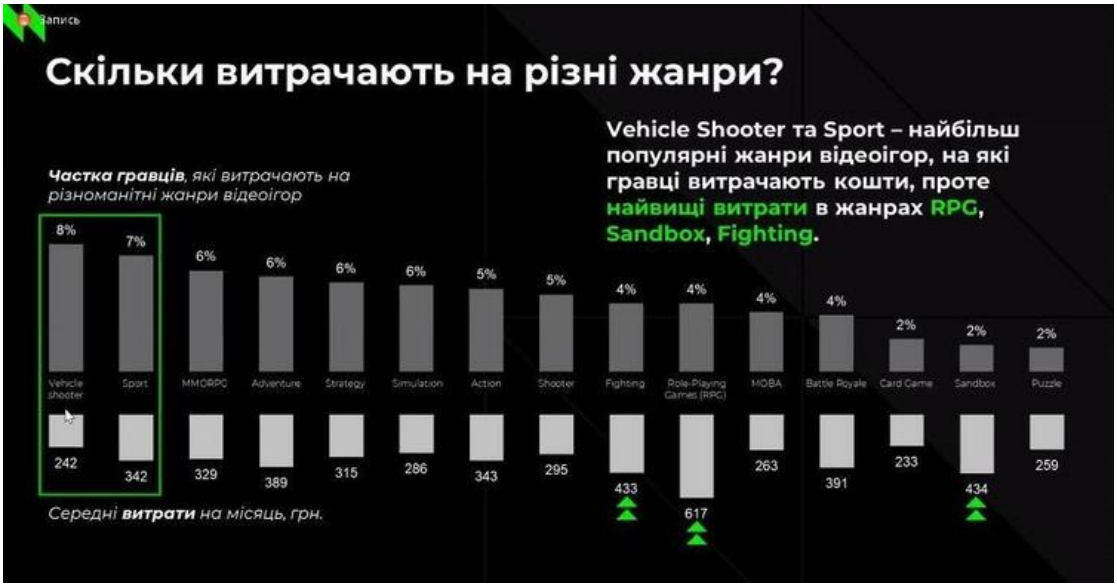

Рисунок 1.10 - Аналіз фінансових витрат на ігри певного жанру

Найбільш популярні жанри ігор на які витрачають кошти за місяць – це Vehicle shooter (242грн) та Sport(342 грн). Проте RPG найдорожчий, але не масовий «сегмент» на який гравці щомісяця витрачають 617 грн Рисунок .

Відповідно до цього визначимо «портрет» гравця цього жанру. Він має наступні особливості: чоловік 40+років, одружений і має дітей, з вищою освітою, який грає у гру 8 годин на тиждень, на персональному комп’ютері або ноутбуці, щоб відключитися від реальності і готовий витратити на гру у місяць близько 242 грн.

1.2 Жанр комп’ютерної гри:

Для визначення жанру гри потрібно орієнтуватися на очікування гравця від комп’ютерної гри, і відповідно до цього очікування умовно можна розділити на три основні групи:

- Терпіння – скільки гравець готовий витратити часу на гру для досягнення мети.

- Реакція – наскільки точно та швидко гравець здатен реагувати на зміну подій у грі.
- Мислення – наскільки правильно або винахідливо гравець використовує ігрові механіки.

Спираючись на вищенаведені основні групи можна попередньо обрати жанр гри. Для успішного просування гри на ринку ігрової індустрії, необхідно спиратися на статистичні дослідження, що визначають цільову аудиторію. Взявши до уваги статистичні дані і визначивши цільову аудиторію потрібно більш конкретно визначитися з жанром, а саме: ігри-шутери, платформери, економічні та воєнні стратегії, моба, королівська битва, симулятори, тайм-кілери, текстові ігри, ігри з картами.

Спираючись на результати статистики та «портрет» гравця сміливо можна обрати жанр мережевої гри – це мережева комп'ютерна гра від першої особи у жанрі шутер.

1.3 Сюжет комп'ютерної гри

Сюжет комп'ютерної гри потребує ідеї, яка може черпати натхнення з різних джерел:

1. Власна оригінальна ідея, що за популярністю використання займає по праву перше місце. Використання власної ідеї розповсюджене серед невеличких фірм та компаній, які не готові виділити додаткові кошти на придбання ліцензії правовласника.

2. Ідея що спирається на сюжет книги , фільму, життя конкретних людей і персонажів і потребує ліцензії. Якщо у основу сюжету покладено ідею, що потребує ліцензії то потрібен дозвіл правовласника. Такий вид використання ідеї притаманна крупним компаніям, які вкладають у виробництво та рекламу ігрової продукції значні кошти, і професійно створений сюжет відіграє дуже важливу роль.

Тому в основу гри покладено власна оригінальна ідея. Розроблені власні: головний персонаж, віртуальний світ та його наповнення тривимірними

моделями. Використані файли із звуковим супроводженням, що не потребують ліцензії і є у вільному доступі мережі інтернет з так званою ліцензією Free.

Shooter відноситься до групи ігор на реакцію із високою динамікою та щільністю подій. Виходячи з цього можна визначитись з сюжетом гри.

Сюжет гри полягає у тому щоб знищити супротивника і тим самим захопити домінування на певній локації мапи.

1.4 Концепція гри

Ігровий ринок – Український з подальшою адаптацією для Європейського ринку.

Цільова аудиторія – старша аудиторія із середнім бюджетом.

Жанр – мережева комп'ютерна гра від першої особи у жанрі шутер.

Сюжет – використання власного сюжету, як бюджетного варіанту.

1.5 Постановка задачі

Кваліфікаційна робота магістра присвячена дослідженню технології проектування мережевого ігрового програмного забезпечення та її використання для розробки мережевого шутера від першої особи. В ході проведеного огляду досліджено інформаційні технології проектування мережевого ігрового програмного забезпечення спираючись на які визначено етапи виробництва гри та застосовані для створення мережевого шутера від першої особи: від розробки концепції до поствиробництва, що відповідає сучасній інформаційній технології проектування мережевого ігрового програмного забезпечення.

Для досягнення поставленої мети необхідно виконати такі завадння:

1. Провести дослідження в області сучасних інформаційних технологій проектування мережевого ігрового програмного забезпечення. На основі аналізу визначені етапи реалізації гри.
2. Провести аналіз ігрового ринку та розроблена концепція гри.

3. Розробити дизайн головного персонажу та інших об’єктів гри на основі якого створити тривимірні моделі, які необхідно текстурувати, анімувати та виконати експорт для подальшого використання у грі.

4. Створити комп'ютерну гру у жанрі мережевого шутера від першої особи, в якій реалізувати: мережеве під’єднання гравців, розробити ігрові механіки для мережевої гри, провести масштабування гри з використанням фінальних тривимірних моделей, анімацій і звуків.

5. Для технічної реалізації проекту обрати ігровий рушій, а для здійснення дизайнерського задуму - тривимірний редактор.

6. Виконати компіляцію комп'ютерної гри у жанрі мережевого шутера від першої особи, та представити у вигляді .exe файлу з набором необхідних динамічних бібліотек.

7. Протестувати та перевірити на працездатність ігрову систему.

2 ПРОЄКТУВАННЯ ІГРОВОЇ СИСТЕМИ

Розроблення гри здійснюється за принципом step by step. Такий підхід обумовлений відсутністю команди. При наявності команди розробників можна значно розпаралелити процес, що позитивно вплине на час виготовлення та реакцію на виправлення певних недоліків. Підготовку у виробництві можна умовно розділити на пункти:

1) Обрання і встановлення необхідних застосунків,
2) створення дизайну рівня у грі,
3) дизайн моделей у грі.

2.1 Обрання необхідних застосунків

Практичне створення гри умовно можна розділити на дві частини:

1) Технічна складова - робота в ігровому рушії.
2) Арт-дизайнерська складова – робота в 3D – редакторі.

Технічна частина гри виконується завдяки ігровому рушію, який розробляється самостійно або використовується вже готовий. Ігровий рушій – це програмний застосунок, що дозволяє виводити графіку і звук, прораховувати фізику віртуального світу, контролювати логіку NPC (non-player character) або AI (artificial intelligence) ігрових персонажів.

В якості ігрового рушія обрано Unity компанії Unity Technologies. Одним із ключових факторів надання переваги Unity є його доступність, простота використання, велика спільнота розробників, велика кількість матеріалів для навчання. Додаткові переваги Unity:

- Безкоштовність: компанія Unity Technologies пропонує безкоштовну версію ігрового рушія з основними функціями, які дозволяють навчатися та створювати ігри. Всього існує три рівні підписки: Pro, Plus і Free. Безкоштовний рівень підписки(Free) надає доступ до основного двигуна і базових функцій редактора Unity.

- Гнучкість: програма-редактор Unity працює на операційних системах Windows, macOS і Linux, а сам рушій може бути запущений на 25 платформах.
- Візуальне середовище розробки: включає не тільки інструментарій візуального моделювання, а й інтегроване середовище в якому дуже зручно проводити тестування.
- Модульна система компонентів: за допомогою якої відбувається конструювання ігрових об'єктів, які можуть бути комбінованими пакетами функціональних елементів. На відміну від механізмів успадкування, об'єкти в Unity створюються за допомогою об'єднання функціональних блоків, а не включення у вузли дерева успадкування. Такий підхід полегшує створення прототипів, що є актуальним при розробці ігор.

Арт-дизайн гри виконується завдяки 3D редактору. За допомогою нього створюються тривимірних моделі для віртуального світу гри, які у подальшому текстуруються та анімуються.

В якості 3D редактора обрано 3DS Max компанії Autodesk. Одним із ключових факторів надання переваги 3DS Max – його універсальність, а саме простота у створенні великого спектру тривимірних об’єктів: архітектурні, інженерні, та об’єкти органічного походження. Висока сумісність формату файлів 3DS Max з ігровими рушіями, у зв’язку з чим не потрібно експортувати об’єкти у сторонні формати. Додаткові переваги 3DS Max:

- Функціональність: 3DS Max має широкий набір інструментів для моделювання, текстурування, освітлення, анімації та рендерингу. Він надає можливість створювати складні 3D-сцени з високою деталізацією. Завдяки відкритому коду, під 3DS Max розроблена величезна кількість плагінів, розширень і іншого додаткового софту, які суттєво розширюють функціональність програми.
- Гнучкість: 3DS Max підтримує різні формати файлів, що дозволяє працювати з файлами інших 3D-програм. Він також має велику кількість

плагінів і додатків, які розширюють його можливості підтримки інших форматів.

- Популярність: 3DS Max одним із перших з’явився на ринку програмних продуктів для 3D-моделювання. Перша версія програми під назвою 3D Studio DOS була випущена у 1990 році, тому 3DS Max має велику спільноту прихильників, і безліч відеоуроків, статей, підручників у мережі Internet які знаходяться у вільному доступі.
- Компанія Autodesk надає останню безкоштовну(Education) версію 3DS Max студентам для навчання, ця версія основним функціоналом мало чим відрізняється від версії Pro.

2.2 Дизайн рівня у грі

Дизайн рівнів впливає на геймплей та загальну ідею гри.

Дизайну рівнів складається з: функціональності, орієнтованості, планування та створення ігрового простору, де:

- Функціональність – зручність та доцільності використання простору рівня, в якому кожна річ має певну функцію.
- Орієнтованість – створити рівень таким чином, щоб гра була комфортною та веселою(що важливо для гри у жанрі shooter).
- Планування – спланувати роботу таким чином щоб основна її частина відбувалася ще до відкриття редактора рівня, завчасно скласти сцену де кожна деталь ігрового рівня матиме свою функцію.
- Створення – формування ігрового простору рівня у 3D з використанням обраних програмних застосунків таким чином, щоб забезпечити сценарні події на рівні.

Проаналізувавши інші ігри у жанрі shooter можна відмітити те, що їм притаманні мапи рівнів з наступними властивостями:

- Мапа рівня зазвичай середнього розміру, щоб до зіткнення суперників не проходило багато часу.

- Мапа рівня повинна мати укриття і схованки, де можна убезпечитися або замаскуватися.
- Мапа рівня повинна мати тактично важливі сектори навколо яких відбуваються зіткнення.

Взявши до уваги вищенаведені дослідження, сцени у грі(їх ще називають рівнями гри) орієнтуватимуться на динамічний геймплей із тактично важливими секторами. Схема рівня створюється за допомогою олівця і аркушу паперу. Спочатку вимальовується схематично мапа рівня (рис. 2.1) , потім більш детально малюються об’єкти у сцені. Реалізація мапи рівня буде здійснена у Unity.

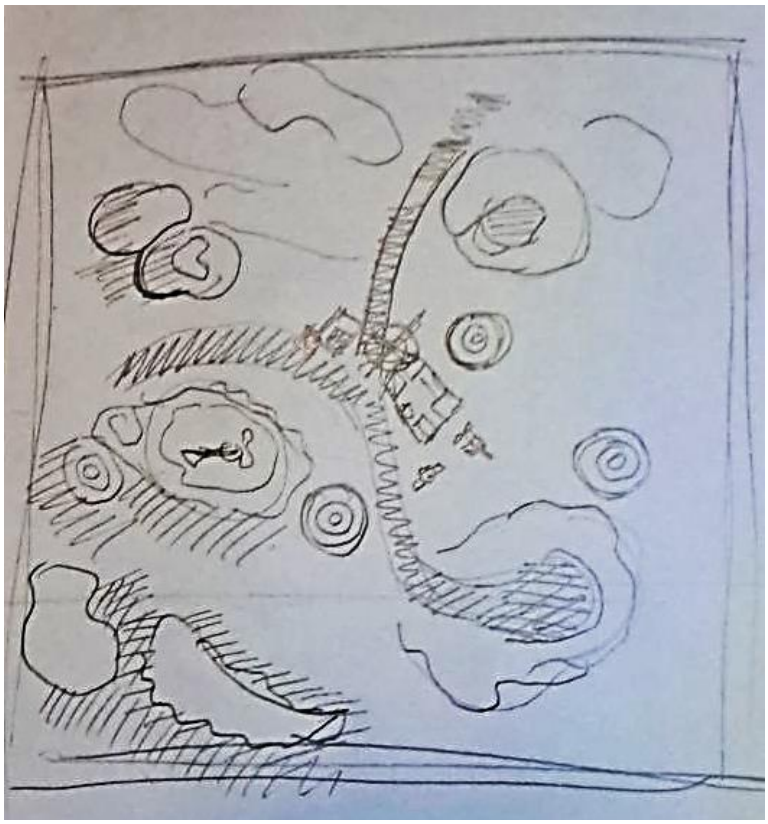

Рисунок 2.1 - Схематична мапа рівня

Створення художнього проекту моделі, тобто її ескізу – це ескізне проектування. Графічне зображення предмета, виконане від руки на аркуші паперу, або при наявності планшету у графічному редакторі.

2.3 Дизайн моделей у грі

2.3.1 Перший етап графічного проектування(пошуку образів та форм).

Цей етап відноситься до так званого пошуку образів та форм. Спираючись на технічне завдання, художник шукає оригінальну форму для персонажу та інших моделей у грі. Фінальний ескіз виконується окремо на аркуші для кожної моделі, а також груп моделей для кращого розуміння того, як вони будуть поєднуватися у сцені (рис. 2.2).

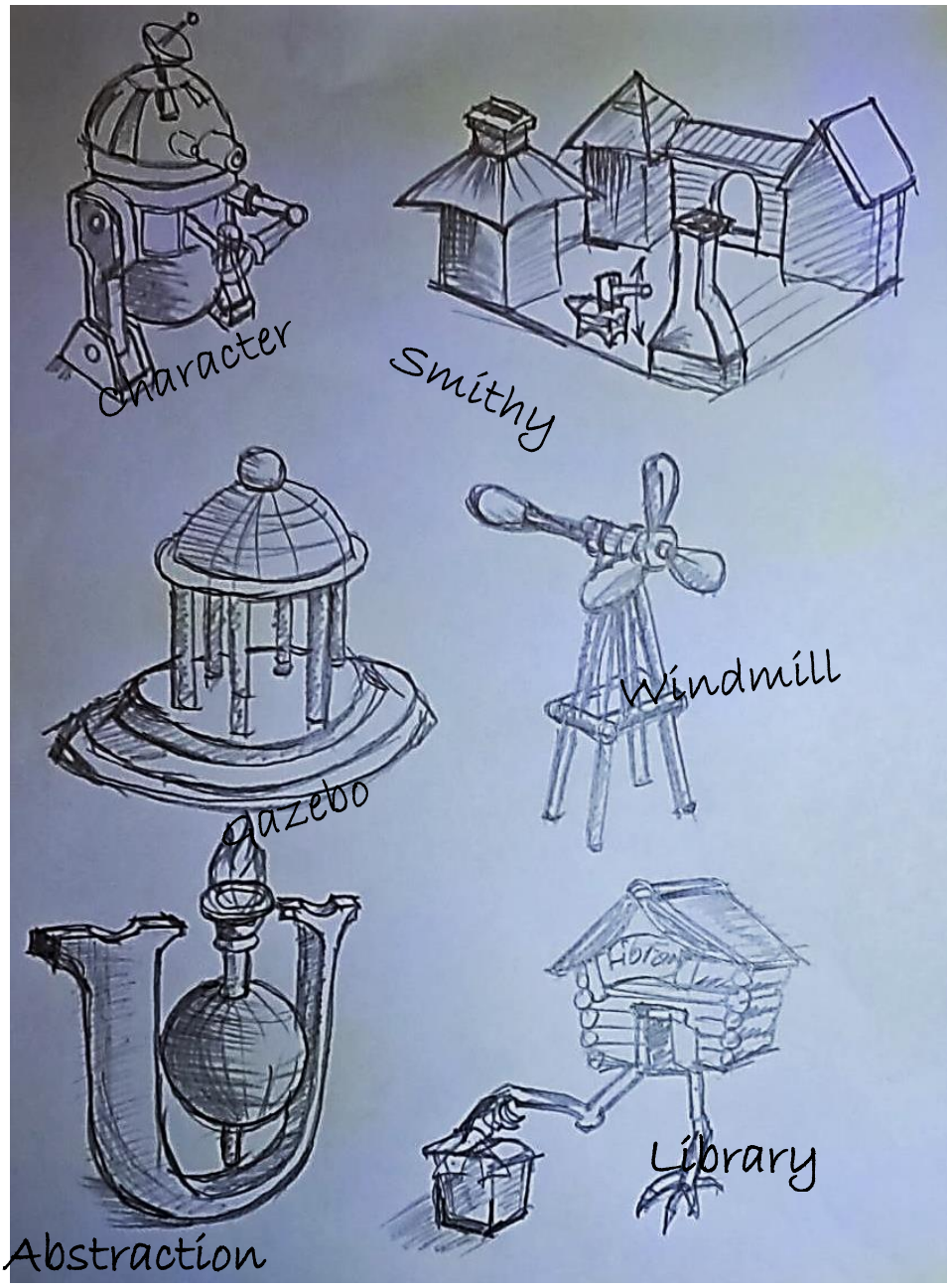

Рисунок 2.2 - Фінальний ескіз

2.3.2 Другий етап графічного проектування(створення 2D-моделей).

На цьому етапі художник деталізує відібране зображення. Також художник підбирає кольори для персонажа та інших моделей (рис. 2.3) Висока деталізація допомагає зробити персонажа більш реалістичним. Колірна гама впливає сприйняття персонажа гравцями.

