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

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: «Мобільний ігровий додаток «Space Invasion»»

за спеціальністю 122 «Комп'ютерні науки»,
освітньо-професійна програма «Інформаційні технології проектування»

Виконавець роботи: студент групи ІТ.м-91/2 Пархоменко Сергій Володимирович

Кваліфікаційну роботу захищено на засіданні ЕК з оцінкою _____ «___» грудня 2020 р.

Науковий керівник _____ к.т.н., доц., Федотова Н. А.
(підпис)

Голова комісії _____ Шифрін Д.М.
(підпис)

Засвідчую, що у цій дипломній роботі немає
запозичень з праць інших авторів
без відповідних посилань.

Студент _____
(підпис)

Суми-2020

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук
Секція інформаційних технологій проектування
Спеціальність 122 «Комп'ютерні науки»
Освітньо-професійна програма «Інформаційні технології проектування»

ЗАТВЕРДЖУЮ
Зав. секцією ІТП

_____ В. В. Шендрик
«___» _____ 2020 р.

ЗАВДАННЯ

на кваліфікаційну роботу магістра студентові

Пархоменко Сергій Володимирович
(прізвище, ім'я, по батькові)

1 Тема проекту Мобільний ігровий додаток «Space Invasion»_____

затверджена наказом по університету від « 26 » листопада 2020 р. № _____

2 Термін здачі студентом закінченого проекту «_07__» ___грудня___ 2020 р.

3 Вхідні дані до проекту *технічне завдання*_____
_____
_____
_____

4 Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *аналіз предметної області, постановка задачі та методи дослідження, проектування мобільного ігрового додатку, реалізація мобільного ігрового додатку*
_____
_____
_____

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) *актуальність роботи, мета та задачі, функціональні вимоги до додатку, інфографіка порівння прибутку різних ігрових платформ, проектування гри, діаграма варіантів використання, етапи розробки ігрового додатку, засоби реалізації, демонстрація механіки руху персонажу, демонстрація механіки повороту та стрільби персонажу, демонстрація механіки нарахування балів, демонстрація роботи головного меню, висновки, апробація результатів роботи*

6. Консультанти випускної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

Дата видачі завдання _____.

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів випускної проекту	Термін виконання етапів проекту	Примітка
1	Отримання завдання на проект	05.09.20 – 10.09.20	
2	Дослідження предметної області	10.09.20 – 21.09.20	
3	Формування вимог до проекту	22.09.20 – 01.10.20	
4	Планування ІТ проекту	02.10.20 – 15.10.20	
5	Проектування додатку	16.10.20 – 21.10.20	
6	Розробка інтерфейсів додатку	21.10.20 – 07.11.20	
7	Розробка персонажів додатку	07.11.20 – 13.11.20	
8	Розробка ігрового оточення	14.11.20 – 23.11.20	
9	Налаштування звуків	24.11.20 – 26.11.20	
10	Фінальна збірка та компіляція додатку	27.11.20 – 02.12.20	
11	Тестування додатку	03.12.20 – 07.12.20	
12	Представлення роботи	08.12.20 – 21.12.20	

Магістрант _____ Пархоменко С. В.

Керівник роботи _____ к.т.н., доц. Федотова Н.А.

РЕФЕРАТ

Тема кваліфікаційної роботи магістра «Мобільний ігровий додаток «Space Invasion»».

Пояснювальна записка складається зі вступу, 4 розділів, висновків, списку використаних джерел із 31 найменування та 2 додатків.

Загальний обсяг роботи − 124 сторінок, у тому числі 82 сторінки основного тексту, 3 сторінки списку використаних джерел, 25 сторінок додатків.

Кваліфікаційну роботу магістра присвячено розробці мобільного ігрового додатку «Space Invasion» з ізометричною камерою та рисами казуального шутеру.

В першому розділі роботи було проведено аналіз предметної області з дослідженням схожих аналогів ігрових додатків, використання ними механік шутеру або ігрові композиції у мобільних іграх.

Визначення мети проекту, задач та засобів реалізації проведено у другому розділі.

Третій розділ присвячено структурно-функціональному моделюванню.

У останньому розділі продемонстрований процес розробки інтерфейсів головного меню, розробки ігрового інтерфейсу, створення тривимірних моделей оточення, налаштування ігрових персонажій та виконане налаштування пост-обробки гри. Після реалізації проекту проведено тестування на працездатність додатка.

Результатом роботи є мобільний ігровий додаток «Space Invasion», представлений у вигляді .apk файлу.

Практичне значення роботи полягає створенні мобільного ігрового додатку для телефонів на базі Android для цікавого проведення часу.

Ключові слова: мобільний ігровий додаток, ізометричний шутер, казуальний шутер, Unity, гра для Android.

ЗМІСТ

ВСТУП

На сьогодні смартфони інтегрувались у життя сучасної людини досить сильно. Вони використовуються для спілкування між колегами, рідними та друзями, для роботи, розваг або навчання. За своєю функціональністю їх можна сміливо порівняти з портативними кишеньковими комп'ютерами, оскільки вони здатні виконувати складні обчислювальні операції[1]. Також наявність сенсорного екрану дозволила збільшити його роздільну здатність та розмір, що зробило його зручнішим для перегляду фотографій та відео. На окрему увагу заслуговує можливість миттєвого виходу через смартфон в інтернет, оскільки з його допомогою можна отримати доступ до численних інформаційних ресурсів, без необхідності користуватися стаціонарним комп'ютером.

Поруч з практичним використанням смартфонів розвивались сфери розваг та мобільного геймінгу. Вони почали досить швидко набирати популярність після стрімкого розвитку технологічних характеристик гаджетів. Збільшувались значення потужності, обсяг оперативної пам’яті, ємність акумулятору та діагональ екрану. Кількість контенту для розваг досить велика, для доступу до нього створені окремі сайти та додатки, які дозволяють зручного його завантажувати та переглядати.

При цьому досить велику частину сфери розваг для смартфонів займають саме ігри, адже це чудовий спосіб розважитись, поспілкуватись з іншими людьми у рамках гри та підвищити різні навички. В основному є декілька способів отримати гру на власний смартфон: безкоштовно завантажити або сплатити певної вартості за можливість завантаження гри. При цьому жанрів ігор дуже багато, але найчастіше у вершині рейтингу Play Market чи App Store знаходяться ізометричні чи 2D-ігри у категорії шутерів чи платформерів[2].

Актуальність роботи зумовлена великою популярністю мобільного геймінгу. Це досить перспективна сфера, у якій навіть не великі студії або розробники-одинаки можуть досягти значних результатів за цікавий продукт і новітню ідею. Саме тому створення оригінального та цікавого продукту дає можливість для

саморозвитку розробника із шансом зробити успішний крок в IT сферу ігрових розробок.