Основними кольорами персонажа вибрані сірий та білий. Додатковими – синій, червоний, жовтий.

По формі персонаж-робота має ергономічні форми, оснащений основною зброєю у вигляді гармати маленького калібру, дальноміром, маніпулятором, локатором. Головний персонаж не схожий формою живу істоту, тому ефект знищення не буде нагадувати сцени жорстокості з живими персонажами. А саме тому, в подальшому гру можна рекомендувати дітям старшого та середнього віку.

Рисунок 2.3 - Деталізація відібране зображення

2.3.3 Третій етап графічного проектування(створення 3D-моделей).

Створення моделей для подальшого використання у грі можна розбити на декілька етапів:

- Створення моделей
- Текстурування моделей
- Створення анімації моделей
- Створення графічних файлів
- Експорт моделей

2.3.3.1 Створення високополігональних об’єктів:

Виходячи з того, що фінальний вигляд гри повинен мати флору, то потрібно створити моделі дерев у 3DS Max для подальшого експорту у ігровий рушій. У вкладці Create відкриваємо Geometry та вибираємо зі списку AEC Extended, натискаємо на кнопці Foliage. У списку, що відкриється потрібно обрати необхідне дерево і перенести його у сцену. Після чого потрібно провести необхідні налаштування для цього відкриваємо вкладку Modify і відредагуємо наступні параметри: Height – висота дерева, Density – щільність листя на дереві, Pruning – обрізка гілок, параметр Level-of-Detail потрібно задати Low для того щоб модель не була занадто високополігональною (рис. 2.4). При створенні даного 3D об’єкту до нього автоматично застосовані матеріали. Тому модель дерева готова до експорту.

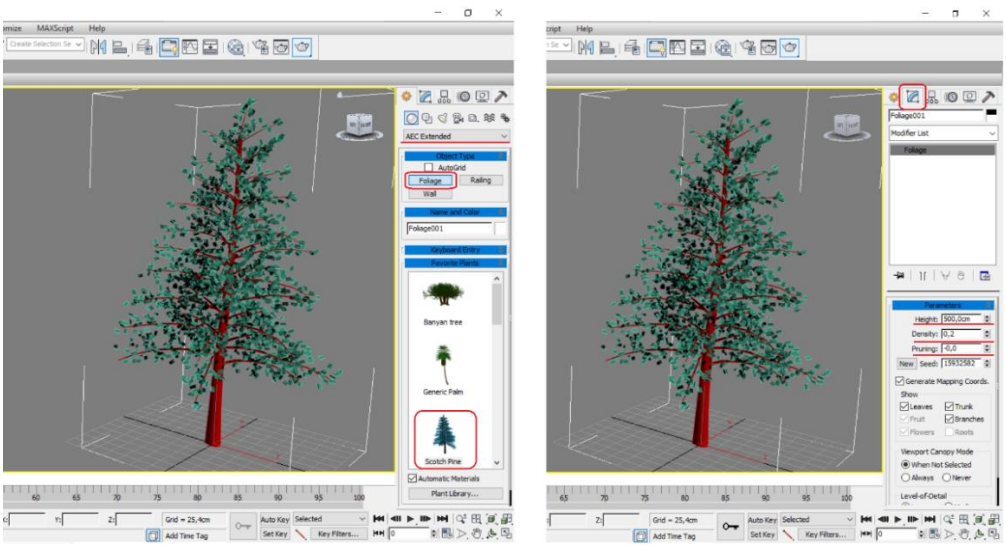

Рисунок 2.4 - Моделі дерев у 3DS Max

Одною з найскладніших задач у грі безумовно є створення головного персонажу. Головний персонаж у грі завжди знаходиться у полі зору, також через нього гравець взаємодіє з віртуальним світом, тому потрібно більш ретельно підійти до його створення. Створення головного персонажу гри потрібно почати з детального аналізу ескізу, а саме визначитися за допомогою яких інструментів краще моделювати ті чи інші частини. Головний персонаж у грі – робот, тому елементів органічного походження не буде. Тіло робота моделювалося за допомогою Extended Primitives(розширених примітивів) інструмент Capsule(капсула), цей примітив найбільш підходить для створення тіла робоота. Після чого потрібно масштабувати та конвертувати примітив капсули на Editable Poly(полігональна поверхня). Полігональну поверхню продовжуємо редагувати до вигляду що показаний на рис. 2.5. Опори робота створені за допомогою Shapes(двовимірні фігури) для створення верхньої частини використана фігура Line(лінія) для нижньої частини Rectangle(прямокутник). Для подальшого редагування потрібно конвертувати прямокутник у елемент Editable Spline(Редагований сплайн) лінію конвертувати не потрібно, тому шо лінія і є редагованим сплайном. Створивши контур у потрібній проекції, можна видавити об’єм за допомогою модифікатора Extrude(Видавити). Після видавлювання на необхідну товщину, конвертуємо опору у полігональну поверхню та продовжимо роботу над деталями. По закінченні створення опори її можна віддзеркалити на протилежну сторону, тому що обидві опори однакові. Частини маніпулятора в основному створювалися за допомогою стандартних примітивів, для створення більш складних деталей стандартні примітиви конвертувалися у редаговані поверхні та доводилися до необхідного вигляду.

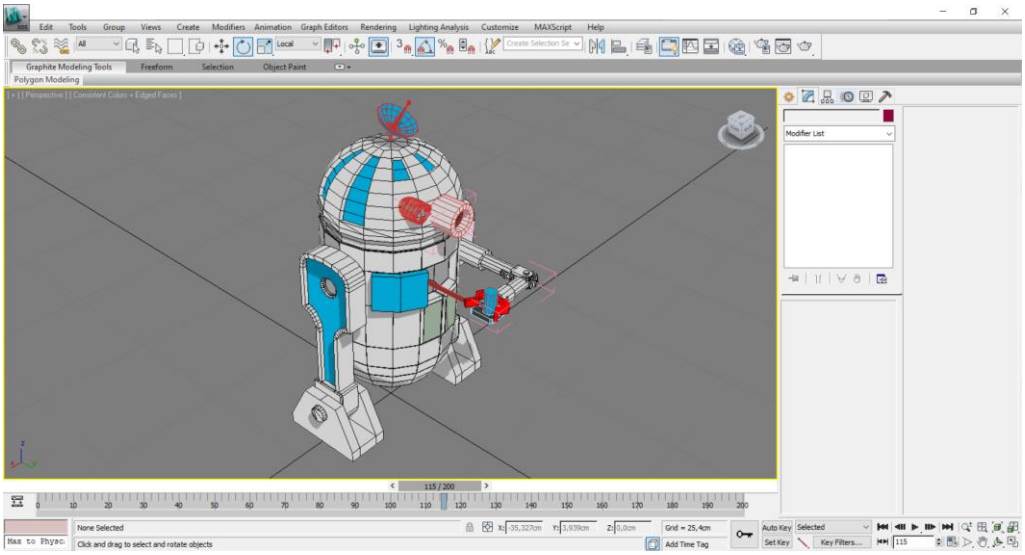

Рисунок 2.5 - Модель робота у вигляді полігональної поверхні

Модель робота на відміну від дерева складається з певної кількості 3D об’єктів котрі необхідно текстурувати.

2.3.3.2 Текстурування моделей:

Текстурування частин персонажу здійснюється за допомогою редактору матеріалів у 3DS Max або за допомогою редагування матеріалів безпосередньо в Unity. Для найбільш складних деталей об’єкта здійснюються розгортки. Головний персонаж, що використовується у грі не складний з точки зору текстурування, він складається з окремих згрупованих частин тому матеріали можна накласти безпосередньо в Unity. Виключення складає верхня частина робота. Зважаючи на мультикольорове забарвлення та ребристу поверхню, слід скористатися редактором матеріалів у 3DS Max для цього потрібно накласти на об’єкт модифікатор Unwrap UVW, який має редактор розгортки UV Editor, За допомогою редактора розгортки вручну розгорнемо площини даної фігури, як показано на рис. 2.6. Збережемо отриману розгортку у файл head.png для подальшого використання в Unity. У подальшому потрібно скористатися безкоштовним 2D редактором Paint, що поставляється разом з операційною системою, і розфарбувати у необхідні кольори, або накласти текстури на створені площини. Вибір 2D редактора Paint продиктований суто його безкоштовністю і тим, що проект невеликий і мало текстур підлягає

редагуванню. Але при великій кількості фонів і текстур, зручніше це робити у професійних застосунках - наприклад Photoshop(продукт фірми Adobe).

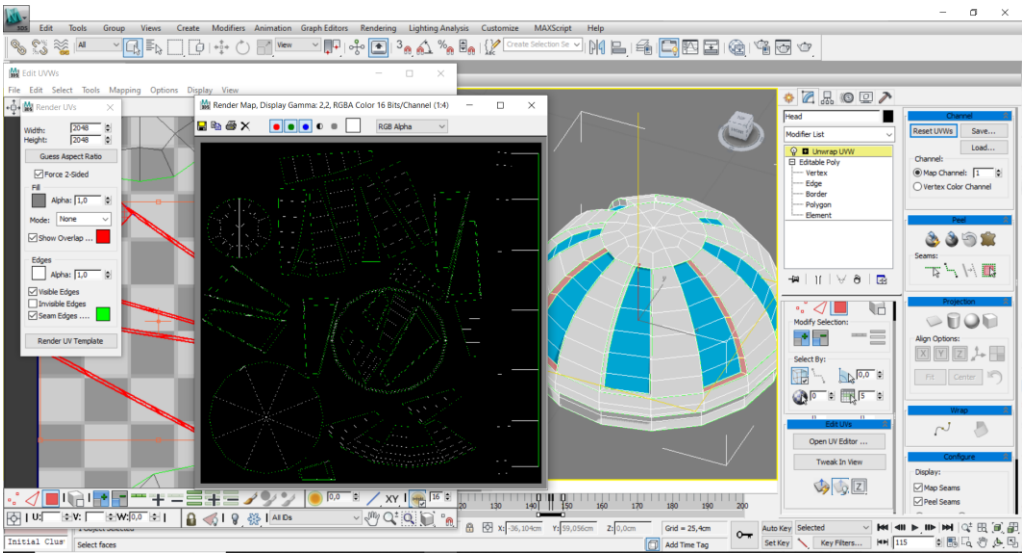

Рисунок 2.6 - Розгортка площини фігури

2.3.3.3 Створення анімації моделей:

При створенні гри важливе місце займає анімація персонажу та інших моделей що присутні у грі та яким потрібна анімація. Деяким об’єктам у грі анімація не потрібна, наприклад укриття та інші стаціонарні об’єкти. Складні анімації були створені у 3DS Max. До таких анімацій відноситься анімація персонажу. Анімація у 3DS Max відбувається за принципом так званих ключових кадрів. Суть цього принципу полягає у тому, що с початковий стан який фіксується за допомогою ключа і через певний відрізок часу моделюється кінцевий стан, який теж фіксується ключем, а проміжок між ними прораховується 3DS Max, враховуючи заданий розробником характер анімації. Розглянемо принцип анімації головного персонажу, тому що це найбільш анімований об’єкт гри, а інші об’єкти були анімовані за тим же принципами. На рис. 2.7 зображено шкалу часу яка займає 200 кадрів. У цей проміжок потрібно помістити всі рухи персонажу:

- З 0 по 20 кадр анімація ходьби – у грі повинна програватися під час руху персонажу. Пара кроків(крок лівою +крок правою) при спокійній ходьбі

схожі між собою і носить циклічний характер. Даний тип анімації краще описати синусоїдою.

- З 21 по 32 кадр анімація стрільби – програється у момент ведення вогню може носити як одиночний так і циклічний характер. У такому випадку орієнтуємося на створення циклічної анімації. Описується, як лінійний руху з прискоренням і уповільненням.
- З 33 по 133 кадр анімація спокою – програється у той момент коли персонаж стоїть нерухомо на місці. Анімація теж носить циклічний характер, тому що програється по циклу багато разів поспіль поки персонаж знаходиться у стані спокою. Але враховуючи те, що в ній задіяно багато різних частин не органічного походження і повинні чітко позиціонуватися тоданий тип анімації можна віднести до лінійного руху з прискоренням або уповільненням.

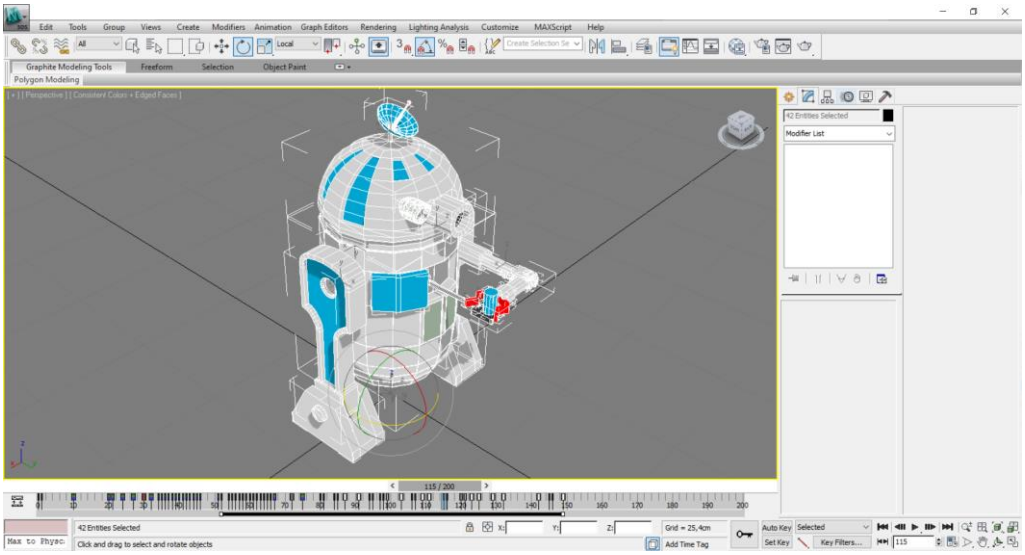

Рисунок 2.7 – Анімація головного персонажу

2.3.3.4 Створення графічних файлів

Створення моделі прицілу для подальшого використання у грі, як двовимірного об’єкту. Приціл у грі відіграє важливу роль, впливаючи на комфорт гри тому є сенс більш детально розглянути його створення. Потрібно створити двовимірний малюнок із співвідношенням сторін один до одного, в якому геометричний центр повинен співпадати з точкою прицілювання. Для

цього можна використати графічні редактори, бажано ті, що працюють з векторною графікою(найвідоміші CorelDRAW та Adobe Illustrator) для отримання правильних геометричних форм. Враховуючи те, що у даній роботі створення малюнку носить разовий характер, використаємо 3DS Max. Скористаємося категорією Shapes в якій знаходяться сплайни(Spline), створимо Circle(Коло), Line(Лінія), Text(Текст) у фронтальній проекції (рис. 2.8). Всі створені фігури повинні бути вирівняні відносно центру координат. Після отримання необхідної геометрії прицілу у сцену потрібно додати камеру без націлювання і вирівняти її відносно центру координат, потім у вікні проекції вибрати вигляд з камери (рис. 2.8). Після чого натиснути на клавіатурі клавішу F10 та виставити розподільчу здатність для збереження графічного файлу у формат png з прозорим фоном(pricel.png).

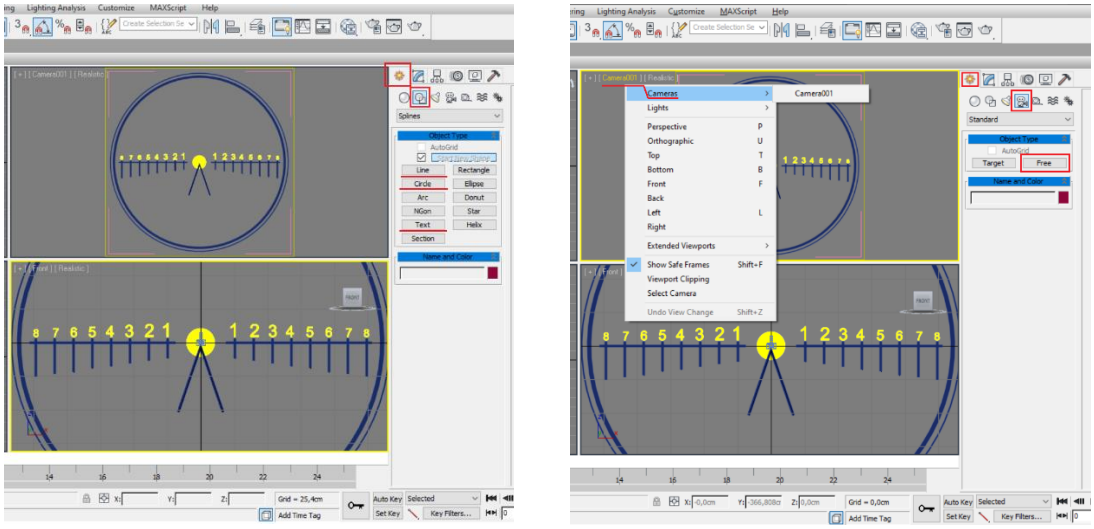

Рисунок 2.8 – Зображення прицілу

2.3.3.5 Експорт моделей

Після створення моделей їх потрібно експортувати для подальшого використання у грі. Складність експорту полягає у тому, щоб при експорті моделей у гру підтяглася не тільки модель, а і використані камери, світло, згаджування елементів, створена анімація і багато чого іншого. Якщо у моделі не складна геометрія і вона низькополігональна то можна імортувати у гру файли формату 3DS Max. Але якщо модель складна то потрібно

використовувати для експорту спеціальний формат файлів. Таким спеціалізованим форматом є fbx. На рис. 2.9 показано вікно експорту моделі головного персонажу із 3DS Max у формат fbx. Потрібно проставити галочки напроти тих пунктів що потрібно включити до моделі. У нашому випадку потрібно поставити галочки напроти згладжування і анімації як показано на рис. 2.9

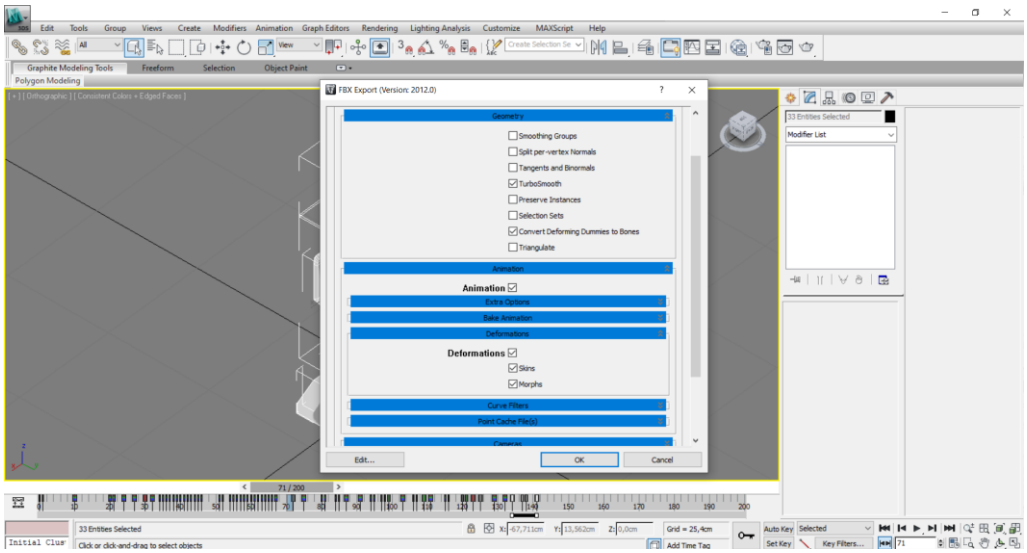

Рисунок 2.9 - Експорт моделі

3 ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

Найбільш складний етап виробництва, що має закінчитися випуском так званого бета тесту гри. В моєму випадку цей етап складається з наступних пунктів:

1) створення мережевого під’єднання гравців,
2) розробка ігрової механіки:
   - синхронізація гри,
   - розробка мережевої взаємодії персонажів,
   - налаштування ігрової механіки з урахуванням анімації персонажу та інших моделей
3) наповнення гри об’єктами.