Об’єктом дослідження виконання роботи є процес аналізу сфери мобільного геймінгу у жанрі шутерів, вивчення їх ігрових механік, особливостей та обмежень. Предметом дослідження є ігрові механіки ізометричного шутеру та їх переваги і вплив на гравця.

Метою проекту є створення гри для мобільного пристою, що буде мати набір функціональних можливостей для підвищення швидкості реакції, тактичних навичок та критичного мислення у гравців.

У майбутньому додаток може бути допрацьований, зокрема можна додати ігрові матеріали для зміни зовнішнього вигляду предметів оточення чи зовнішніх ефектів, таким чином не впливаючи на ігрові механіки можна буде налаштовувати зовнішній вигляд під власний смак, а також це один з напрямків отримання потенційного прибутку від гри[3].

Для досягнення мети та вирішення завдання реалізації мобільного ігрового додатку було визначено інструментарій для розробки, проведено планування та документування проекту. Обране програмне забезпечення, програми тривимірного моделювання, редактор векторної графіки, сучасний ігровий рушій з відповідним проведенням налаштування сцени з ігровими сценаріями дозволить отримати спланований результат та успішно виконати обрану задачу.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Загальний аналіз жанру мобільного шутеру в ігровій індустрії

На сьогодні кількість ігрових додатків для телефонів стрімко збільшується, як у будь-якій сфері розваг є ігри у які варто спробувати пограти хоча б з метою ознайомлення, так і такі продукти які не заслуговують уваги.

Найбільш відомі платформи для розміщення ігрових продуктів створені для популярних операційних систем, для смартфонів на базі Android це Play Market, а для iOS – Apple Store. Вони дають можливість категоризувати ігри та полегшити користувачам процес пошуку та завантаження цікавого для них контенту на смартфоні.

Досить часто першість рейтингу цих площадок займають шутери, адже вони надають можливість досить цікаво провести час, при цьому зняти стрес та підвищити рівень реакції та тактичних навичок[4].

Канонічний шутер має панель здоров’я у вигляді невеликого індикатору на екрані, що відображає загальну кількість здоров’я у персонажу та скільки з них залишилося. Після закінчення всього здоров’я герой помирає, та має можливість відродитися в певній точці і часу ігрового простору, зазвичай на початку рівня. Відновлювати таке здоров’я можна за допомогою спеціальних предметів які можуть приймати різні види в залежності від фантазії розробника[5]. У багатьох нових іграх використовується інша система здоров’я, в якій прямих індикаторів немає. Персонаж може отримати поранення, які, будучи серйозними, будуть відображатися різними ефектами на екрані. Щоб ці поранення пройшли, гравець повинен відсидітися в безпечному місці. Іноді, заради реалізму реалізму, можливість лікувати поранення відсутня взагалі.

1.2 Огляд існуючих ігор зі схожими механіками

На початку розробки власного ігрового додатку потрібно визначити його програмні та функціональні особливості, для цього був проведений аналіз схожих ігор за ігровими механіками, цілями та дизайном оточення і персонажів. До уваги були взяті наступні додатки:

– Brawl Stars від компанії Supercell;

– Alto’s Adventure від інді-студії Snowman;

– Hotline Miami від видавця Devolver Digital.

Brawl Stars

Brawl Stars – це багатокористувацький аркадний шутер з ізометричною камерою від компанії Supercell. Битви відбуваюся командами по троє гравців у команді в режимі реального часу, взявши під контроль одного з безлічі персонажів як сучасного, так і фентезійного світу. Всі вони володіють унікальними атаками, суперздібностями та зовнішнім виглядом, який можна налаштовувати. Кастомізація складається, в основному, з наборів скінів, які, як і герої, купуються за реальні гроші або за внутрішньо ігрову валюту, одержувану в боях[6].

Рисунок 1.1 – Процес гри в Brawl Stars

Для керування героєм достатньо лише двох джойстиків та кнопки для активації супер здібностей. Цього цілком достатньо щоб зручно пересуватись персонажем, при цьому не обов'язково потрапляти саме в те місце де він зображений, достатньо натиснути поруч і джойстик автоматично відобразиться у зоні дотику.

Стрільба реалізована двома способами, при короткому натисканні на кнопку пострілу відбудеться автоматичний вистріл у напрямку найближчого ворога, що зручно коли замало часу для прицільної стрільби під час динамічного бою. Якщо ж потримати стік та рухати його у потрібному напрямку, то буде зображено зону ураження у яку відбудеться постріл, це зручно коли потрібно зробити постріл у напрямку ворога, наприклад із засідки.

Від цього проекту можна почерпнути основні риси налаштування механік роботи камери та правил бою персонажів, адже керування досить приємне та реалізація бою досить комфортна для гри на телефоні.

Alto’s Adventure

Alto's Adventure – гра від студій Snowman. Вона, безумовно, має досить приємний зовнішній вигляд двовимірної графіки з витонченими ефектами паралакса, чудовому саундтреку та ігровому процесу, який легко засвоїти та ідеально підходить для довготривалої відтворення[7].

Також розробники додали цікаву особливість інді-платформерів – короткий курс навчання, який запускається автоматично при першому запуску гри. Це дуже правильний хід, завдяки якому гравець відразу розуміє що йому пропонують робити для досягнення результатів.

Рисунок 1.2 – Процес гри в Alto’s Adventure

Керування грою відбувається натисканням на екран, щоб перестрибнути небезпеку або захопити трохи повітря, для успішного виконання трюків. Додатково зустрічаються бонуси розсіяні по всьому ландшафту, а також монети, які можуть розблокувати постійні посилення гравця. Однією з головних сильних сторін гри є те, що вона не намагається застосувати додаткові функції та механіку, щоб ускладнити процес гри.

Цей проект привертає увагу саме своєю атмосферою, простий сюжет, спрощені цілі та зручне керування надають можливість цілком розслабитись та відчути приємні емоції від гри[8]. Якісна, розслаблююча музика і реалістичні фонові звукові ефекти надають додаткову чарівність всьому, що відбувається на екрані, і задоволенню ігровим процесом. При таких умовах навіть успішне виконання завдань ніби відпадає на другий план.

Hotline Miami

Hotline Miami – комп'ютерна інді-гра в жанрі 2D top-down action. Особливостями гри є вид зверху, елементи стелса, надзвичайна жорстокість, а також

сюрреалістичний сюжет і стильний саундтрек. Гра розділена на кілька розділів, кожен з яких поділена на етапи[9].

На кожному етапі гравець орієнтується в будівлі з точки зору зверху вниз, де метою майже завжди є вбити кожного противника в ній. Іноді вони також повинні перемагати боса в кінці глави або знаходити ключові предмети під час вивчення. Деякі рівні також включають приховані маски, які гравець може знаходити по дорозі, часто в тілах інших вбивць, які не виконали завдання, яке зараз виконує головний герой.