3.1 Створення мережевого під’єднання гравців.

На цьому етапі необхідно створити під’єднання гравців у сцену гри. Мережева архітектура гри має наступний вигляд: клієнти для входу у гру під’єднуються до серверу, який може бути або окремо виділений, або один з клієнтів виступає як сервер.

Запускаємо Unity, обираємо створення нового проекту з назвою ShooterCZ1 вказуємо шлях до теки на комп’ютері де буде розміщено проект гри і вказуємо шаблон 3D, як показано рис. 3.1.

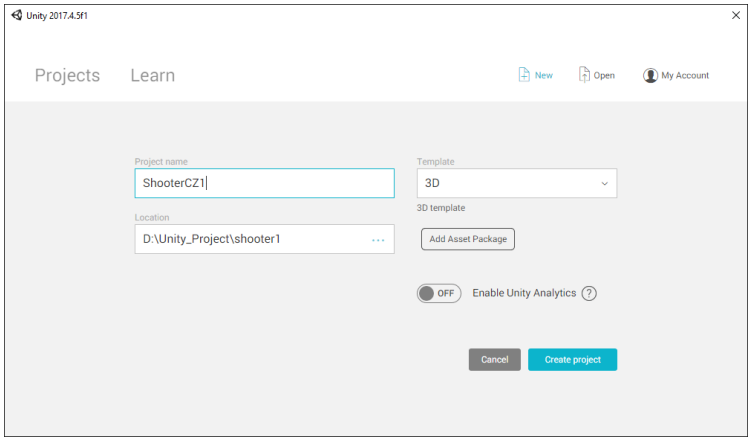

Рисунок 3.1 - Створення нового проекту

При створенні сцена має стандартний вигляд

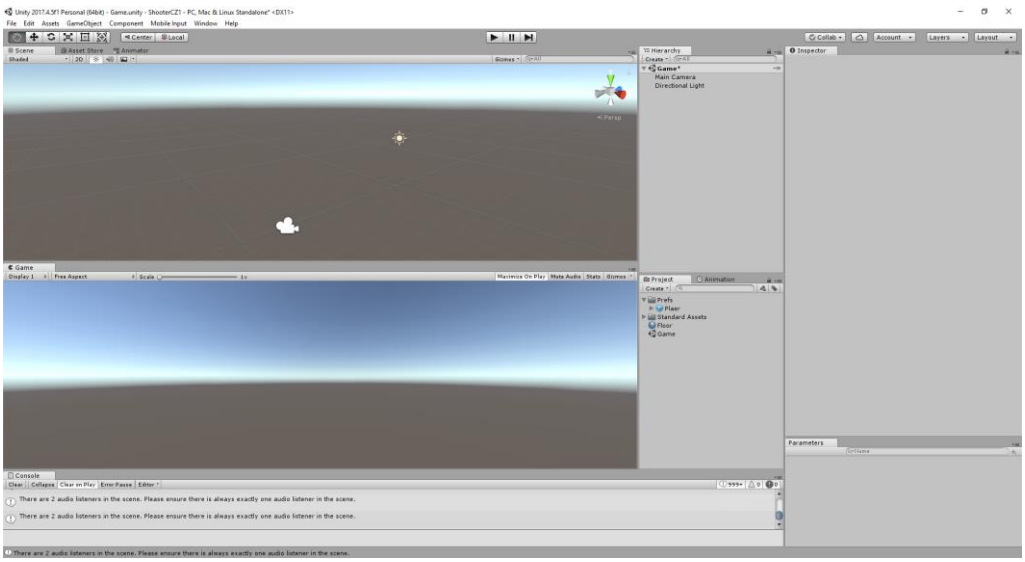

Рисунок 3.2 – Створення сцени

Як видно з рис. 3.2 при створенні сцени за замовчуванням присутнє джерело світла і камера. Джерело світла потрібно для того, щоб об’єкти були видно у сцені, а за допомогою камери гравець має змогу спостерігати за віртуальним світом. Перше що потрібно створити: площину у вигляді підлоги, матеріал для підлоги та декілька примітивних об’єктів для орієнтації у просторі, а також FPSController для того щоб створити динаміку і механіку персонажу в сцені. Для створення примітивних об’єктів в головному меню обираємо GameObject->3D Object->(Plane/Cube), у вікні Inspector трансформуємо до потрібних розмірів, назначаємо, за потреби матеріал, регулюємо фізичні властивості(зіткнення, видимість і т.д.), як показано на рис. 3.3.

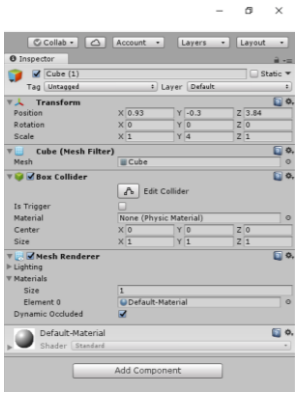

Рисунок 3.3 - Вікно Inspector

Назначаємо матеріал підлозі, для цього потрібно натиснути правою кнопкою мишки (ПКМ) у вікні Project і вибрати матеріал рис. 3.4, надати йому назву, наприклад Floor, у вікні Inspector встановити властивості(колір, відблиск і т.д.)

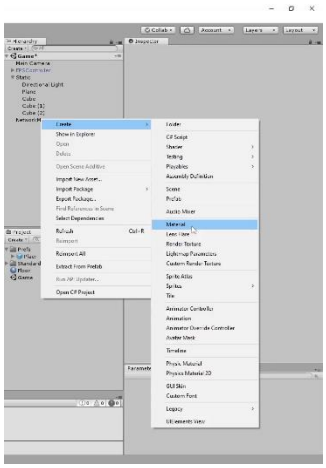

Рисунок 3.4 - Вікніо Project

Для створення FPSController у сцену потрібно додати стандартний пакет, для цього потрібно натиснути ПКМ у вікні Project і у меню вибрати Import Package->Characters як показано на рис. 3.5. Після цього додається стандартний пакет з персонажами, які спеціально використовуються для shooter від першої особи.

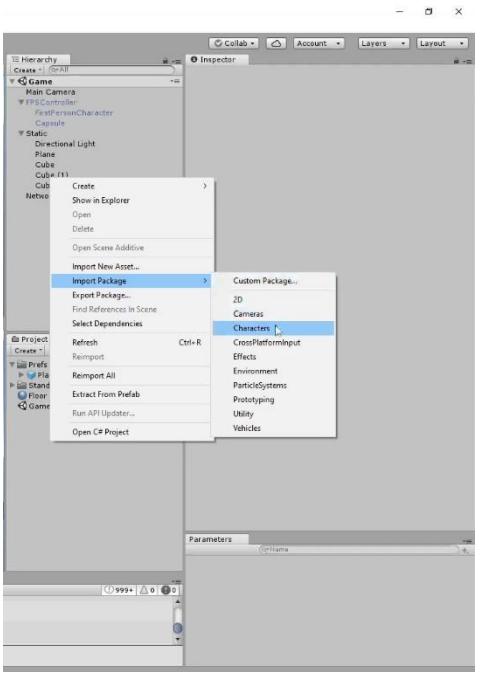

Рисунок 3.5 – Імпорт стандартного пакету з персонажами

У вікні Project додається папка із стандартними заготовками Standard Assets. Для додавання потрібно перейти у папку:

Standard Assets/Characters/FirstPersonCharacter/Prefabs/ натиснути лівою кнопкою мишки на FPSController і перетягнути його у вікно сцени (рис. 3.6).

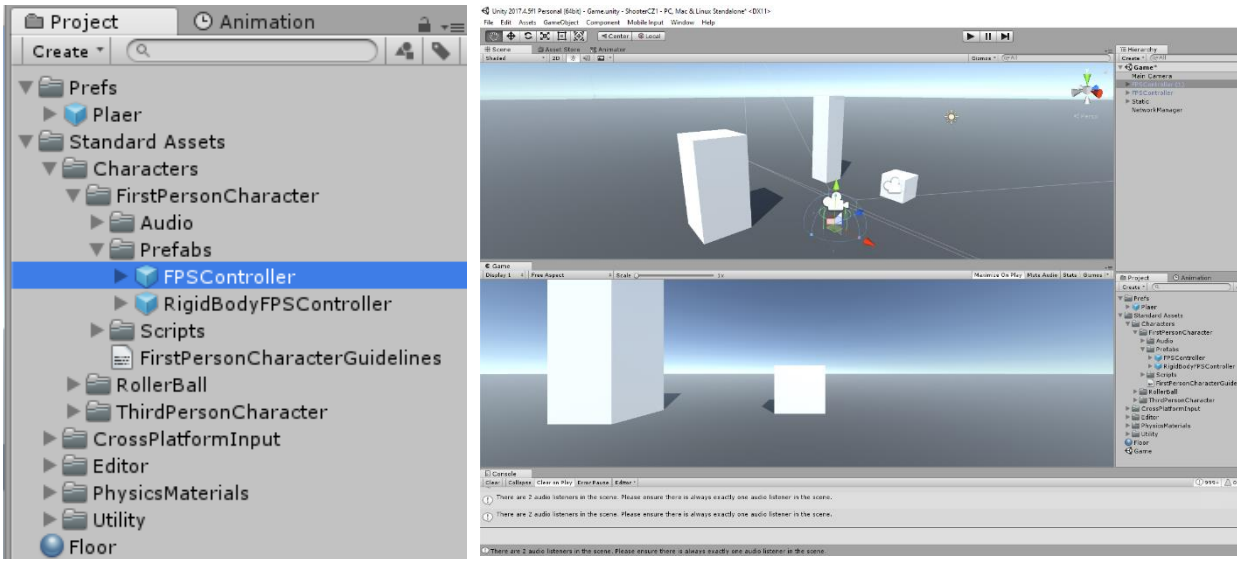

Рисунок 3.6 – FPSController

Для візуалізації FPSController потрібно натиснути на ньому ПКМ у вікні Hierarchy і у спливаючому меню вибрати 3D Object/Capsule, після чого FPSController буде мати вигляд, як показано на рис. 3.7. Для налаштування мережевого під’єднання гравців на даному етапі розробки цього вистачить.

Рисунок 3.7 – Візуалізація FPSController

Щоб гравець автоматично створювався при під’єднанні у гру при виконанні скрипта потрібно помістити FPSController у Prefab. Для цього у вікні Project необхідно створити папку Prefs потім на ній натиснути ПКМ, у випадаючому меню обрати Create/Prefab і перейменувати створений New

Prefab на Plaer. З вікна Hierarchy натиснувши ЛКМ на FPSController не відпускаючи перетягнути на створений Plaer доки він не стане синього кольору і відпустити.

Для створення менеджера мережевого підключення потрібно додати пустий об’єкт для цього у головному меню натискаємо GameObject/Create Empty. Створений пустий об’єкт перейменовується у Network Manager. У вікні Inspector натискаючи на кнопку Add Component у випадаючому списку потрібно додати компонент NetworkManager. Так само потрібно додати компонент NetworkManagerHUD, як показано на рис. 3.8.

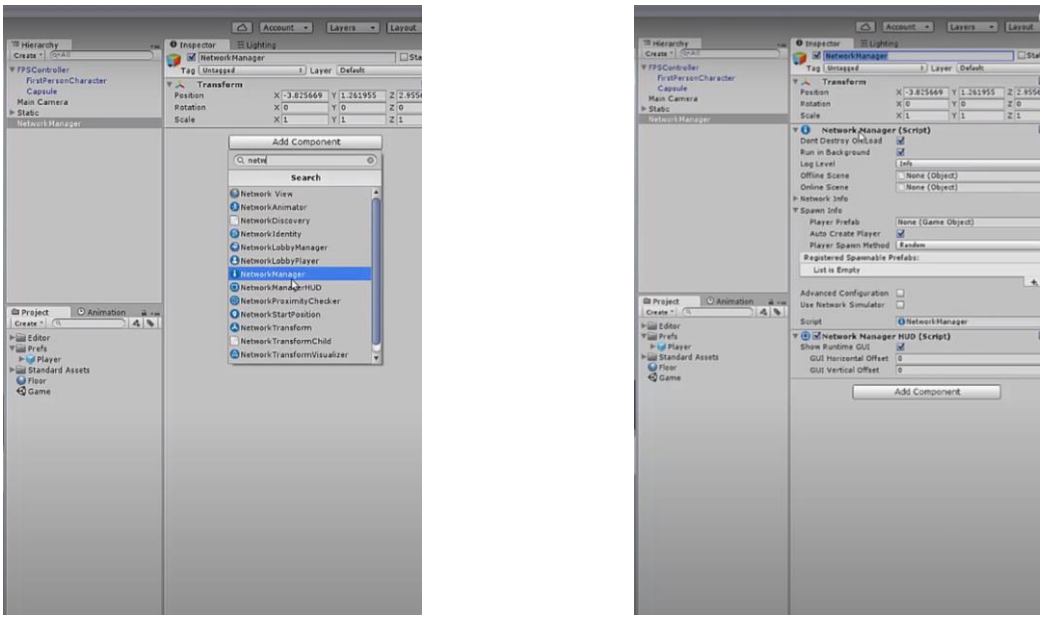

Рисунок 3.8 – Компонент NetworkManagerHUD

компонент NetworkManager відповідає за створення серверу і підключення до нього, а компонент NetworkManagerHUD відповідає за його візуалізацію, при запуску гри автоматично створюється меню з кнопками LAN_Host(H), LAN_Client(C), LAN_Server_Only(S).

Для запуску потрібної сцени при запуску гри у вікно Inspector списку властивостей NetworkManager поле Online Scene потрібно натиснувши ЛКМ перетягти сцену Game.

Для автоматичного створення гравця при під'єднанні потрібно у властивості FPSController додати компонент Network Identity Local в якому потрібно ввімкнути галочку напроти поля Player Authority, що дозволяє керувати гравцем будучи сервером. Також потрібно додати компонент Network Transform, він відповідає за синхронізацію позиції і обертання гравця. Потім потрібно натиснути кнопку Apply і всі зміни у Player залишаються.

Після чого продовжується налаштування NetworkManager в поле Player Prefab потрібно перетягнути Player.

Наступним кроком потрібно організувати перевірку того, які компоненти якому гравцю належать. Без такої перевірки клієнт, що під'єднався до гри може перехватити управління «чужим» гравцем. Попередньо потрібно відключити(зняти галочки) ті компоненти, які потрібно контролювати у FPSController це компоненти Character Controller та First Person Controller (Script). У дочірньому компоненті FirstPersonCharacter потрібно вимкнути камеру Camera і слухача звуків Audio Listener, як показано на рис.3.9**Ошибка! Источник ссылки не найден.**. У тимчасовій моделі гравця Capsule вимкнути компонент Capsule Collider(зіткнення), тому що за зіткнення відповідає контролер.

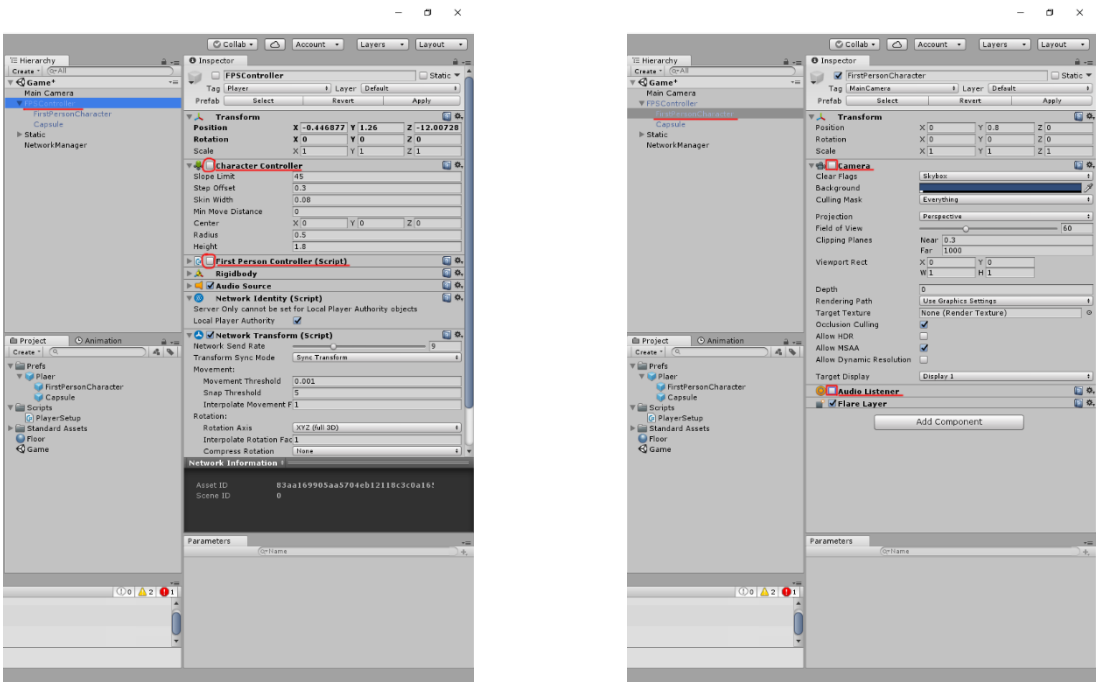

Рисунок 3.9 – Компонент NetworkManagerHUD

Для контролю відключених параметрів потрібно створити скрипт що буде їх контролювати. Створюємо теку з назвою Scripts, куди і будуть у подальшому додаватися всі скрипти. У вікні Project створимо C#Script під назвою PlayerSetup. Подвійне натискання на ньому ЛКМ запропонує його відкриття у редакторі. Потрібно відредагувати скрипт наступним чином:

```
using UnityEngine;
using System.Collections;
using UnityEngine.Networking; //підключаэмо додаткову бібліотеку

public class PlayerSetup : NetworkBehaviour //Змінюємо батьківський клас на NetworkBehaviour
{
  // вказуємо ті компоненти, що будемо робити для кожного клієнта унікальними
  public Camera CharacterCamera;// публічна змінна типу Камера
  public AudioListener CharacterAudioListener; // публічна змінна звуків
  // Use this for initialization
  void Start () // відбувається ініціалізація гравця
  {
    if (isLocalPlayer)
    {
      Destroy(GameObject.Find("Main Camera"));
      GetComponent<CharacterController>().enabled = true; //підключаємо контролер

  GetComponent<UnityStandardAssets.Characters.FirstPerson.FirstPersonController>().enabled = true; //підключаємо скрипт
      CharacterCamera.enabled = true; // вмикається камера, що належать гравцю
      CharacterAudioListener.enabled = true; // вмикаються звуки, що належать гравцю
    }
  }

  // Update is called once per frame
  void Update () {
```

```
    }
}
```

Після редагування скрипта потрібно додати його до FPSController і у скрипті вказати камеру і список звуків. Для цього потрібно натиснувши ЛКМ на скрипті PlayerSetup перетягти його у вікно Inspector, а FirstPersonCharacter із вікна Hierarchy перетягти на поля CharacterCamera та CharacterAudioListener, як показано на рис. 3.10.