Рисунок 1.3 – Процес гри в Hotline Miami

Штучний інтелект ворогів дещо відрізняється від звичайного «стій роби постріли», змушуючи їх час від часу непередбачувано рухатися, і ускладнює планування ідеального підходу. Щоб запам'ятати це, механіки гри дозволяють гравцеві перезапустити кожен етап, як тільки вони помирають, даючи можливість їм швидко налаштувати свій підхід до кожного завдання за кілька спроб. В кінці кожного розділу гра оцінює продуктивність гравця на основі таких факторів, як швидкість, мінливість та необдуманість, з високими оцінками розблокування нової зброї та додаткових масок.

Розглянутий на перший погляд типовий шутер «біжи та роби постріли», але саме ігрові механіки динамічного бою, легковажність смерті гравця та різноманітність способів проходження рівнів роблять цю гру особливою. Навіть

сьогодні, через 8 років після випуску гри, вона залишається популярною та отримує позитивні відгуки в Steam.

Жанр шутерів досить популярний у сфері комп'ютерного геймінгу. У мобільному геймінгу від досить спрощується для зручності гравців та через технічні обмеження смартфонів, але це не означає що він перестає бути цікавим Загальні механіки успішно реалізуються успішно на кожній з платформ, але спосіб реалізації може відрізнятися.

Розглянувши популярних представників ігор різних жанрів, були обрані основні та допоміжні механіки гри та стилі зовнішнього вигляду та предметів оточення для майбутнього додатку. Було обрано ізометричний вигляд камери для спрощення ігрового процесу, адже гравцю так буде зручніше контролювати оточення та ворогів. Після закінчення кількості здоров'я головного героя рівень можна буде перезапустити, щоб спробувати пройти його знову або покращити результат. Додатково передбачена можливість відновлення здоров’я гравця через особливі предмети.

2 ПОСТАНОВКА ЗАДАЧІ ТА МЕТОДИ ДОСЛІДЖЕННЯ

2.1 Мета та задачі дослідження

У результаті аналізу предметної області було сформульовано мету дипломної роботи: розробка ігрового мобільного додатку «Space Invasion» для розвитку реакції, критичного мислення та тактичних навичок.

Для досягнення мети дипломної роботи був визначений перелік основних завдань:

- проаналізувати предметну область та виконати пошук прототипів;
- визначити методи та інструментарій розробки;
- провести планування, документування робіт з ІТ-проекту;
- спроектувати функціонал мобільного додатку,
- розробити спрайти ігрового продукту;
- розробити механіки бою та поведінку ворогів;
- розробити інтерфейс ігрового продукту;
- налаштувати сцени та розробити ігрові рівні;
- протестувати ігровий додаток.

Мобільний додаток повинен забезпечувати веселе та корисне проводження часу у грі з можливістю покращення швидкості реакції у грі чи тактичних навичок.

Використання ігрового додатку спрямоване на підвищення реакції та критичного видів мислення. Однією з основних задач подібних ігрових додатків є надати можливість гравцеві цікавіше та з більшою користю провести власний час у процесі очікування будь-чого.

Впровадження ігрового додатку «Space Invasion» передбачає такі функції:

- вибір рівня гри;
- проходження обраного рівня гри;
- вибір складності гри;
- зміна спорядження персонажу для гри;
- перегляд рахунку за пройдені рівні.

2.2 Методи дослідження

Під час розвитку технічних наук наукові дослідження відбуваються на двох рівнях: теоретичному та емпіричному. Вони відрізняються повнотою, глибиною і всебічністю дослідження об’єкту; цілями, методами та способами зображення знань; рівнем важливості в них чуттєвого та раціонального пізнання. На емпіричному рівні здійснюються спостереження за об’єктами, фіксуються факти, знаходяться закономірні зв’язки між окремими явищами. На такому рівні аналізу буде досліджено предметну область для кращого розуміння наявних механік у жанрі шутерів, впливу оточення на ігровий процес та обмеження та домовленості яких потрібно дотримуватись у процесі створення гри.

На теоретичному рівні досліджень основною задачею є розкриття суттєвих причин та зв’язків між явищами, а пізнавальною функцією – пояснення явищ у формі законів, теорій, теоретичних систем та системних законів. На цьому рівні домінує раціональне пізнання, тому за допомогою такого методу будуть побудовані декілька моделей: IDEF0[10] для детальнішого розуміння процесу побудови ігрового додатку використано діаграми; UML діаграми[11], а саме діаграма варіантів використання програмного продукту (Use Case)[12], для детального аналізу дій користувачів відносно додатку, та розмежування доступу до певних його частин.

2.3 Вибір засобів реалізації

Для створення ігрового додатку на сьогодні можна виділити два основні та популярні ігрові рушії та пов’язані з ними мови програмування. Вони простіші у використанні, а також через їх популярність є досить велика кількість документації та допоміжної інформації про них:

- Unreal Engine;
- Unity.

Unreal Engine – розробка компанії Epic Games, яка спочатку створювалася для гри Unreal, але після випуску гри розробники швидко зрозуміли, що заробляти кошти можна, ліцензуючи сам ігровий рушій. З того часу з використанням Unreal Engine було створено понад сотні ігор і інших проектів[13].

Як і Unity, Unreal Engine дозволяє створювати продукти для більшості операційних систем, консолей і мобільних платформ. Завдяки підтримці різних систем рендерингу графіки Direct3D, OpenGL і т. д., Підтримці різних аудіосистем і можливостей для онлайн-ігор Xbox Live, Windows Live і т. д., Unreal Engine може використовуватися для створення безлічі ігор, включаючи MMORPG (наприклад на UE був створений Lineage II). Unreal Engine включає набір інструментів розробки (SDK) і редактор[14].

Програмування у цьому рушії реалізовано за допомогою системи візуальних сценаріїв Blueprints. Це повна система сценаріїв ігрових процесів, заснована на концепції використання інтерфейсу на основі вузла для створення елементів ігрового процесу в Unreal Editor. Ця система надзвичайно гнучка та потужна, оскільки надає можливість дизайнерам використовувати практично весь спектр концепцій та інструментів, загалом доступних лише програмістам. Крім того, спеціальна розмітка Blueprint, доступна в реалізації C ++ від Unreal Engine, дозволяє програмістам створювати базові системи, які можуть бути розширені дизайнерами[15].

На відміну від Unity, Unreal Engine має відкритий вихідний код, написаний на C++. Якщо говорити про порівняння і переваги Unreal Engine, нам потрібно розуміти, що Unity більше підходить для мобільних і 2d ігор, а UE4 дозволяє створювати потужну графіку. Epic Games недавно намагалася заманити 2d і мобільних розробників, але статистика як і раніше показує, що вони як і раніше схильні обирати Unity.

Unity - це середовище для розробки комп'ютерних ігор, в якій об'єднані різні програмні засоби, що використовуються при створенні ПЗ - текстовий редактор, компілятор, відладчик і так далі. При цьому, завдяки зручності використання, Unity робить створення ігор максимально простим і комфортним, а мультиплатформеність

ігрового рушія дозволяє розробникам охопити якомога більшу кількість ігрових платформ і операційних систем[16].