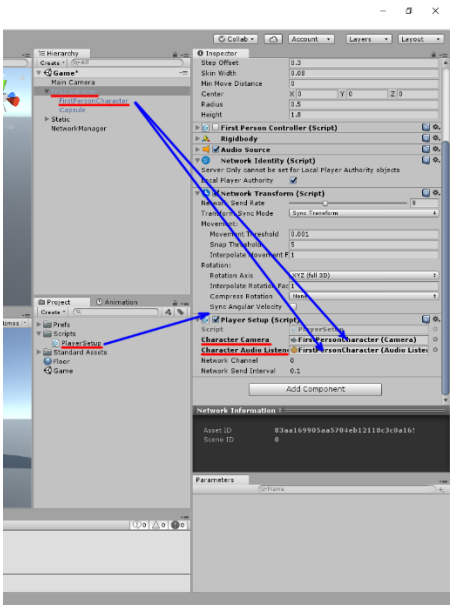

Рисунок 3.10 - Вікна Hierarchy

Потрібно змінити назву компонента FirstPersonCharacter на Camera, як показано на рис.3.11.

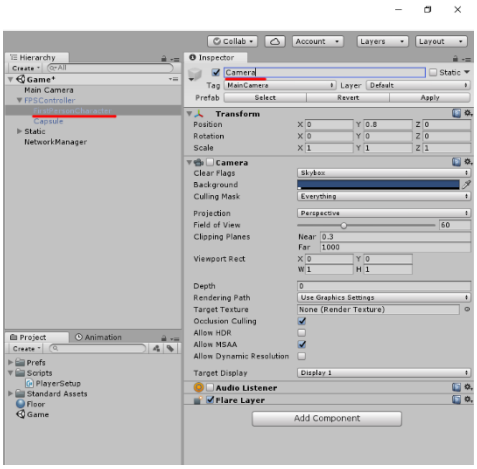

Рисунок 3.11 – Компонент Camera

Після цього запускаємо гру у вікні Unity, щоб перевірити чи не виникає помилок. Після чого компілюємо гру натиснувши File/BuildSettings і натискаємо кнопку Build, як показано на рис. 3.12 Unity запропонує вибрати теку у яку буде скомпільовано файл гри.

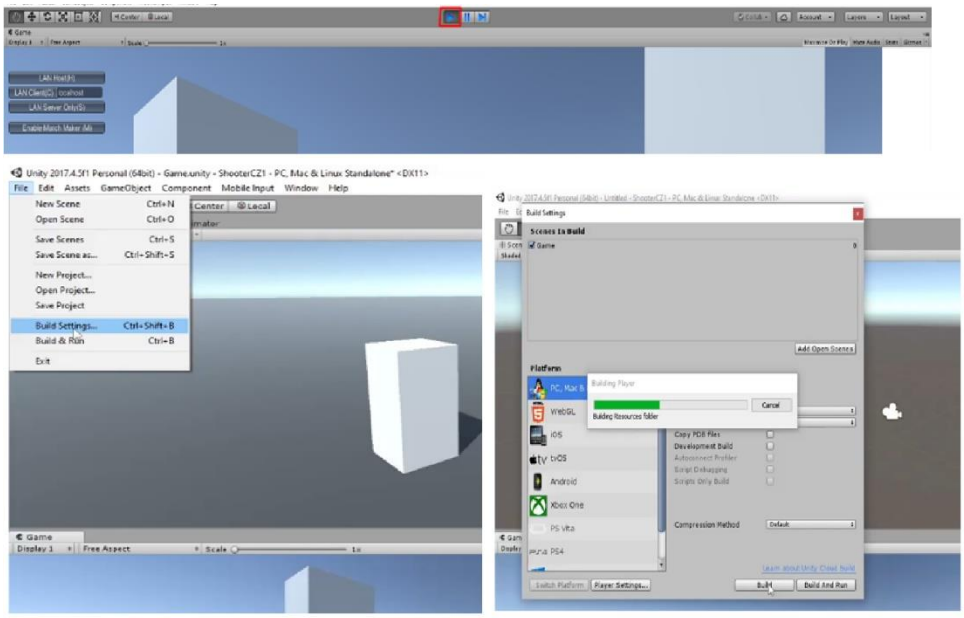

Рисунок 3.12 - Скомпіляція файлу гри

Щоб пересвідчитися що гра працює правильно потрібно запустити одного гравця з оболонки Unity, а другого зі скомпільованого файлу.

Запускаємо сервер з Unity, натискаючи на кнопку LAN Host(H), як показано на рис. 3.13.

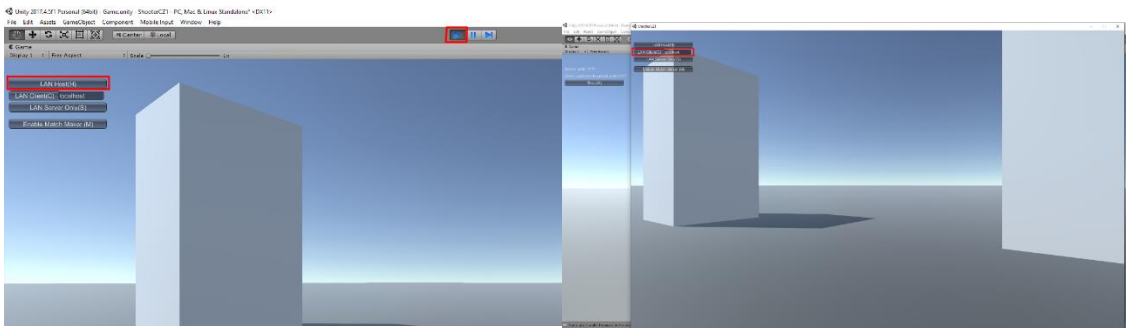

Рисунок 3.13 - Запуск серверу

При запуску скомпільованого файлу потрібно вибрати в меню LAN Client(C) у полі залишити localhost, тому що тестується з того самого комп’ютера.

На цьому налаштування мережевого під’єднання можна вважати завершеними. Але для повного тесту потрібно, щоб від першої особи можна бачити тільки руки, а персонаж суперника бачити як окремий об’єкт(на даному етапі це капсула). Потрібно експортувати створений раніше об’єкт рук до Unity, для цього потрібно в головному меню натиснути Assets/Import New Asset… і вибрати шлях до теки із раніше створеним об’єктом рук, а саме до файлів під назвою Hands1.fbx та текстури до ньогоText1.jpg, після чого у вікні Project з’явиться об’єкт Hands1. Щоб прив’язати руки до персонажу, потрібно створити пустий об’єкт, зробити його дочірнім FPSController і переіменувати у Hands, і вложити у нього об’єкт Hands1, перетягнувши мишкою із вікна Project. У подальшому в нього можна вкласти всі необхідні об’єкти, не тільки руки, а і зброю. Для меншого навантаженні на графіку гри потрібно вимкнути тіні на об’єкті Hands1, для цього потрібно перемкнути CastShadows у положення off, як показано на рис. 3.14

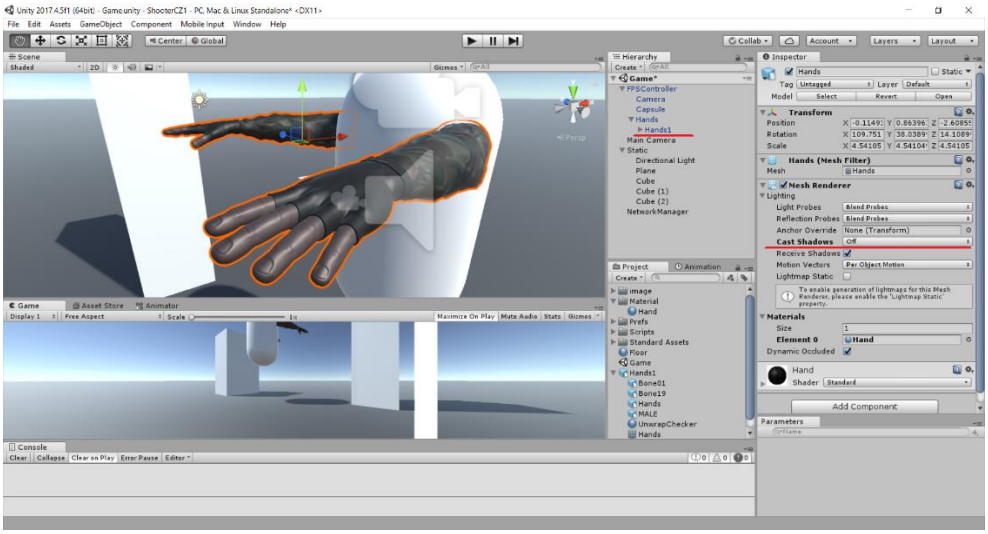

Рисунок 3.14 - Налаштування мережевого під’єднання

Потрібно внести поправки у скрипт PlayerSetup, а саме контроль відображення у власного персонажа рук, а у стороннього - тіла. Після внесення змін у редакторі, що виділені жирним шрифтом, скрипт матиме вигляд:

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using UnityEngine.Networking; //підключаэмо додаткову бібліотеку

public class PlayerSetup : NetworkBehaviour //Змінюємо батьківський клас на NetworkBehaviour
{
  // вказуємо ті компоненти, що будемо робити для кожного клієнта унікальними
  public Camera CharacterCamera;//створюємо публічну змінну типу Камера
  public AudioListener CharacterAudioListener; //створюємо публічну змінну звуків
  public Transform hands; //публічна змінна для відображення рук
  public Transform clientModel; //публічна змінна для відображення тіла

  // відбувається ініціалізація персонажу
  void Start ()
  {
    if (isLocalPlayer) {
      Destroy (GameObject.Find ("Main Camera"));
      GetComponent<CharacterController> ().enabled = true; //підключаємо контролер
  GetComponent<UnityStandardAssets.Characters.FirstPerson.FirstPersonController> ().enabled = true; //підключаємо скрипт
      CharacterCamera.enabled = true; // вмикаємо камеру, що належить гравцю
      CharacterAudioListener.enabled = true; // вмикаємо список звуків, що належить гравцю
      hands.GetComponent<MeshRenderer> ().enabled = true; // вмикаємо руки якщо це наш персонаж
    }
    else {
      clientModel.GetComponent<MeshRenderer> ().enabled = true; // вмикаємо тіло якщо це чужий персонаж
    }
```

```
    }

    // Update is called once per frame
    void Update () {

    }
}
```

У вікні Inspector потрібно змінним скрипта PlayerSetup hands та clientModel присвоїти значення, як показано на рис. 3.15. Також необхідно вимкнути Mesh Render об’єктів Capsula та Hands. Переходимо до FPSController і зберігаємо зміни натиснувши Apply.

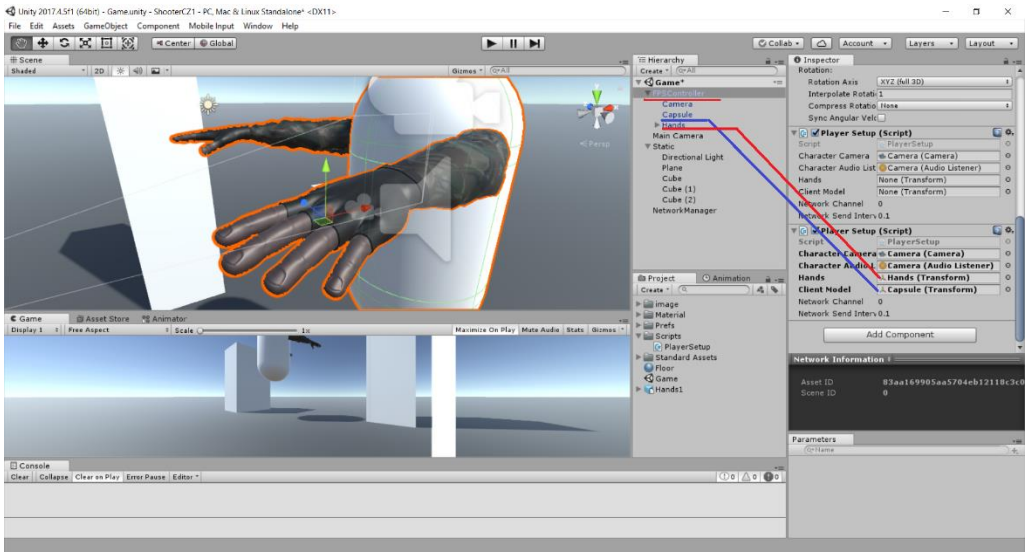

Рисунок 3.15 - Об’єкти Capsula та Hands

Після цього потрібно знову скомпілювати проект і провести перевірку, запустивши одного гравця з вікна Unity, а другого зі скомпільованого файлу гри.

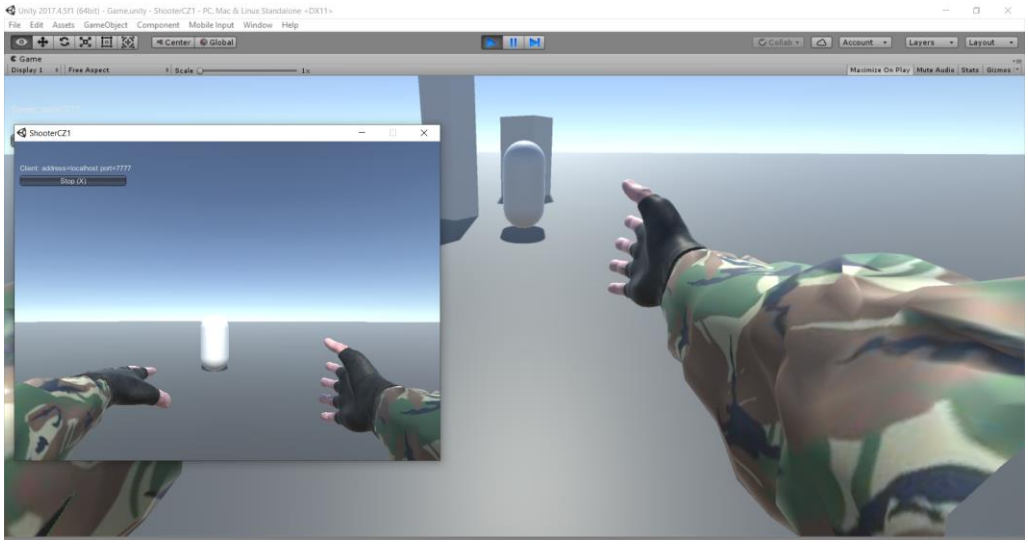

Рисунок 3.16 – Перевірка мережевого під’єднання

На рис. 3.16 видно, що кожен гравець бачить у власного персонажа руки, а чужого персонажа бачить як об’єкт. На цьому етап мережевого під’єднання повністю завершений і випробуваний.

3.2 Розробка ігрової механіки

3.2.1 Синхронізація гри, згладжування обертання та рухів персонажів:

Згладжування синхронізації персонажів всіх клієнтів у грі відіграє важливу роль, це стосується ігрового комфорту, задача полягає у тому, щоб мінімізувати ривки при переміщенні персонажів як свого так і сторонніх клієнтів у грі. Потрібно створити скрипт синхронізації персонажів саме для даного проекту і замінити ним стандартний.

Як і у попередньому випадку потрібно створити скрипт мовою C#, дати назву PlaerSyncPos і відредагувати наступним чином:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Networking; // підключення стандартної мережевої бібліотеки

public class PlaerSyncPos : NetworkBehaviour // Заміна батьківського класу
{ [SyncVar] //атрибут, що робить змінну синхронізованою, тобто при її зміні сервер посилає її значення всім клієнтам
```

```
  private Vector3 syncPosition; // змінна відповідає за позицію персонажів окрім нашого їх наступна точка руху це змінна suncPosition
  public float lerpSpeed = 20 ;
  [SyncVar]
  private Quaternion syncRotation; // змінна відповідає за обертання персонажів окрім нашого

  void FixedUpdate () //функція гарантовано незалежно від "лагів" гравця буде оновлюватися 0,02 сек(за замовчуванням)
  {
    LerpPosition ();
    TransmitPosition ();
  }
  void LerpPosition() // функція що відповідає за згладжування
  {
    if(!isLocalPlayer)// робимо перевірку якщо ми сторонній кліент(згладжуємо позицію всім окрім себе)
    {
      transform.position = Vector3.Lerp (transform.position, syncPosition, Time.deltaTime*lerpSpeed); /*зглажуємо позицію відносно
      поточного положення клієнта transform.position до наступної точки syncPosition швидкість згладжування Time.deltaTime
      її для зручності зміни привїяжемо до змінної lerpSpeed*/
      transform.rotation = Quaternion.Lerp (transform.rotation, syncRotation, Time.deltaTime * lerpSpeed);/*зглажуємо обертання з тією швидкістю, що і позицію*/
    } }
  [Client]/*атрибут Client означає те що функція виконується у всіх окрім сервера*/
  void TransmitPosition()
  {
    if (isLocalPlayer) //перевірка якщо ми є гравцем то ми відправляємо нашу позицію і обертання на сервер
    {
      CmdSendPosition (transform.position);
      CmdSendRotation (transform.rotation);
    }
  }
  [Command]/*атрибут відправляэ любі команди від клієнта на сервер. Посилаемо сповіщення на сервер з поточної позиції гравця*/
  void CmdSendPosition(Vector3 pos) //назва функції обов'язково повинна починатися Cmd
  {
    syncPosition = pos; //сервер дані приймає і записує у змінну
  }
  [Command]
  void CmdSendRotation(Quaternion rot) //назва функції обов'язково повинна починатися Cmd
  {
    syncRotation = rot; //сервер дані приймає і записує у змінну
  }}
```

Потрібно протестувати роботу скрипта. Для цього переносимо скрипт PlaerSyncPos на Prefab з іменем Player і вимикаємо стандартний скрипт Network Transform, як показано на Рисунок . Якщо залишити капсулу то неможливо буде побачити обертання. Тому у гру додамо створеного головного персонажа, але без анімації рухів. Скомпілювати гру і запустити сервер у вікні Unity, а клієнта із вікна скомпільованої гри. Скрипт працює правильно ривків у рухах і обертанні персонажів не виявлено вікно гри має вигляд, як показано на рис. 3.18