В першу чергу, рушій Unity3D дає можливість розробляти гри, не вимагаючи для цього особливих знань. Тут використовується компонентно-орієнтований підхід, в рамках якого розробник створює об'єкти (наприклад, головного героя) і до них додає різні компоненти (наприклад, візуальне відображення персонажа і способи управління ним). Завдяки зручному Drag & Drop інтерфейсу і функціональним графічного редактора рушій дозволяє малювати карти і розставляти об'єкти в реальному часу та відразу ж тестувати кінцевий результат.

Друга перевага рушія – наявність величезної бібліотеки асетів та плагінів, за допомогою яких можна значно прискорити процес розробки гри. Їх можна імпортувати і експортувати, додавати в гру цілі заготовки - рівні, ворогів, патерни поведінки і так далі. Ніякої метушні з програмуванням. Багато асетів надаються безкоштовно, інші пропонуються за невелику суму, і при бажанні можна створювати власний контент, публікувати його в Unity Asset Store і отримувати від цього прибуток.

Третя сильна сторона Unity 3D – підтримка величезної кількості платформ, технологій, API. Створені на рушії гри можна легко перенести між ОС Windows, Linux, OS X, Android, iOS, на консолі сімейств PlayStation, Xbox, Nintendo, на VR AR-пристрої. Unity підтримує DirectX і OpenGL, працює з усіма сучасними ефектами рендерингу, включаючи новітню технологію трасування променів в реальному часі[17].

Саме тому через простоту розробки і крос-платформність Unity був обраний у якості ігрового рушію для розробки.

Також через цей вибір для розробки буде використана мова програмування C#, адже саме її підтримує Unity. Це об’єктно-орієнтована мова програмування, яка відноситься до С-подібних мов. Синтаксис мови схожий з синтаксисом мов C++ та Java [18]. C# було розроблено під впливом вказаних вище мов програмування і ввібрала в себе лише їх переваги та особливості, разом з тим позбулась майже від усіх мінусів своїх попередників[19].

Через те що у якості мови програмування було обрано рушій Unity та мову програмування C# можна виділити два основних можливих середовища розробки:

– Visual Studio Code;

– Sublime Text.

Visual Studio Code – редактор вихідного коду, розроблений Microsoft для Windows, Linux та macOS[20].

До особливостей цього редактору можна віднести:

– VS Code дозволяє розробляти як консольні додатки, так і додатки з графічним інтерфейсом, в тому числі з підтримкою технології Windows Forms, а також веб-сайти, веб-додатки, веб-служби як в рідному, так і в керованому кодах для всіх платформ;

– у редакторі присутні вбудований відладчик, інструменти для роботи з Git і засоби рефакторинга, навігації по коду, автодоповнення типових конструкцій і контекстних підказок;

– продукт підтримує розробку для платформ ASP.NET, Node.js та вважається легким рішенням, яке дозволяє обійтися без повного інтегрованого середовища розробки;

– великим плюсом редактора є підтримка великої кількості мов, таких як C ++, C #, Python, PHP, JavaScript та інших[21].

Sublime Text – це багатоплатформовий текстовий редактор, розроблений для користувачів, які шукають ефективний, але мінімалістський інструмент для редагування коду. Редактор, звичайно ж, простий, в якому відсутні панелі інструментів або діалогові вікна[22].

До переваг цього редактору коду можна віднести:

– текстовий редактор створений, щоб дозволити кінцевому користувачеві легко налаштувати програмне забезпечення на свій лад. Sublime дозволяє налаштовувати безліч функцій, включаючи: прив'язки клавіш, меню, фрагменти, макроси і багато інших. Крім того, змінюйте зовнішній вигляд, налаштувавши свої теми для додатку;

– крос-платформна підтримка - в редакторі доступна на більшості поширених

настільних клієнтів, включаючи Windows, macOS і Linux;

– Sublime з відкритим вихідним кодом, а тому повністю безкоштовний. Але водночас ПЗ також можна купити за бажанням. Важливо відзначити, що безкоштовна версія працює просто відмінно;

– з редактором, ви можете комфортно перемикатися між різними файлами.

– простота у використанні редактора підходить для будь-якого користувача, незалежно від рівня його досвіду.

Для створення спрайтів буде використано Adobe Illustrator. Ця програма використовується художниками-ілюстраторами для створення веб-графіки. На відміну від знаменитого Adobe Photoshop, Illustrator працює з векторними зображеннями, а не растровими. Якщо говорити простими словами, то растрова графіка створюється за допомогою великої кількості пікселів, кожен з яких зберігає свій власний колір. Для побудови векторної графіки використовуються математичні формули, тому зображення складається з примітивних геометричних фігур (кола, дуги, трикутники, прямокутники і інші) [23].

Також ця програма входить до комплексу Creative Cloud від Adobe, а це надає користувачеві доступ до колекції програмного забезпечення, що використовується для графічного дизайну, редагування відео, веб-розробки, фотографії, а також набору мобільних додатків та додаткових хмарних сервісів. Додатково проекти автора зберігаються у хмарному сховищі Adobe, то у разі втрати даних на персональному комп'ютері чи зміні робочого місця буде достатньо підключити профіль Adobe і продовжити роботу без зупинок[24].

Також для створення моделей оточення та персонажів застосовано програмне забезпечення для створення і редагування тривимірної графіки – Blender. З огляду на відкритість вихідного коду, доступність і функціональність пакет отримав високу популярність не тільки серед початківців, а й серед досвідчених розробників. У процесі розвитку програми її обирають у якості робочого інструменту для більш серйозних проектів, що не дивно. Blender є досить потужним 3D редактором, який активно розвивається. Звичайно, поки він не може змагатися з професійними програмами для 3D моделювання. Однак навіть зараз він являє собою відмінну

альтернативу дорогим додаткам і цілком виконує поставлені завдання. Blender - це прекрасний варіант для початківців в 3D моделюванні, а також для тих, хто не має наміру перетворювати комп'ютерну графіку в джерело доходу і має намір творити для себе[25].

Беручи до уваги вище описані програми та їх переваги найкращим рушієм для створення продукту є використання Unity із застосуванням мови програмування C#. Розробка програмного коду буде відбуватися у середовищі Visual Studio Code. Для створення спрайтів буде використана програма Adobe Illustrator, а для створення 3D моделей сцени та персонажів буде використано Blender. Короткий порівняльний аналіз обраного програмного забезпечення наведено в таблиці 2.1.

Таблиця 2.1 – порівняльний аналіз програмних продуктів для розробки мобільного додатку.

Категорія	**Варіанти програм**	**Висновок**
Ігровий рушій	Unity, Unreal Engine	Unity краще підходить для створення мобільних ігрових додатків.
Мова програмування	C#	Ігровий рушій визначив мову програмування
Середовище розробки програмного коду	Sublime Text, Visual Studio Code	VS Code краще працює з мовою програмування C#, адже окрім візуального виділення синтаксису може передбачати прості конструкції
Створення спрайтів	Adobe Illustrator	Дуже потужна програма для роботи з векторною графікою з великою спільнотою користувачів
Створення 3D моделей	Blender	Програма яка містить основні необхідні інструменти для створення 3D моделей, не вимоглива до ресурсів комп’ютера

3 ПРОЕКТУВАННЯ ДОДАТКУ

3.1 Структурно-функціональне моделювання процесу

Для відображення функціонування функціональної системи використовують методологію IDEF0. Цей спосіб дозволяє відобразити як працює система, при цьому завдяки графічному відображенню інформацію вона не потребує використання великої кількості слів та все одно є дуже зрозумілою в описі логічних взаємозв'язків між роботами. Сам процес побудови діаграм має чіткі рамки яких потрібно дотримуватися[26]. На рис. 3.1 зображено основний блок «Процес користування мобільним ігровим додатком «Space Invasion».

Рисунок 3.1 – Основний блок «Процес користування мобільним ігровим додатком «Space Invasion»

На рисунку видно назву головного блоку діаграми. Стрілка яка входить в нього з лівого боку це початкові дані які потрібні для роботи. Вихідні стрілки з правого боку – це вихідні дані які отримані в результаті роботи.

Стрілки «Ігрові механіки» та «Правила гри» це шаблони які контролюють межі роботи системи. При цьому стрілка яка входить в нижню сторону прямокутника під назвою «Мобільний додаток «Space Invasion» та «Пристрій для гри» це механізм. Завдяки ньому користувач запускає гру та взаємодіє з нею.

Таким чином задаються основні параметри процесу, його входи та виходи. Але це тільки загальна схема взаємодії з системою в цілому, при цьому вона діляться на етапи. На рис. 3.2 зображено наступний рівень робіт.

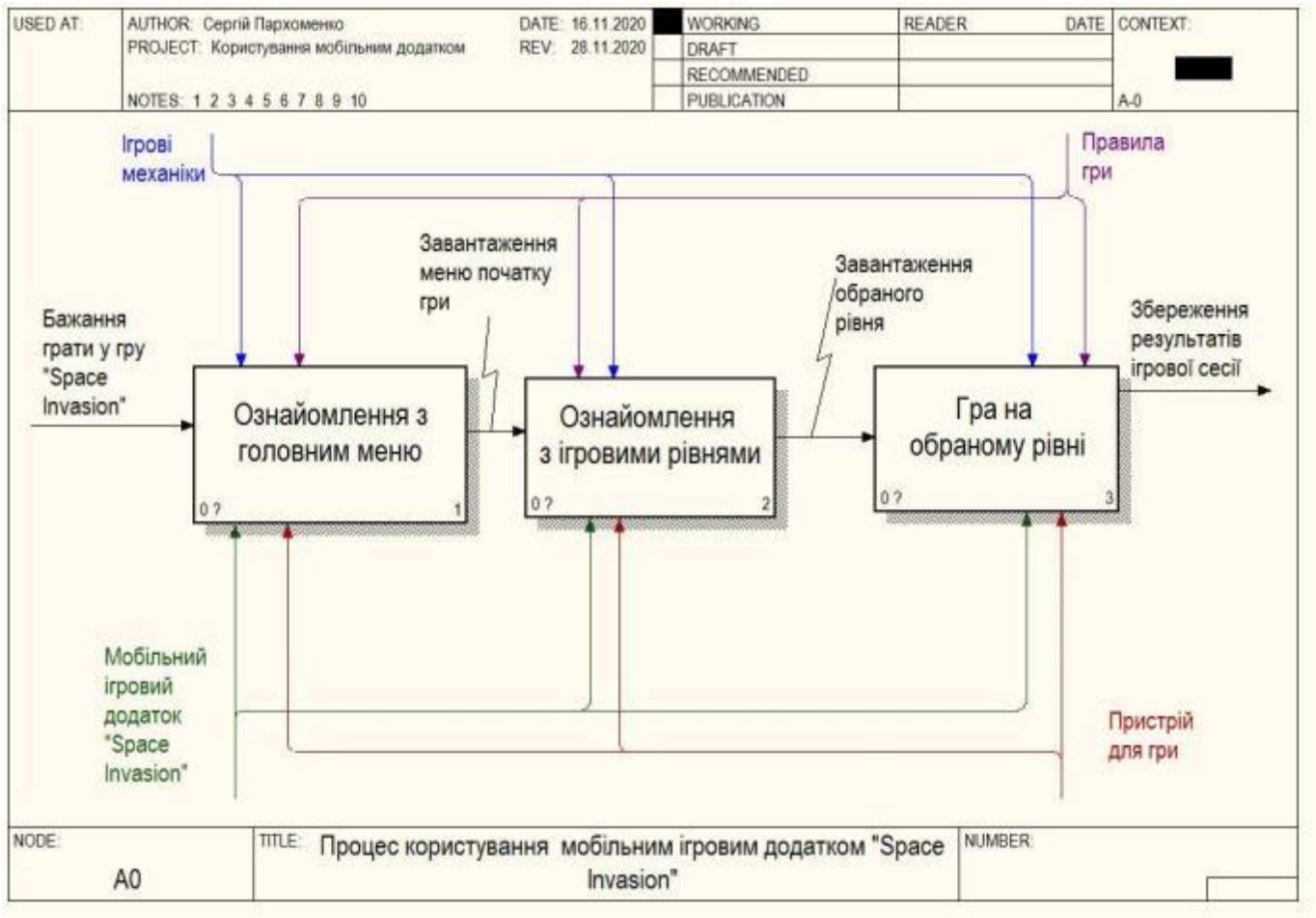

Рисунок 3.2 – Декомпозиція першого рівня головного блоку системи

Декомпозиція будь якого блоку діаграми дозволяє детальніше описати зв’язки виконання робіт в системі. Прийнятною є 3-4 блоки декомпозиції, не більше 6. Якщо кількість блоків більше це одночасно, і перенавантажує діаграму великою кількістю інформації, і означає, що скоріш за все, є процеси які можна декомпозувати на блоки

нижчого рівня. На схемі детально зображено на якому етапі які управляючі елементи і які механізми залучено.

Так для початкової взаємодії з мобільним додатком потрібно ознайомитись з головним меню мобільного додатку, при цьому його зображено на рис. 3.3.