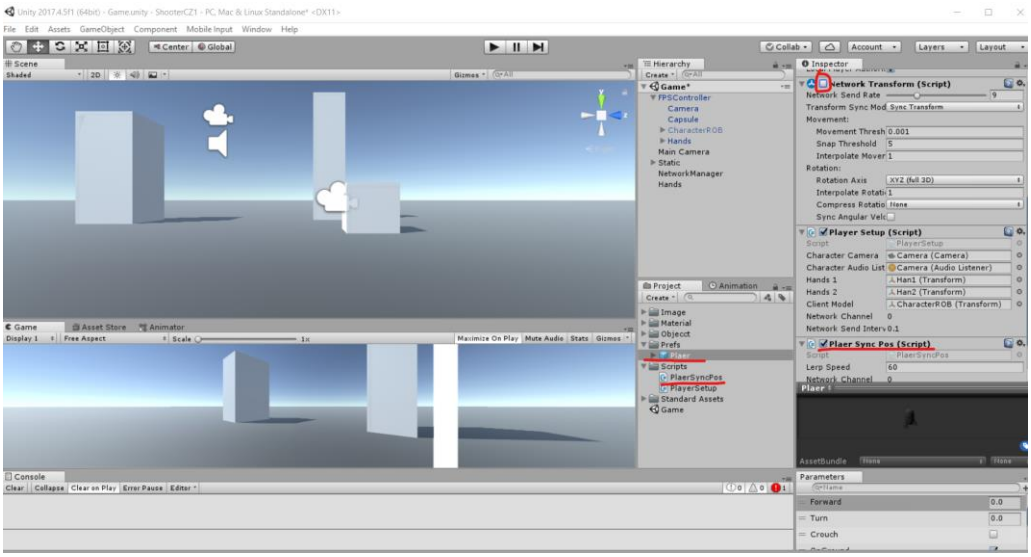

Рисунок 3.17 – Тестування роботи скрипта

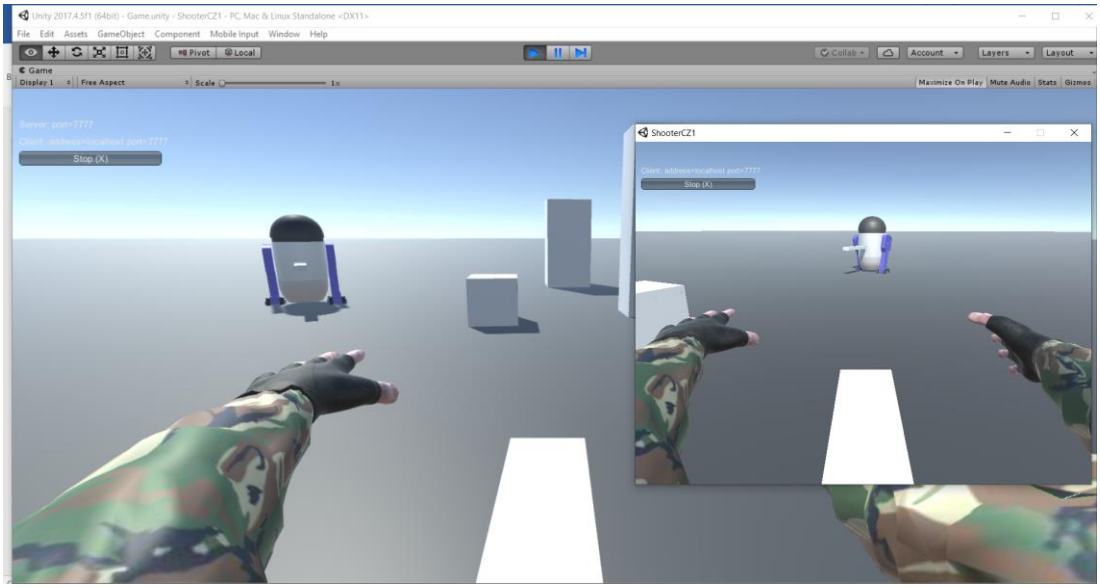

Рисунок 3.18 - Вікно гри

3.2.2 Розробка мережевої взаємодії персонажів:

Реалізація мережевої взаємодії полягає у тому що гравець повинен мати доступ до інформації, яка необхідна для успішного геймплея і ця інформація повинна синхронізуватися. Мінімальний набір таких даних – це ім’я гравця, шкала здоров’я, влучання та отримання пошкоджень, знищення та відродження гравця.

3.2.2.1 Ім’я гравця:

Завдання цього етапу полягає у тому, щоб дати персонажам клієнтів, що заходять у гру унікальні імена(прототип ніків гравців). У подальшому до цих імен можна звертатися і зберігати ігрову інформацію персонажів. Імена можна формувати за принципом «ім’я+номер». Ім’я надавати, а номер брати зі змінної Net Id, що знаходиться у компоненті Network Identity і містить унікальний номер підключення, що присвоюється кожному персонажу, який заходить у гру. Тому задачу можна спростити, а саме, прив’язати ім’я гравця до його унікального номера.

Як і раніше створимо скрипт мовою C#, надамо назву PlaerInfo і відредагуємо наступним чином:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Networking;

public class PlaerInfo : NetworkBehaviour {

  // Use this for initialization
  void Start ()
  {
    transform.name = "Player " +
GetComponent<NetworkIdentity>().netId.ToString();     /*отримуємо
значення
    змінної netId і фунцією ToString() перетворюємо у строкове
значення*/
  }

  // Update is called once per frame
  void Update () {

  }
}
```

Перетаскуємо скрипт PlaerInfo на Prefab з іменем Player, після чого протестуємо роботу скрипта. Запускаємо гру у мультивіконному режимі і можемо бачити що персонаж має ім’я Player 1 (рис. 3.19). На даний момент скрипт працює коректно.

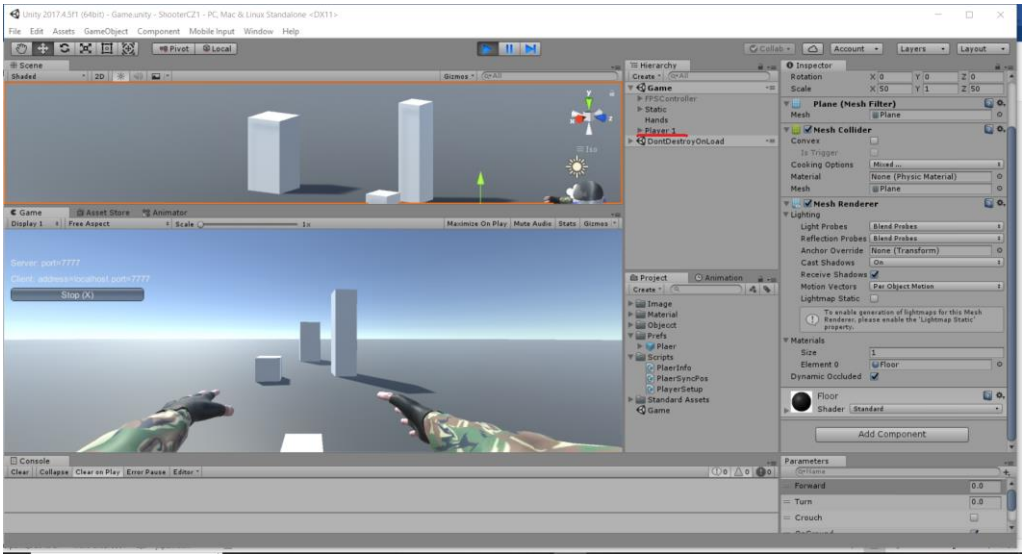

Рисунок 3.19 - Мультивіконний режим

Наступний крок – виведення текстової інформації над гравцем. Для цього створюємо пустий елемент назвемо його Text і додамо до FPSController. Після цього, натискаючи кнопку Add Component у вікні Inspector, додамо у елемент Text компонент Mesh/Text Mesh як показано на рис. 3.20. У полі Text напишемо для початку «Player».

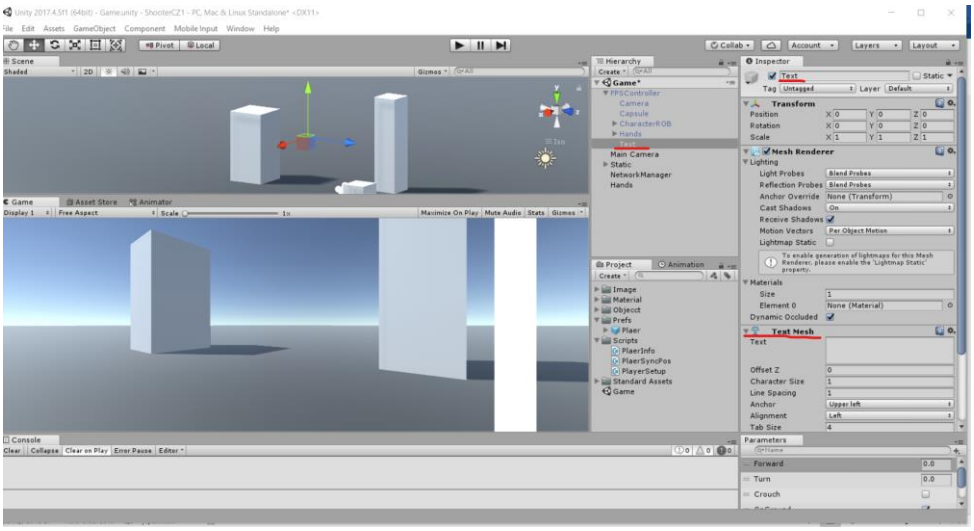

Рисунок 3.20- Компонент Mesh/Text Mesh

У скрипт PlaerInfo допишемо код, для того, щоб можна було звертатися до тексту і виводити потрібну інформацію. Допишемо декілька команд, після чого він буде мати наступний вигляд:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Networking;

public class PlaerInfo : NetworkBehaviour {
  public TextMesh tm;
  // Use this for initialization
  void Start ()
  {
    transform.name = "Player " +
GetComponent<NetworkIdentity>().netId.ToString();
    tm.text = transform.name;
  }

  // Update is called once per frame
  void Update () {

  }
}
```

Протестуємо роботу скрипта PlaerInfo на даному етапі. На Рисунок видно що наші персонажі мають імена з номерами біля назви. Гравець, що зайде сервером матиме перший номер, а інші номери будуть присвоєні у тому порядку, у якому гравець зайде у гру.

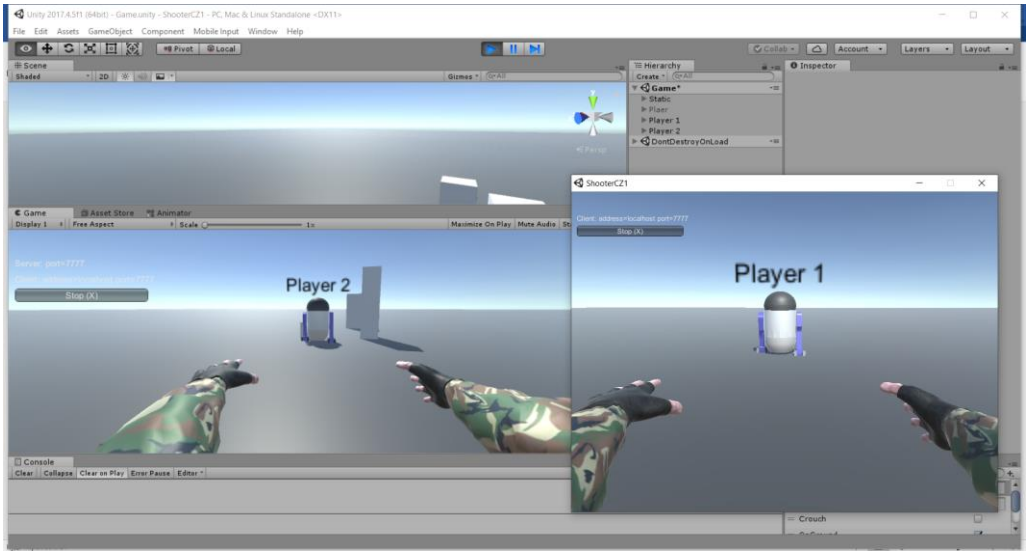

Рисунок 3.21 - Тест роботи скрипта PlaerInfo

3.2.2.2 Шкала здоров’я:

На даному етапі задача полягає у тому, щоб при вході у гру присвоїти значення одиниць здоров’я(шкала здоров’я), цю інформацію повинні бачити інші гравці і на додачу ця інформація повинна синзронізуватися. Для реалізації цієї задачі прродовжимо редагувати скрипт PlaerInfo:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Networking;

public class PlaerInfo : NetworkBehaviour {

  [SyncVar] public int Health = 100;/*змінна, що
ввідповідає за здоров'я атрибут [SyncVar] синхронізує
її*/
  public TextMesh tm;
  // Use this for initialization
  void Start ()
  {
    transform.name = "Player " +
GetComponent<NetworkIdentity>().netId.ToString();
    tm.text = transform.name;
  }

  // Update is called once per frame
  void Update () {

  }
  void OnGUI()/*виводимо здоров'я в інтерфейсі гравця*/
  {
    if (isLocalPlayer)  /*якщо ми є гравцем то на екран
буде виводитися тільке наше здоров'я*/
    {
      GUI.Label (new Rect (Screen.width - 100, 25, 200,
50), "Health: " + Health);
    }
  }
}
```

Протестувавши виконання скрипта можна помітити, що у правому верхньому куті з’явилася текстова інформація про стан «нашого» здоров’я (рис. 3.22).

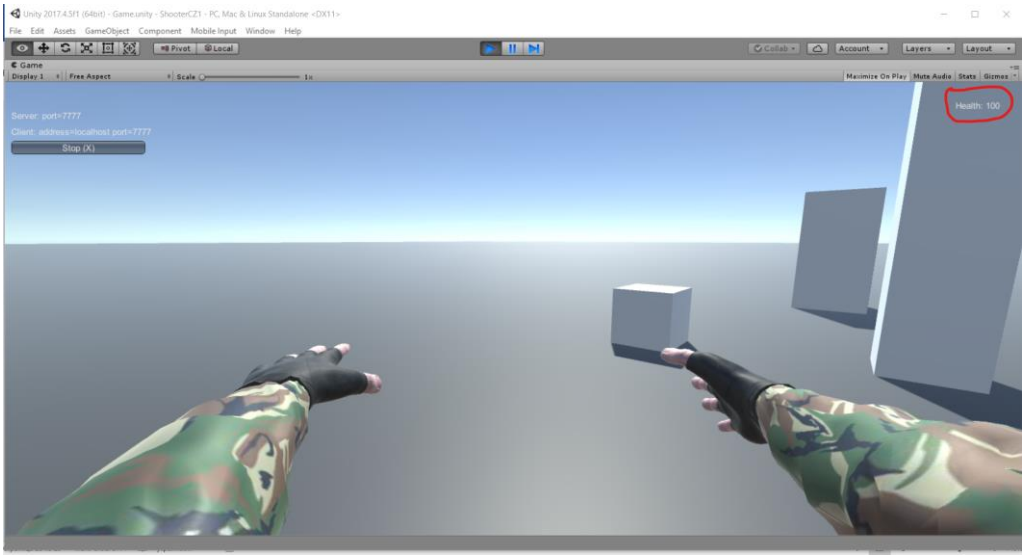

Рисунок 3.22 - Шкала здоров’я

3.2.2.3 Влучання та отримання пошкоджень:

Останній крок у створенні мережевої взаємодії це написання скрипта стрільби, що буде реалізовувати механізм стрільби і при влучанні забирати здоров’я гравця. Траєкторія пошкодження буде у вигляді променів, які виходять з камери до об'єкта. Пошкодження наносяться завдяки кліку мишкою на моделі персонажу. Створимо скрипт мовою C#, який буде реалізовувати отримання пошкоджень у результаті влучання надамо назву PlaerShooting. Цей скрипт буде використовуватись самим гравцем тому зразу додамо його у Prefab з іменем Player і вимкнемо(знімемо галочку). Потрібно відкрити скрипт PlayerSetup і додати умову, щоб компонен стрільби(PlaerShooting) вмикався у тому випадку коли ми є гравцем. Після редагування він буде виглядати наступним чином:

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using UnityEngine.Networking; //підключаэмо додаткову
бібліотеку

public class PlayerSetup : NetworkBehaviour //Змінюємо
батьківський клас на NetworkBehaviour
{
  // вказуємо ті компоненти що будемо робити для
кожного клієнта унікальними
```

```
  public Camera CharacterCamera;//створюємо публічну
змінну типу Камера
  public AudioListener CharacterAudioListener;
//створюємо публічну змінну звуків
  public Transform hands1; //публічна змінна для
відображення руки 1
  public Transform hands2; //публічна змінна для
відображення руки 2
  public Transform clientModel; //публічна змінна для
відображення тіла

  // відбувається ініціалізація гравця
  void Start ()
  {
    if (isLocalPlayer) {
      Destroy (GameObject.Find ("Main Camera"));
      GetComponent<CharacterController> ().enabled =
true; //підключаємо контролер

  GetComponent<UnityStandardAssets.Characters.FirstPers
on.FirstPersonController> ().enabled = true;
//підключаємо скрипт
      CharacterCamera.enabled = true; // вмикаємо
камеру, що належить гравцю
      CharacterAudioListener.enabled = true; // вмикаємо
список звуків, що належить гравцю
      hands1.GetComponent<MeshRenderer> ().enabled =
true; // вмикаємо руки якщо це наш персонаж
      hands2.GetComponent<MeshRenderer> ().enabled =
true; // вмикаємо руки якщо це наш персонаж
      GetComponent<PlaerShooting>().enabled = true;
    }
    else {
      clientModel.GetComponent<MeshRenderer> ().enabled
= true; // вмикаємо тіло якщо це чужий персонаж
    }
  }

  // Update is called once per frame
  void Update () {

  }
}
```

Також додамо у скрипт PlaerInfo функцію, що буде віднімати від здоров’я персонажу завдані йому пошкодження:

```
public void GetDamage(int dmg)
  {
    Health -= dmg;
  }
```

Після редагування скрипта PlaerInfo та додавання умови вмикання скрипта PlaerShooting переходимо до його написання:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Networking;

public class PlaerShooting : NetworkBehaviour {
  public int Damage = 25;/*змінна, що відповідає за
нанесення пошкодження*/
  /*траєкторія пошкодження у вигляді променів, які
виходять з камери до об'єкта пошкодження наносимо
натискаючи мишкою на ігровому об1єкті*/
  public Camera cameraTR;// промінь, що виходить з
камери
  private RaycastHit hit; //інформація про об'єкт в
який ми попадаємо променем
  private Ray ray;//інформація про напрямок променя

  void Update ()
  {
    if(Input.GetKeyDown(KeyCode.Mouse0)) //умова
натискання лівої кнопки мишки
    {
      Shoot ();
    }
  }
  void Shoot ()
  {
    ray = cameraTR.ScreenPointToRay
(Input.mousePosition);//промінь пускається з камери у
положення мишки
    if (Physics.Raycast (ray, out hit, 1000))/*переврка
якщо фізично промінь з параметрами що вказані і
радіусом 1000 попадає у об'єкт*/
    {
      if (hit.transform.tag == "Player") /*перевірка чи
обєкт Player*/
      {
```