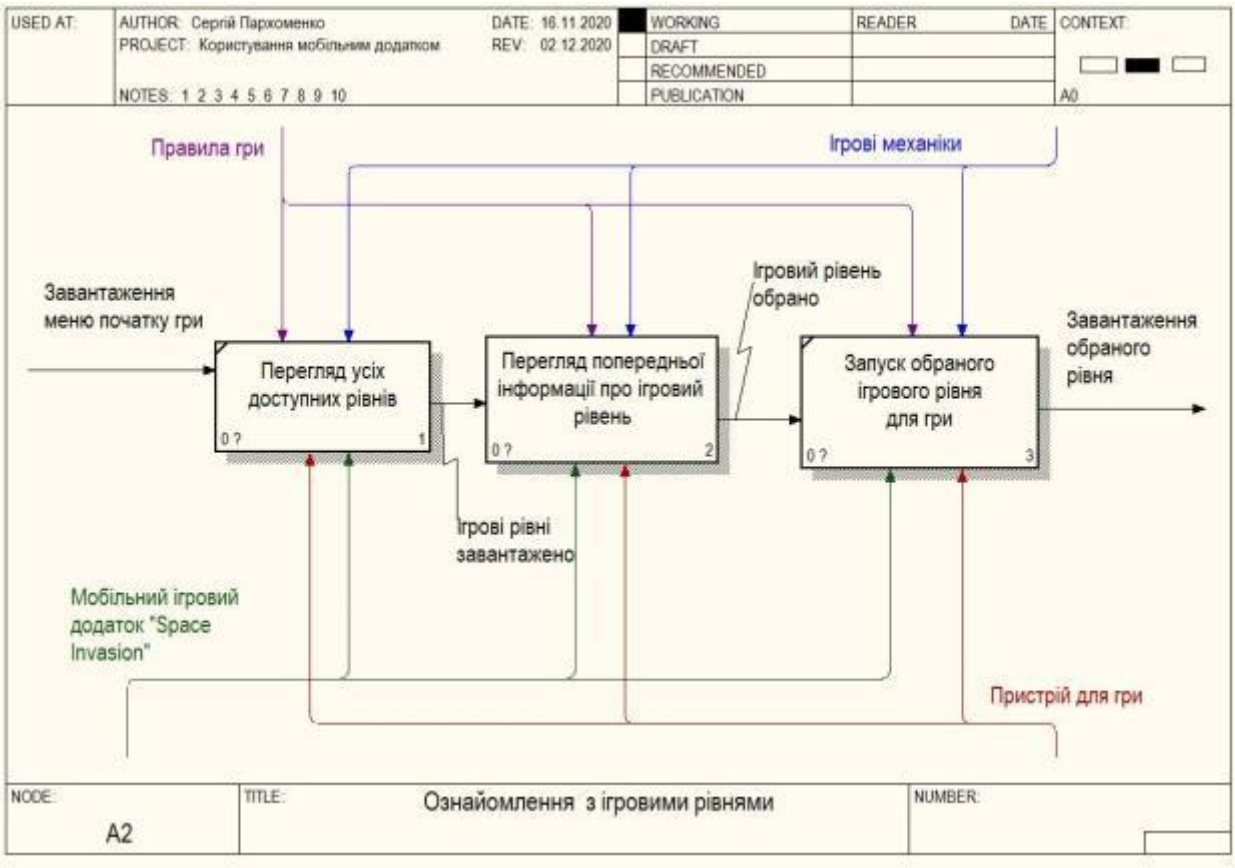

Рисунок 3.3 – Декомпозиція блоку «Ознайомлення з ігровими рівнями»

Важливою частиною побудови IDEF0 діаграми є визначення її мети та точки зору з якої її слід розглядати. Виходячи їх визначення моделювання тих самих процесів може виглядати по різному. Взагалі рекомендовано створювати дві нотації, перша описує процеси «як вони є», друга «як повинна бути». Для створення першого типу нотації проводять інтерв’ю з особами які відповідальні за роботи, щоб краще їх зрозуміти і правильно описати. Друга нотація дає змогу оптимізувати та покращити існуючі процеси[27].

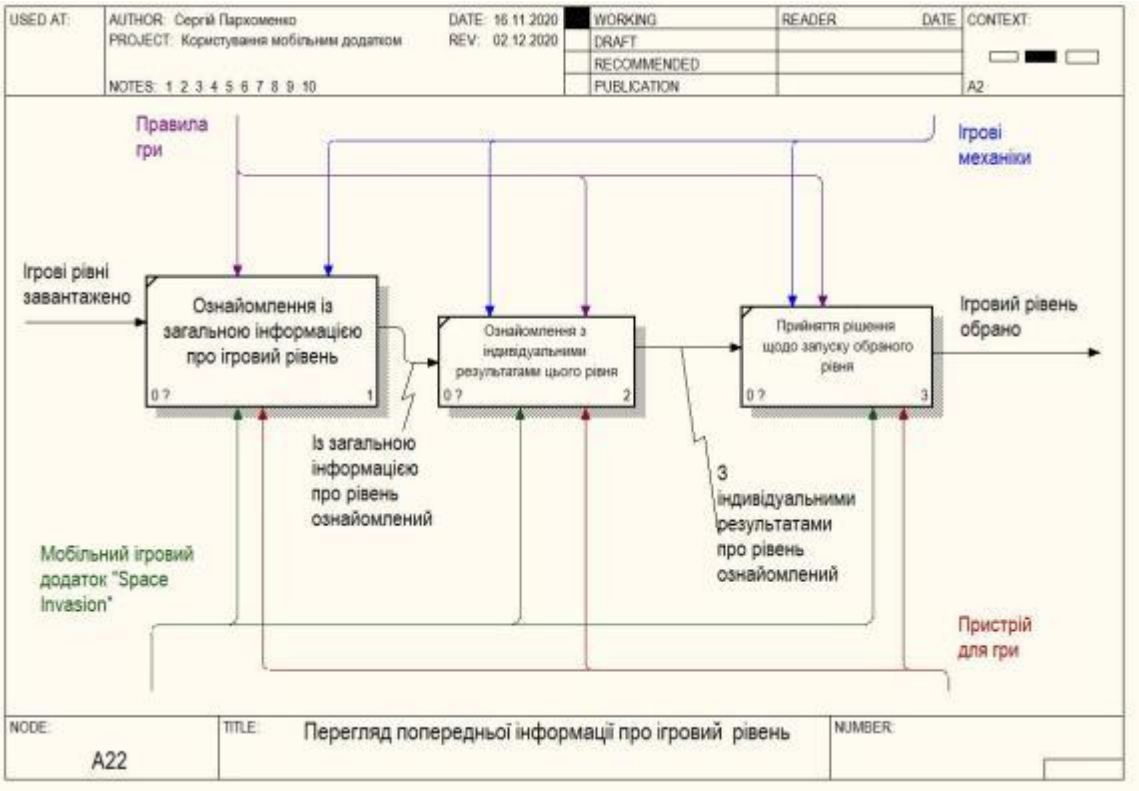

Рисунок 3.4 – Декомпозиція блоку «Перегляд попередньої інформації про ігровий рівень»

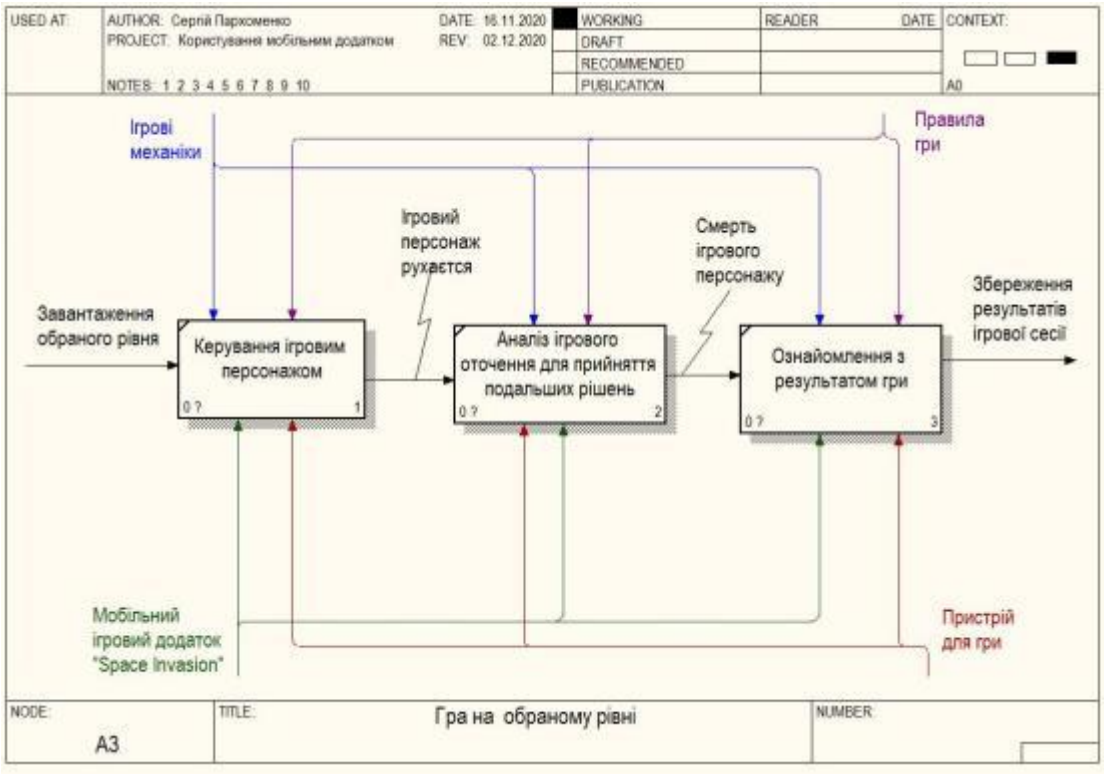

Рисунок 3.5 – Декомпозиція блоку «Гра на обраному рівні»

3.2 Моделювання варіантів використання

Візуальне моделювання в UML можна представити як певний процес порівневого спуску від найбільш загальної і абстрактної моделі вихідної системи до логічної, а потім і до фізичної моделі відповідної програмної системи. Для досягнення цих цілей спочатку будується модель у формі діаграми варіантів використання (use case diagram), яка описує функціональне призначення системи або, іншими словами, те, що система буде робити в процесі свого функціонування[28].

Суть даної діаграми полягає в наступному: проектована система представляється у вигляді безлічі сутностей або акторів, що взаємодіють з системою за допомогою так званих варіантів використання. При цьому актором або дійовою особою називається будь-яка сутність, що взаємодіє з системою ззовні. Це може бути людина, технічний пристрій або програма, які можуть служити джерелом впливу на систему, що моделюється, так, як визначить сам розробник. У свою чергу, варіант використання (use case) служить для опису сервісів, які система надає актору.

У нашому випадку система містить наступних акторів:

– Гравець: особа яка запускає мобільний ігровий додаток на своєму пристрої який відповідає технічним характеристикам для цього;

– Ворог: персонаж який керується ігровим додатком, виступає суперником Гравця під час проходження рівня.

Рисунок 3.6 – діаграма варіантів використання

4 РЕАЛІЗАЦІЯ ІНТЕРАКТИВНОГО ДОДАТКА

4.1 Налаштування інтерфейсу головного меню

Спочатку для створення додатку в ігровому рушії Unity була визначена мова програмування процедур ігрового сценарію. Далі під час створення ігрового додатку було налаштовані базові налаштування проекту. В шаблонах обрано тип гри як 3D, вказано назву проекту та розташування його файлів. Усі описані вище налаштування у вікні створення проекту наведені на рис. 4.1.

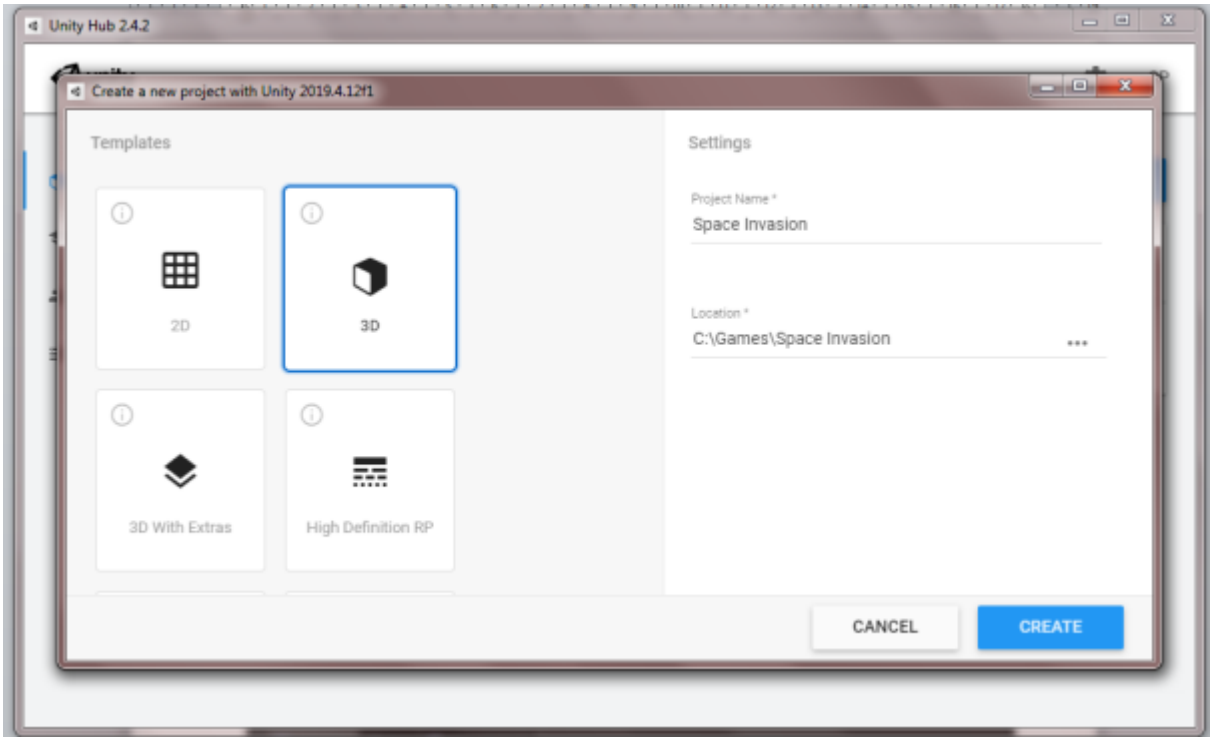

Рисунок 4.1 – Налаштування початкових параметрів проекту

Сцена, яка створена після створення проекту, була використана для головного меню. Зараз вона має вигляд як показано на рис. 4.2.