```
            string id = hit.transform.name; /*записуємо у
строкову змінну ім'я гравця у якого влучили*/
            CmdShoot (id, Damage);
        }
    }
  }
  /*посилаємо на сервер інформацію про гравця у якого
влучили і пошкодження*/
  [Command]
  void CmdShoot(string Id, int dmg)
  {
    GameObject go = GameObject.Find (Id); /*сервер
знахоить гравця з іменем id*/
    go.GetComponent<PlaerInfo> ().GetDamage (dmg);
/*сервер повідомляє гравцю, що зменшилося здоров'я*/
  }
}
```

Протестувавши виконання скрипта можна помітити, що у правому верхньому куті значення стану здоров’я зманшилося у обох гравців, **Ошибка! Источник ссылки не найден.**.

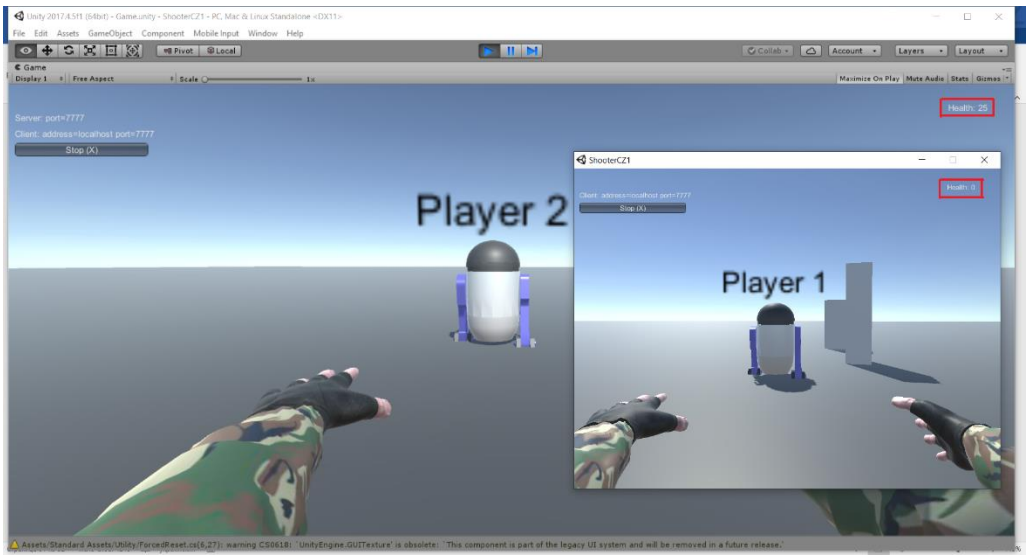

Рисунок 3.23 - Тестування виконання скрипта

3.2.2.4 Знищення та відродження гравця

Знищення персонажу гравця буде відбуватися шляхом оповіщення всіх компонентів, які ми вмикаємо у скрипті PlayerSetup, а не знищенням самого об’єкта. Змінимо тег у камери, що знаходиться у сцені як показано на **Ошибка! Источник ссылки не найден.**, тому що при знищенні гравця потрібно щоб при обертанні мишкою головна камера оберталася у всіх напрямках. Якщо

цього не зробити то при відродженні гравця стандартний скрипт FirstPersonController буде шукати камеру персонажу по тегу. А обидві камери у сцені мають тег Main Camera. Тому змінимо тег Main Camera на Untagged, як показано на рис.3.24. А також вимикаємо всі компоненти Main Camera у вікні Inspector

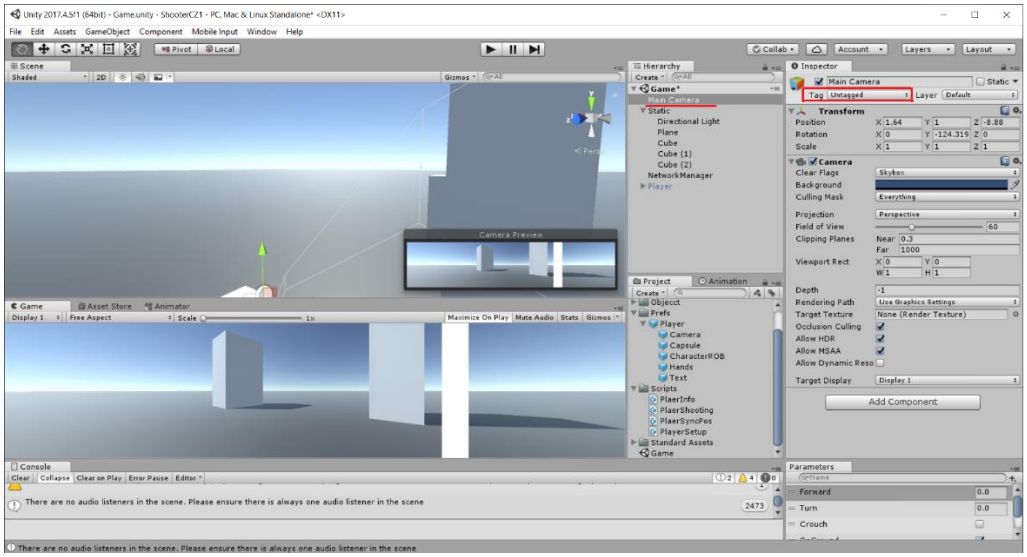

Рисунок 3.24 - Тестування виконання скрипта FirstPersonController

Після цих змін відредагуємо скрипт PlayerSetup:

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using UnityEngine.Networking; //підключаэмо додаткову
бібліотеку

public class PlayerSetup : NetworkBehaviour //Змінюємо
батьківський клас на NetworkBehaviour
{
  // вказуємо ті компоненти що будемо робити для
кожного клієнта унікальними
  public Camera CharacterCamera;//створюємо публічну
змінну типу Камера
  public AudioListener CharacterAudioListener;
//створюємо публічну змінну звуків
  public Transform hands1; //публічна змінна для
відображення руки 1
  public Transform hands2; //публічна змінна для
відображення руки 2
```

```
  public Transform clientModel; //публічна змінна для
відображення тіла
  private bool CanResp; //змінна тригер відродження
  private float timer;
  public float RespawnTime;

  // відбувається ініціалізація гравця
  void Start ()
  {
    EnblePlayer ();
  }

  void EnblePlayer () /*функціяя вивкликається при
відродженні персонажу і при старті гри*/
  {
    if (isLocalPlayer)
    {
      Destroy (GameObject.Find ("Main Camera"));
      GetComponent<CharacterController> ().enabled =
true; //підключаємо контролер

  GetComponent<UnityStandardAssets.Characters.FirstPers
on.FirstPersonController> ().enabled = true;
//підключаємо скрипт
      CharacterCamera.enabled = true; // вмикаємо
камеру, що належить гравцю
      CharacterAudioListener.enabled = true; // вмикаємо
список звуків, що належить гравцю
      hands1.GetComponent<MeshRenderer> ().enabled =
true; // вмикаємо руки якщо це наш персонаж
      hands2.GetComponent<MeshRenderer> ().enabled =
true; // вмикаємо руки якщо це наш персонаж
      GetComponent<PlaerShooting>().enabled = true;
    }
    else
    {
      clientModel.gameObject.SetActive(true); //
вмикаємо тіло якщо це чужий персонаж
    }
    CanResp = false;
    GetComponent<PlaerInfo> ().Health = 100;
/*поповнюємо здоров'я гравця*/
    transform.position = new Vector3 (0, 1.5f,
0);/*координати відродження гравця*/
  }
```

```
  public void DisablePlayer()
  {
    /*функція вимикає всі раніше включені компоненти
якщо гравця було знищено*/
    GetComponent<CharacterController> ().enabled =
false;

  GetComponent<UnityStandardAssets.Characters.FirstPers
on.FirstPersonController> ().enabled = false;
    CharacterCamera.enabled = false;
    CharacterAudioListener.enabled = false;
    hands1.GetComponent<MeshRenderer> ().enabled =
false;
    hands2.GetComponent<MeshRenderer> ().enabled =
false;
    GetComponent<PlaerShooting>().enabled = false;
    clientModel.gameObject.SetActive(false);
    CanResp = true;
  }
  void Update ()
  {
    if (CanResp) /*перевірка тригера відродження*/
    {
      timer += Time.deltaTime; /*відлік часу
відродження*/
      if (timer > RespawnTime) /*перевірка на
перевищення часу відродження*/
      {
          timer = 0;/*обнуляемо таймер*/
          EnblePlayer ();/*відновлюємо персонажа*/
      }
    }
  }
}
```

Після цього потрібно викликати функцію знищення зі скрипта PlaerInfo коли здоров’я становиться менше нуля. Після редагування скрипт PlaerInfo буде мати вигляд:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Networking;

public class PlaerInfo : NetworkBehaviour {
```

```
  [SyncVar] public int Health = 100;/*змінна, що
ввідповідає за здоров'я атрибут [SyncVar] синхронізує
її*/
  public TextMesh tm;
  // Use this for initialization
  void Start ()
  {
    transform.name = "Player " +
GetComponent<NetworkIdentity>().netId.ToString();
    tm.text = transform.name;
  }
  void Update ()
  {
    tm.text = transform.name + "/Health " + Health;
//тимчасовий напис над гравцем для тестування
    if (Health <= 0)
    {
      GetComponent<PlayerSetup>().DisablePlayer ();
    }
  }

  public void GetDamage(int dmg)
  {
    Health -= dmg;
  }
  void OnGUI()/*виводимо здоров'я в інтерфейсі гравця*/
  {
    if (isLocalPlayer)  /*якщо ми є гравцем то на екран
буде виводитися тільке наше здоров'я*/
    {
      GUI.Label (new Rect (Screen.width - 100, 25, 200,
50), "Health: " + Health);
    }
  }
}
```

Потрібно у Prefab на ім’я Player проставити значення змінної RespawnTime три секунди, скомпілювати клієнта і протестувати зміни.

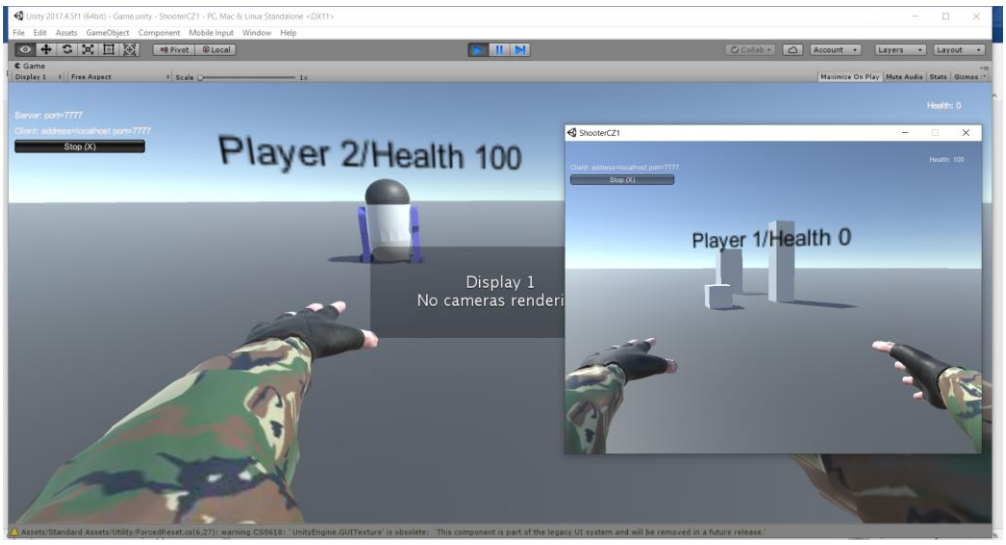

Рисунок 3.25 - Тестування клієнта

З тестування етапу знищення та відродження можна зробити висновок, що все працює коректно рис. 3.25. Після проведення тесту можна відключити показник здоров'я над персонажем. Для цього достатньо у скрипті PlaerInfo позначити, як коментар рядок, де у змінну tm записується показник здоров'я, а саме:

```
//tm.text = transform.name + "/Health " + Health;
```

3.2.3 Налаштування ігрової механіки з урахуванням анімації моделей.

На даному етапі потрібно замінити головний персонаж у грі на його фінальну текстуровану та анімовану версію, а також налаштувати ігрові механіки з урахуванням анімації. Також потрібно імпортувати інші створені моделі для наповнення простору сцени.

Імпортуємо збережений раніше файл CharacterROB_new.fbx у якому знаходиться модель головного персонажу до проекту гри у Unity для цього вибираємо у головному меню Assets/Import New Asset… і обираємо шлях до файлу. Після цього потрібно перетягти на майданчик сцени персонаж, правильно розташувати його відносно старого персонажу у Player. Після цього вказати нового персонажа у скриптах об'єкту Player. Потім у вікні Project виберемо CharacterROB і у вікні Inspector розподілимо створену у 3DS Max

анімацію. Для цього перейдемо на вкладку Animation, як показано на рис.3.25. Виділимо першу анімацію ходьби. Взагалі немає значення яку анімацію виділяти першою. Як видно з рис. 3.25 у поля впишемо: назва анімації – Go, Start – 0, End – 20. Ставимо галочки на Loop Time, щоб зациклити час виконання анімації, а також галочку у Loop Pose для плавного переходу на позицію. Ідентично попередній опишемо інші анімації тільки змінимо назви та поля Start End. Якщо анімація не носить циклічного характеру то галочки Loop Time ставити не потрібно(наприклад анімація стрільби). Після цього натискаємо кнопку Apply(Застосувати)

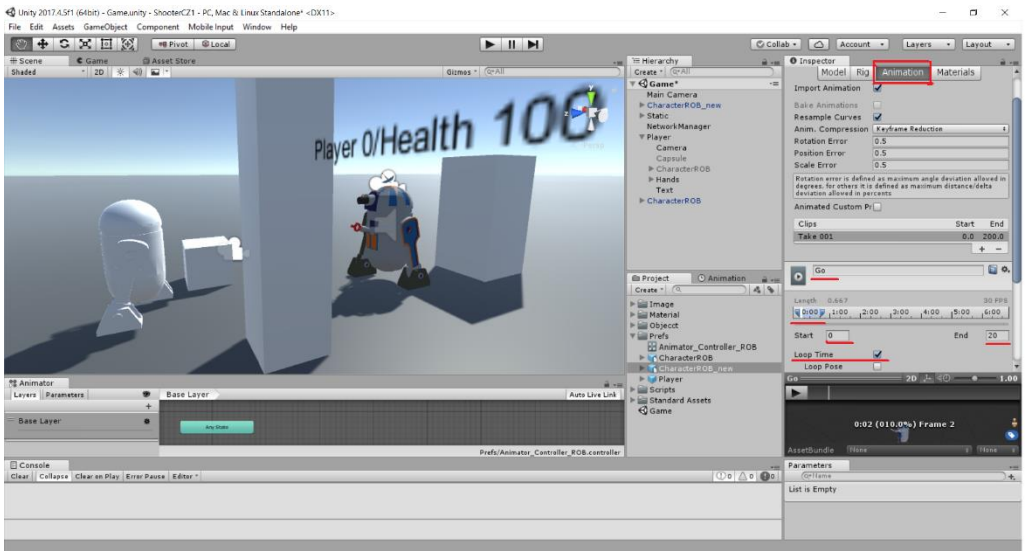

Рисунок 3.25 - Вкладка Animation

Для того щоб анімація програвалась потрібно додати Animator_Controller, Для цього у вікні Project натискаємо ПКМ і у випадаючому меню переходимо Create/Animator_Controller. Одразу перейменуємо його на PlayerAnim_ROB_NEW і вказуємо його відразу в аніматорі моделі персонажу, а саме затискаючи ЛКМ перетаскуєо в поле Controller, як показано на рис. 3.26

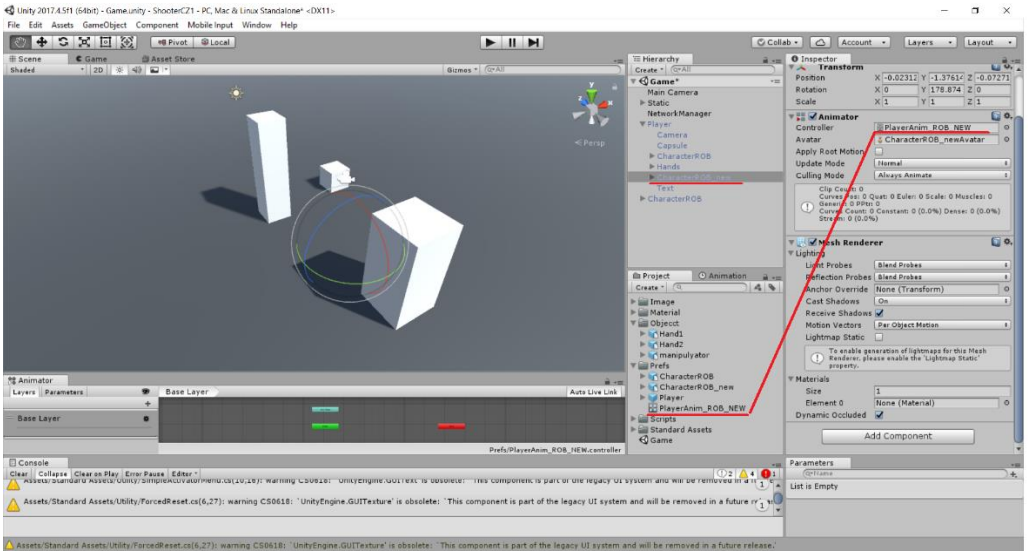

Рисунок 3.26 – Компонент Animator_Controller

Після цього переходимо у вікно Аніматора і відразу додаємо анімацію спокою з назвою StopROB(створену раніше), як показано на рис.2.27. Для цього у дереві персонажу вибираємо анімацію StopROB

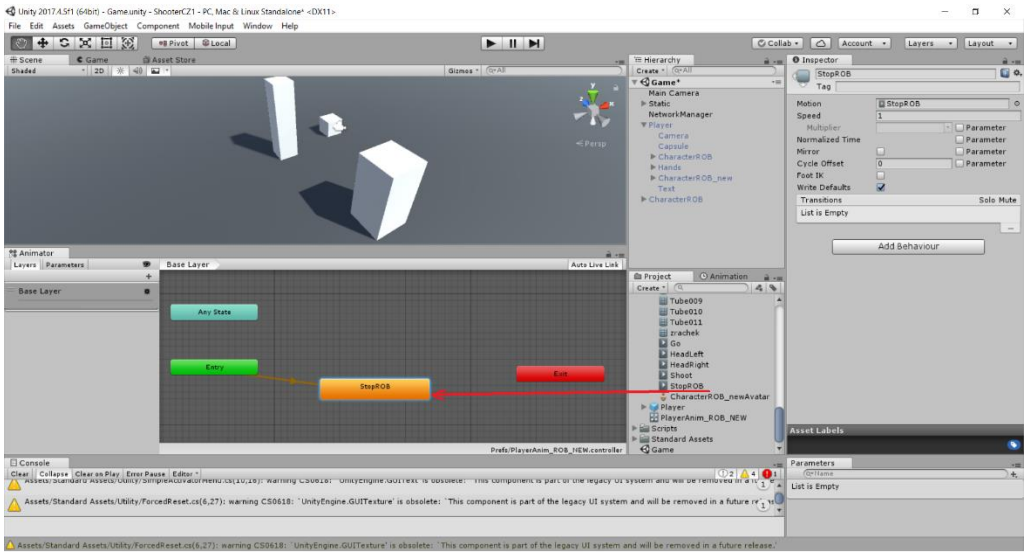

Рисунок 3.27 - Анімація спокою StopROB

При запуску гри додана анімація відразу ж починає програватися.

3.2.4 Створення комфортної стрільби клієнтів

На цьому етапі гравцям потрібно надати можливість комфортної стрільби, для чого потрібно створити приціл та здійснювати створення об’єктів зі сторони клієнта.

Додамо текстуру прицілу в проект гри, для цього перетягнемо раніше створений файл pricel.png з провідника до папки Image, що розташована у вікні Project. Клацнувши ЛКМ на файлі pricel.png у вікні Inspector змінимо значення поля Texture Type на Sprite(2D and UI) **Ошибка! Источник ссылки не найден.**. Після цього потрібно створити новий UI елемент у вигляді картинки для цього натискаємо ПКМ у вікні Hierarchy та вибираємо UI та Image **Ошибка! Источник ссылки не найден.**.

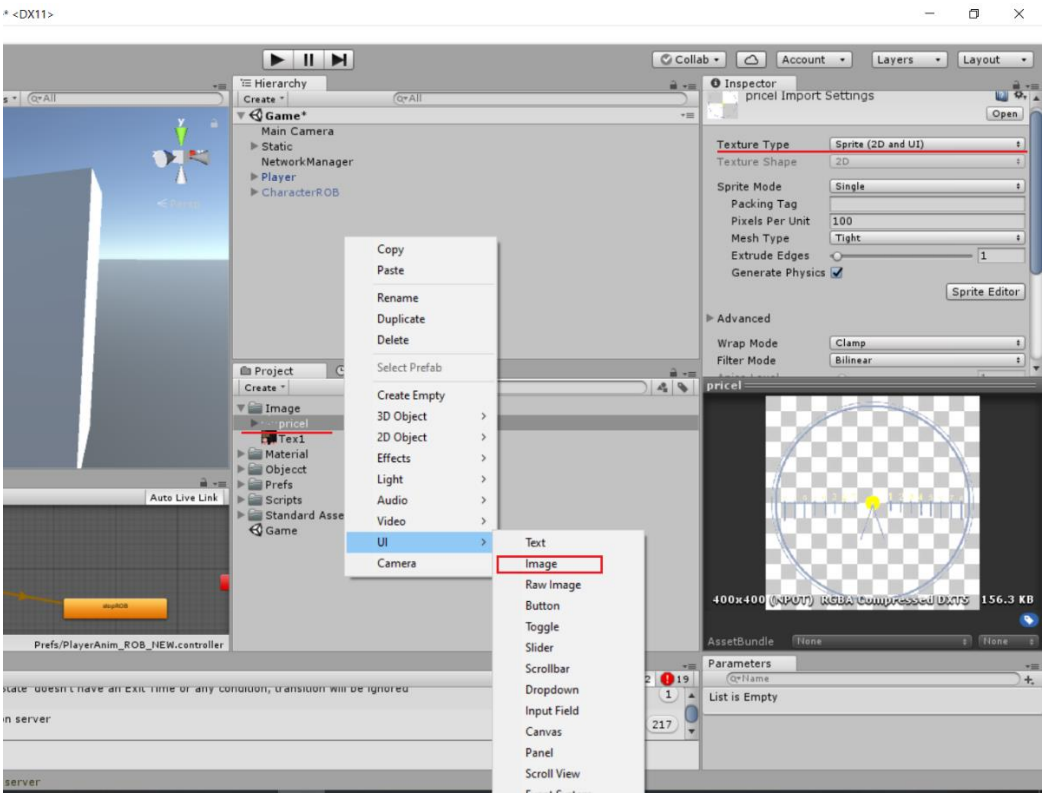

Рисунок 3.28 - Новий UI елемент

Після цього потрібно провести налаштування графічного інтерфейсу у поле SourceImage перетягти картинку pricel.png, прослідкувати щоб у полях Pos X PosY Pos Z стояло значення нуль, довжину та ширину(Width, Heigh) ставимо на власний розсуд в залежності від того які будуть розміри прицілу, якщо колір прицілу не задовольняє то можна задати його на власний розсуд у полі Color **Ошибка! Источник ссылки не найден.**.**Ошибка! Источник ссылки не найден.**

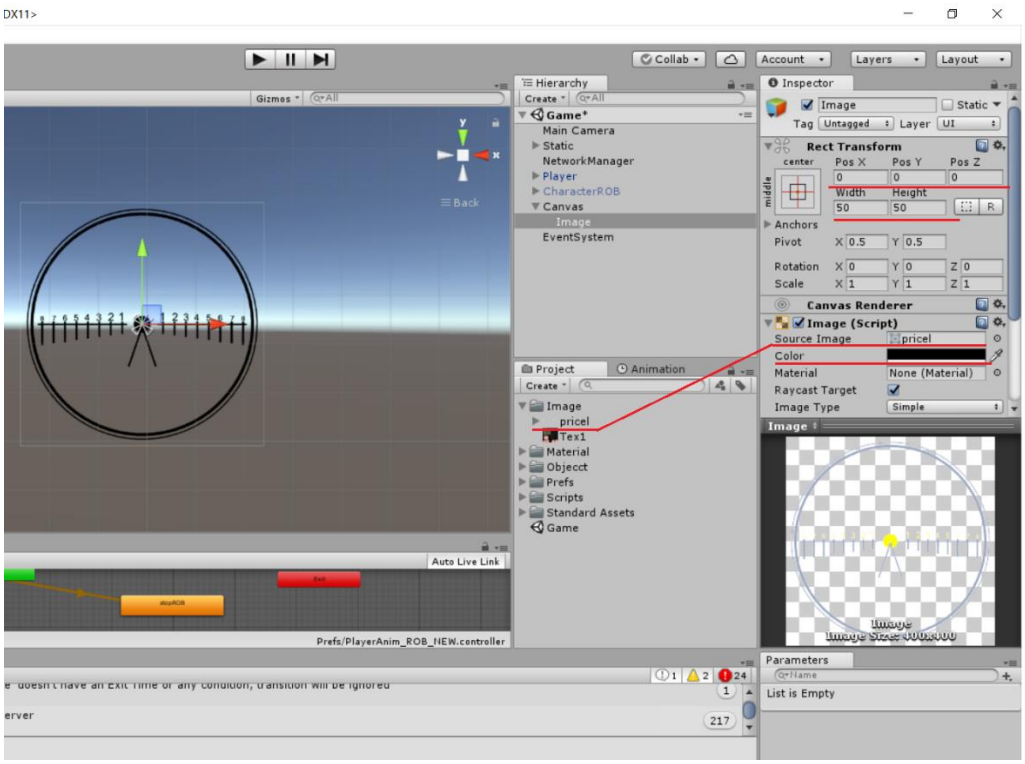

Рисунок 3.29 - Налаштування графічного інтерфейсу

3.3 Наповнення гри об’єктами

3.3.1 Створення навколишнього середовища

Створення навколишнього середовища потрібно почати зі створення землі(terrain). Для цього потрібно вибрати GameObject > Terrain з меню (це також додасть відповідний ассет terrain'a у вікно Project) (рис. 3.30). При цьому ландшафт спочатку буде нічим іншим, як просто великою, плоскою рівниною. Однак, при виділеному об'єкті terrain'a, представлено ряд інструментів, які можна використати для створення будь-яких потрібних ландшафтів (рис. 3.30)

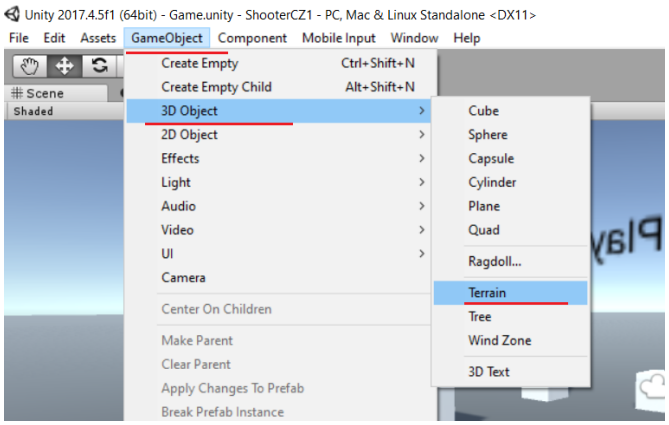

Рисунок 3.30 - Створення навколишнього середовища

На рис. 3.30 підкреслені червоною лінією кнопки зліва на право відповідають за:

- перша кнопка активує інструмент Raise/Lower Height (підвищити/знизити висоту),
- друга кнопка активує інструмент Paint Height, аналогічний інструменту Raise/Lower, але є додаткова властивість для встановлення цільової висоти,
- третя кнопка активує інструмент Smooth Height, усереднює сусідні області. Це пом'якшує ландшафт та зменшує появу різких змін,
- четверта кнопка додає текстури в результаті чого terrain можна розмалювати доданими текстурами. Для цього потрібно натиснути на кнопку Edit Textures потім у випадаючому меню вибрати Add Texture…, після чого у вікні Add Terrain Texture натиснути кнопку Select вибрати текстуру двойним кліком ЛКМ та натиснути кнопку Add для збереження (рис.3.31)
- Четверта та п’ята кнопки працюють аналогічно але додають моделі дерев та рослинності.

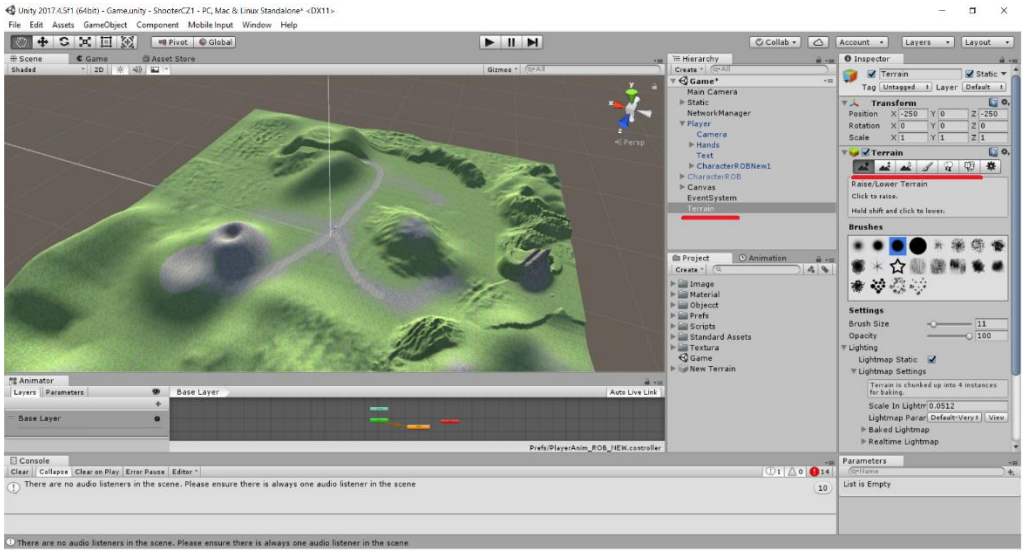

Рисунок 3.31 - Додавання текстур

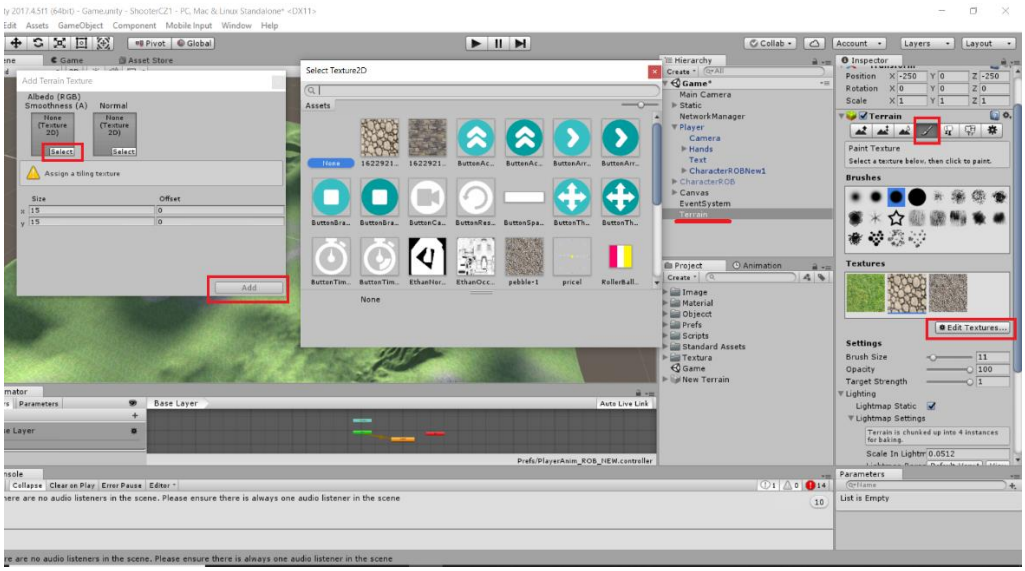

Рисунок 3.32 - Налаштування текстур

Наступний крок це додання раніше створених моделей будівель та споруд для цього натискаємо ПКМ у вікні Project і у спливаючому списку натискаємо Import New Asset і вказуємо шлях до fbx файлу з моделлю, що потрібно додати. Моделі у котрих є анімація анімуємо за тим самим принципом, як і головного персонажа. У гру було додано моделі: бесідки, хатинки на курячих ніжках(анімовано циклічною анімацією), будівля кузні(анімовано циклічною анімацією), споруда вітряка(анімовано циклічною анімацією), герб СумДУ(анімовано циклічною анімацією) **Ошибка! Источник ссылки не найден.**(рис.3.33)

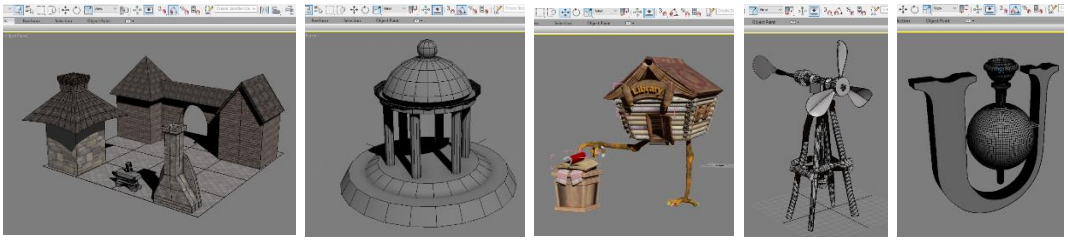

Рисунок 3.33 - Налаштування анімації

3.3.2 Створення фотореалістичного неба

Створення фотореалістичного неба надасть грі більшої привабливості і задасть певний настрій. Небо у грі створюється завдяки Skybox – це 6-

сторонній куб, який промальовується позаду всієї іншої графіки у грі. Кроки, необхідні для створення Skybox:

- Зробити 6 текстур, які відповідають кожній із 6 сторін Skybox та покласти їх у папку Assets (рис.3.34).
- Для кожної текстури потрібно змінити режим обгортки (Wrap Mode) з Repeat на Clamp (рис.3.34)

Цей процес нагадує підготовку розгортки в 3D редакторі, яким він і є по суті. Тому краще картинки неба підготувати(розрізати та видалити фон) у графічному редакторі.

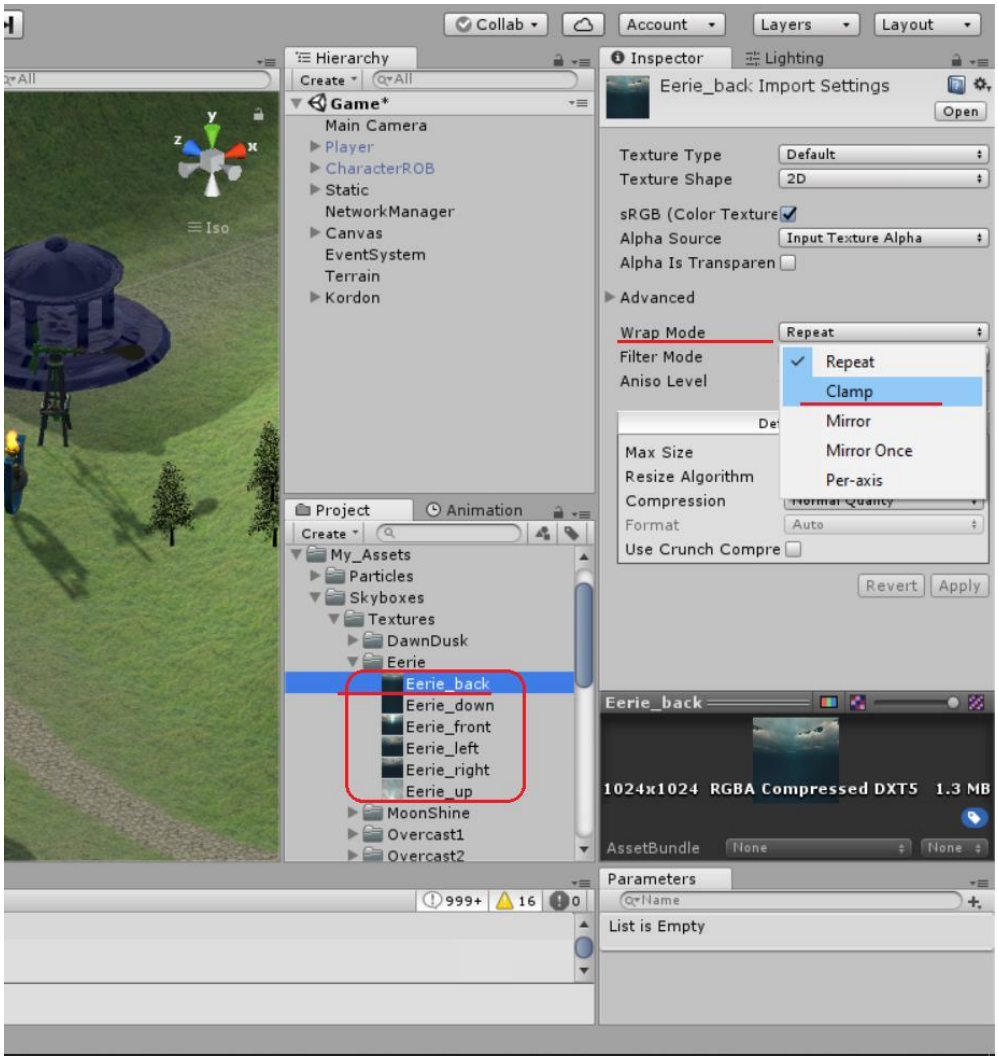

Рисунок 3.34 - Фотореалістичне небо

- Створюємо новий Material, натиснувши ПКМ у вікні Project вибираємо

Create->Material.

- Редагуємо властивості матеріалу у верхній частині вікна Inspector, вибираємо зі списку шейдер Skybox ->6 Sided (рис. 3.35).

- Призначаємо 6 текстур для кожного слота текстури у матеріалі. Робимо це шляхом перетягування кожної текстури з вікна Project у вікно Inspector рис.3.35

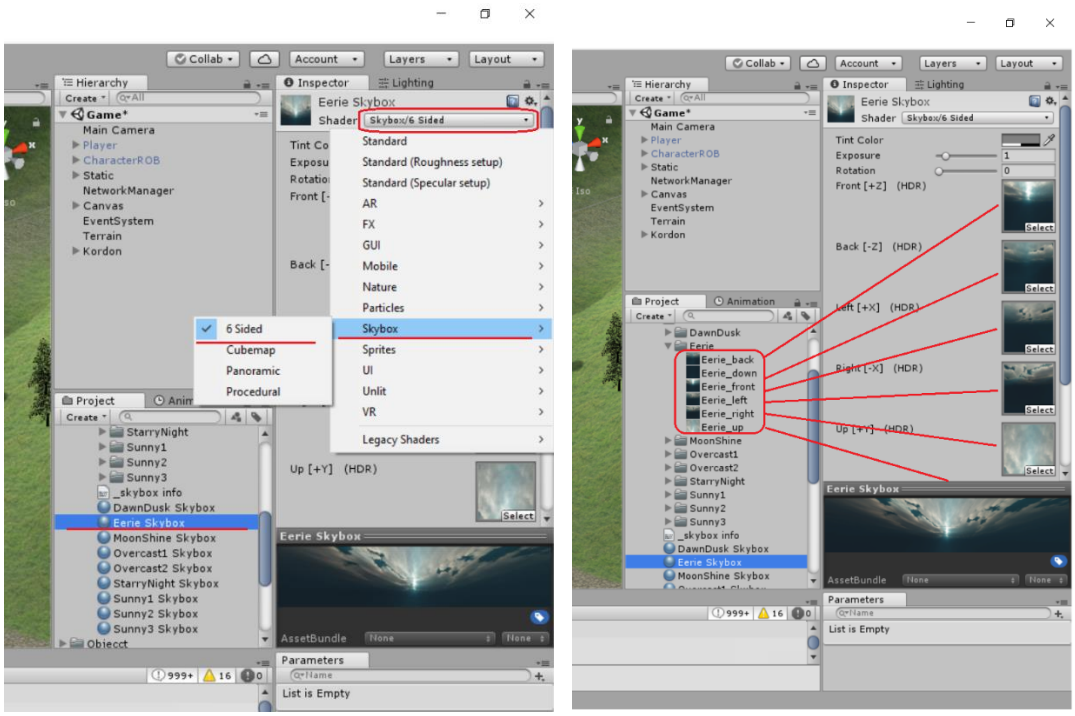

Рисунок 3.35 - Властивості матеріалу

Для прив'язки SkyBox до сцени необхідно зробити наступне:

- У головному меню відкриємо вікно налаштування освітлення для цього вибираємо Window->Lightin->Settings (рис.3.36).

- У вікні, що з’явилося активуємо вкладку Scene і у поле Skybox Material перетягуємо створений матеріал для неба (рис.3.36).

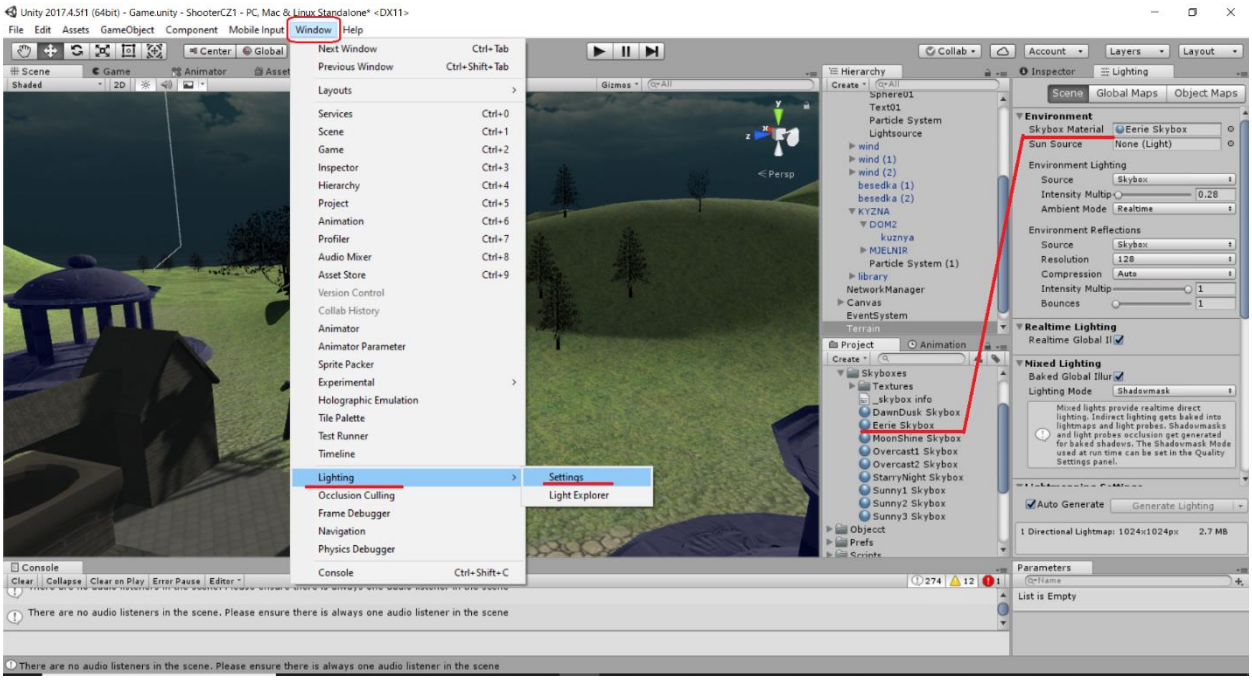

Рисунок 3.36 - Прив'язка SkyBox

3.3.3 Створення аудіосупроводження

Після наповнення сцени моделями залишається задати аудіо супроводження. Звук допомагає гравцю відчути емоції, які неможливо передати іншими способами. Він допомагає пожвавити світ гри, зробити світ більш реалістичним і додає певного колориту. У грі потрібно озвучити такі події як постріли зі зброї, атмосферне озвучування, озвучування об’єкту кузня. Для цього потрібно скачати із спеціалізованих сайтів звуків відповідні звукові файли, що є у вільному доступі і права на їх використання не потрібні, а саме: звук пострілу з пістолета, звук удару молота по ковадлу, атмосферний шум природи.

У проекті гри створюємо папку для звуків під назвою Sound і перетягуємо з провідника раніше скачані файли. Спочатку додаємо звук до об’єкту кузні. Для цього потрібно натиснути кнопку Add Component, вибрати зі списку Audio, потім Audio Source. Після цього з’явиться компонент Audio Source і вже його потрібно налаштувати для гри. У поле Clip Play потрібно перетягти наш файл kuznec із вікна Project, поставити галочки в поля On Awake(програвання при старті гри) і Loop(зациклювання звуку), у полі Spatial

Blend поставати одиницю, щоб звук був об’ємним і затухав при віддаленні від об’єкту (рис.3.37).

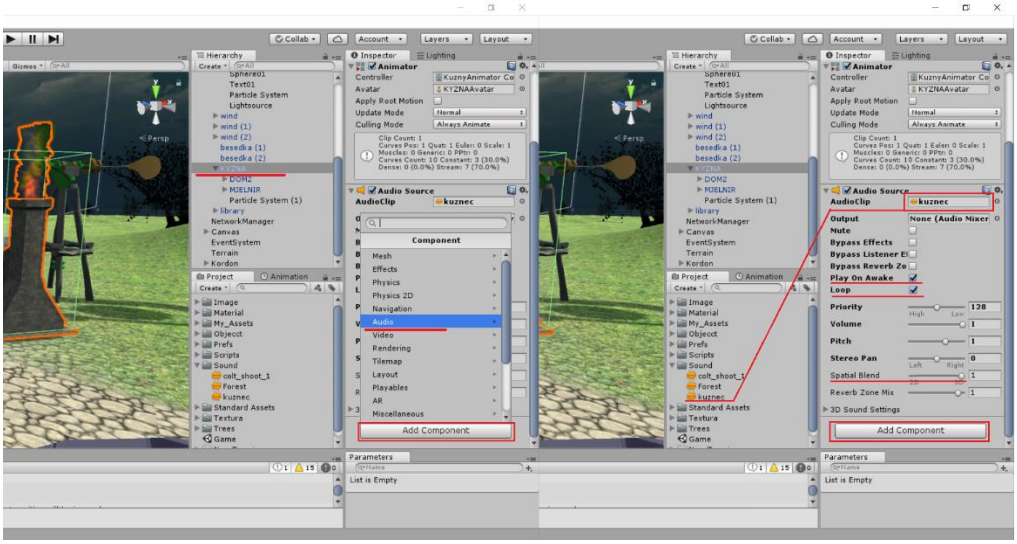

Рисунок 3.37 – Налаштування звуків

Атмосферне озвучування, що знаходиться у файлі Forest потрібно додати до UI елементу Canvas аналогічно попередньому. Налаштування аналогічні попереднім, окрім одного. Атмосферне супроводження у грі застосовується до UI елементу, що завжди присутній у грі поряд з персонажем тому об’ємний звук не потрібен, відповідно у поле Spatial Blend потрібно поставити нуль.

Для озвучування пострілу зі зброї потрібно змінити скрипт PlaerShooting і додати в нього публічну змінну з аудіо та програвати її при натисканні лівої кнопки мишки, а саме:

```
public AudioClip clip; // звук пострілу
GetComponent<AudioSource>  ().PlayOneShot  (clip);  //
програється звук пострілу
```

Після редагування скрипт буде мати вигляд:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Networking;

public class PlaerShooting : NetworkBehaviour {
  public  int  Damage  =  25;/*змінна,  що  відповідає  за
нанесення пошкодження*/
```

```
  /*траекторія пошкодження у вигляді променів, які виходять з камери до об'єкта пошкодження наносимо натискаючи мишкою на ігровому об1єкті*/
  public Camera cameraTR;// промінь, що виходить з камери
  public AudioClip clip; // звук пострілу
  private RaycastHit hit; //інформація про об'єкт в який ми попадаємо променем
  private Ray ray;//інформація про напрямок променя

  void Update ()
  {
    if(Input.GetKeyDown(KeyCode.Mouse0))          //умова натискання лівої кнопки мишки
    {
      GetComponent<AudioSource> ().PlayOneShot (clip); // програється звук пострілу
      Shoot ();
    }
  }
  void Shoot ()
  {
    ray            =            cameraTR.ScreenPointToRay (Input.mousePosition);//промінь пускається з камери у положення мишки
    if (Physics.Raycast (ray, out hit, 10000))/*переврка якщо фізично промінь з параметрами що вказані і радіусом 1000 попадає у об'єкт*/
    {
      if (hit.transform.tag == "Player") /*перевірка чи обєкт Player*/
      {
          string id = hit.transform.name; /*записуємо у строкову змінну ім'я гравця у якого влучили*/
          CmdShoot (id, Damage);
      }
    }
  }
  /*посилаємо на сервер інформацію про гравця у якого влучили і пошкодження*/
  [Command]
  void CmdShoot(string Id, int dmg)
  {
    GameObject  go  =  GameObject.Find  (Id);  /*сервер знахоить гравця з іменем id*/
```

```
    go.GetComponent<PlaerInfo>    ().GetDamage    (dmg);
/*сервер повідомляє гравцю, що зменшилося здоров'я*/
  }
}
```

При роботі зі звуком було розглянуто три способи додавання звуку у гру. Існують більш розгалужені звукові моделі, але всі вони використовують розглянуті вище принципи.

На цьому етап виробництва можна вважати закінченим, потрібно лише провести компіляцію гри і зробити її реліз.

Кінцевий варіант гри має вигляд рис. 3.38

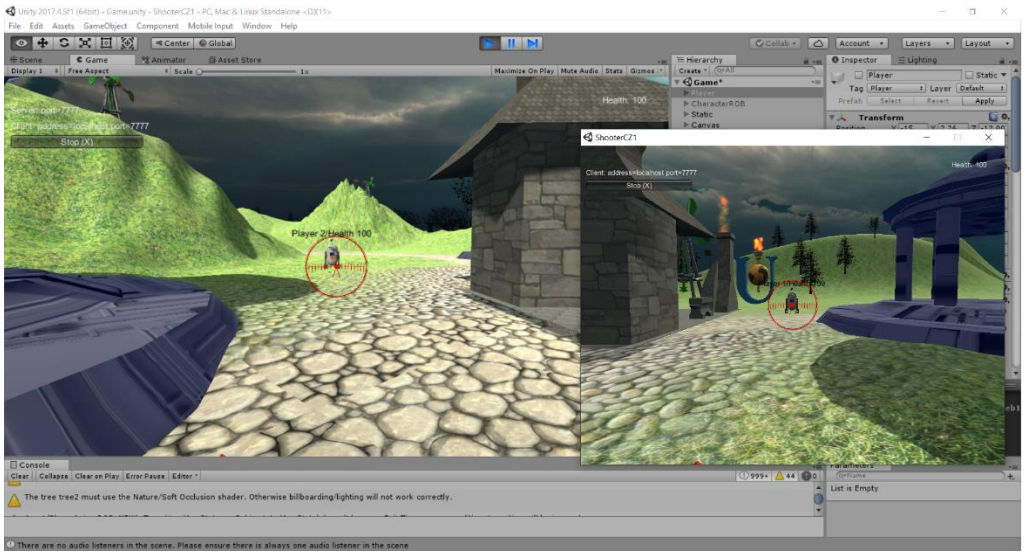

Рисунок 3.38 - Кінцевий варіант гри

4 ТЕСТУВАННЯ ТА ТЕХНІЧНА ПІДТРИМКА

Після релізу відразу починається етап підтримки створеної гри(постпродакшн). У великих компаніях цей етап завчасно планують. Якщо команда не багаточисельна, або як у моєму випадку складається з однієї людини то все залежить від того чи стала гра популярною і чи приносить вона дохід. Якщо гра стала популярною приносить гроші, то її потрібно підтримувати, а саме випускати оновлення(патчі) з усуненням недоліків, та додаткових доповнень. Потрібно враховувати задачі, котрі необхідно реалізувати в оновленні, може статися так, що буде задіяний ланцюжок всіх етапів(наприклад ввод у гру нового персонажу). Тому потрібно враховувати скільки часу і фінансів потрібно витратити. Також потрібно аналізувати фідбек(критичні відгуки) від користувачів і вдосконалювати гру доки це не стане збитковим. Після того, як гра була скомпільована, тестери повинні грати в неї і знайти всі можливі помилки.

Тестування гри виявило суттєвий недолік – Terrain немає меж, персонаж доходячи до краю мапи продовжує рух. Недолік було усунуто шляхом додавання об’єктів Cube з вимкнутими Mesh Renderrer але Box Collider залишилися ввімкнутими, що не дозволяє персонажу доходити до краю карти, він натикається на невидиму стіну (рис. 4.1). Паралельно у гру додано системи часточок що імітують вогонь та дим. Таким чином випущено відповідний патч з усуненням недоліку та реалізовано доповнення.

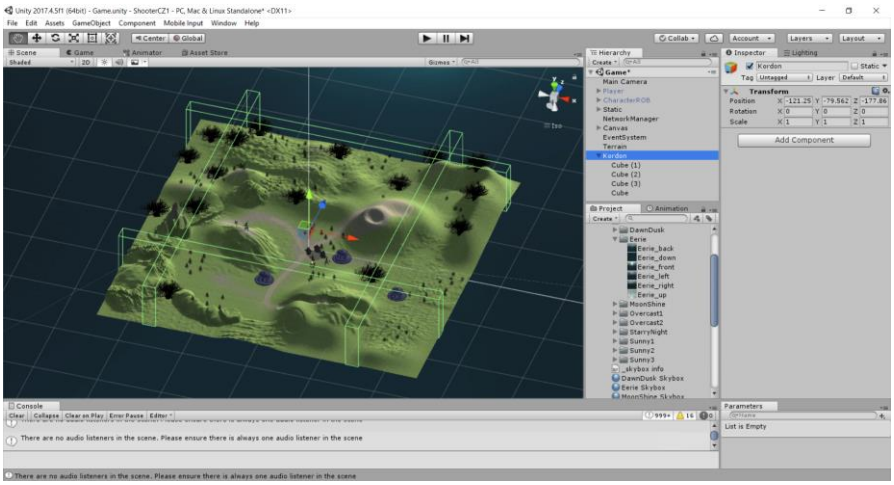

Рисунок 4.1 - Додавання об’єктів Cube

ВИСНОВКИ

Створення гри від першої особи у стилі шутер на основі інформаційних технологій проектування мережевого ігрового програмного забезпечення є унікальним процесом і потребує поєднання таланту і творчих здібностей: соціолога, аналітика, арт-дизайнера, програміста та менеджера. Інтерактивність комп'ютерної гри має гнучкий характер і можливість масштабуватися, підлаштовуючись під запит користувача. На початку розробки здійснено: планування завдань, розподіл їх пріоритету та графік виконання. При створенні мережевої комп'ютерної гри використовувалися сучасні програмні застосунки, що в свою чергу дозволило суттєво спростити окремі низькорівневі програмні процеси, а деякі взагалі прибрати, що скоротило час створення гри. Також це значно підвищило гнучкість та масштабованість гри.

Підводячи підсумок варто зазначити, що комп'ютерні ігри є особливим видом програмного забезпечення з окремими правилами розробки, і включають в себе багато галузей знань, спрямовані на задоволення естетичних та соціальних потреб.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Schell, J. The Art of Game Design: Carnegie Mellon University / J. Schell. – 2008. –518 p.

2. Fullerton, T. Game design workshop: a playcentric approach to creating innovative games / T. Fullerton, Ch. Swain, S. Hoffman. – 2nd ed. – 2008. – 491 p.

3. С. Бондаренко, М. Бондаренко 3ds Max 9. Библиотека пользователя: підручник Питер 2007 р. 640 ст.

4. Технології розробки комп'ютерних ігор: довідник модуля. / В.С. Бреславець.- Х.: «Друкарня Мадрид», 2018. - 162 с

5. Google for games The new frontier of gaming URL:https://games.withgoogle.com/reports/beyondreport/ (дата звернення).

6. Ігрова індустрія в цифрах: скільки українці витрачають на відеоігри URL:https://mind.ua/publications/20222353-igrova-industriya-v-cifrah-skilki-ukrayinci-vitrachayut-na-videoigri (дата звернення).

7. Етапи розробки комп'ютерних ігор URL:https://blog.fugas.space/gamedev-stages/ (дата звернення).

8. Повний посібник 3D моделювання персонажів для непосвячених URL:https://jobs.kevurugames.com/ua/povnij-posibnik-z-3d-modelyuvannya-personazhiv-dlya-neposvyachenih/ (дата звернення).

9. Autodesk 3DS Max 2023 Edit Geometry Rollout (Mesh) URL:https://help.autodesk.com/view/3DSMAX/2023/ENU/?guid=GUID-89D446C3-87F6-4A26-B168-2398E715F87A(дата звернення).

10. Autodesk 3DS Max 2023 Material Editor, Materials, and Maps Maps and Shaders Coordinates Rollout (2D) URL:https://help.autodesk.com/view/3DSMAX/2023/ENU/?guid=GUID-8AE3643F-BDB4-498B-B220-92646FC8A562 (дата звернення).

11. Autodesk 3DS Max 2023 Animation and Time Controls URL:https://help.autodesk.com/view/3DSMAX/2023/ENU/?guid=GUID-818205DD-D58A-495E-BD0C-DC69BDE8DAC2 (дата звернення)

12. Hocking, J. Unity in Action: Multiplatform Game Development in C# with Unity 5 1st Edition / J. Hocking. – 2015. – 352 p.

13. Unity – руководство: Using the Network Manager URL:https://docs.unity3d.com/Manual/UNetManager.html (дата звернення).

14. Руководство Unity/Multiplayer and Networking/Networking Overview/ The High Level API URL: https://docs.unity3d.com/ru/530/Manual/UNetSetup.html (дата звернення).

15. Unity Documentation/The High Level API URL:https://docs.unity3d.com/ru/530/Manual/UNetUsingHLAPI.html (дата звернення).

16. Unity – руководство: Ландшафтный движокТекстури URL:https://docs.unity3d.com/ru/530/Manual/terrain-Textures.html(дата звернення).

17. Руководство Unity/Графика/Подборка уроков по графике: URL:https://docs.unity3d.com/ru/530/Manual/HOWTO-UseSkybox.html (дата звернення)

18. Unity Documentation/GUI URL:https://docs.unity3d.com/ru/2018.4/ScriptReference/GUI.html (дата звернення)