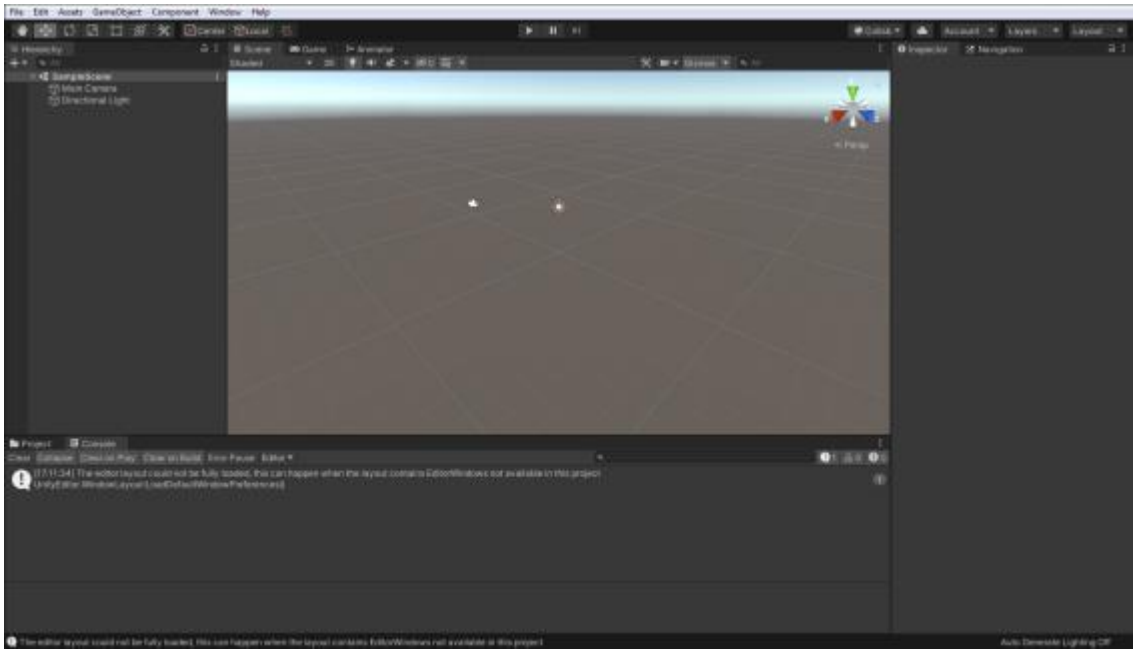

Рисунок 4.2 – Вікно проекту Unity після творення нової сцени

Для початку було налаштовано камеру, яка відображує гравцеві необхідні кнопки, підписи та інші засоби інтерфейсу. Для цього до компоненту (Component) Main Camera, який створений автоматично, додано об’єкт Canvas з групи UI та для нього у компоненті Canvas обрано RenderMode - Space Screen Camera та обрано камеру цієї сцени.

Далі змінено назву Canvas на UI, та додано до нього три GameObject у яких створено по об’єкту типу Panel. Назва у цих об’єктів вказана MainMenu, SettingMenu, SelectLevelMenu. Також для UI додано один об’єкт GameObject з назвою Sound та в ньому створено два AudioSource з назвами Background Music та GameSounds, це контролери для звуків гри. Останнім об’єктом було додано компонент EventSystem з групи UI, він відповідає за обробку натискань на кнопки меню. На цьому етапі ієрархія меню має вигляд як на рис. 4.3.

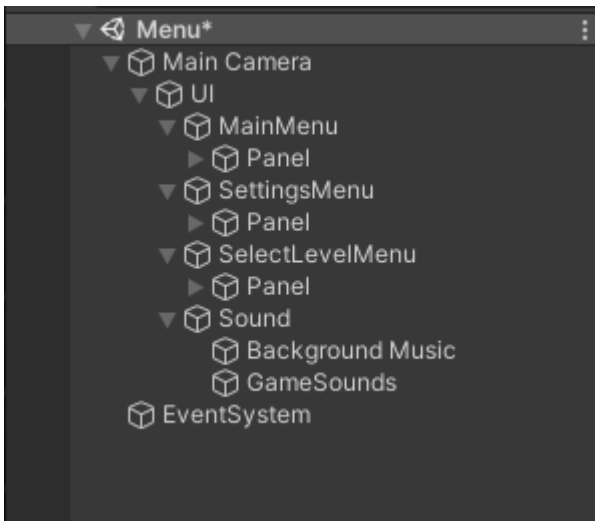

Рисунок 4.3 – Зображення панелі ієрархії (Hierarchy) на цьому етапі проекту

Наступним кроком було створено інтерфейси тексту та кнопок для цих груп меню. Для початку додане фонове зображення головного меню, для цього завантажене зображення з інтернету та додане у компоненті Image об’єкта Panel. Також до Panel додано два об'єкти Text та три Button. У першому об’єкті Text вказано назву гри – Space Invasion, у другому назву поточного меню – «Меню». Назву Button також нова, для цього потрібно знайти вкладений в них об’єкт текст та вказати нову назву: «Початок гри», «Налаштування», «Вихід».

Також раніше створені об’єкти було розташовано на головному меню відповідно до своїх місць, зробити це у компоненті Rect Transform змінюючи параметри Pos X, Pos Y та Anchor Presets. Назву гри зверху по центру, нижче назву меню і потім кнопки. Додатково до проекту додано не стандартні шрифти. Для логотипу це шрифт з назвою Lkdown, для іншого тексту Clickuper, їх завантажено на пристрій та додано мишкою у вікно Project. Unity автоматично опрацював їх та підготував для роботи. Також змінено стандартне оформлення тексту, його розмір, стиль та вирівнювання. Для цього обрано створені об’єкти Text та у вікні Inspector змінено параметри компоненту Text. На цьому етапі розробки меню має наступний вигляд, як зображено на рис. 4.4. Параметри тексту назви гри та оформлення кнопок зображено на рис. 4.5.

Рисунок 4.4 – Зображення інтерфейсу головного меню на поточному етапі

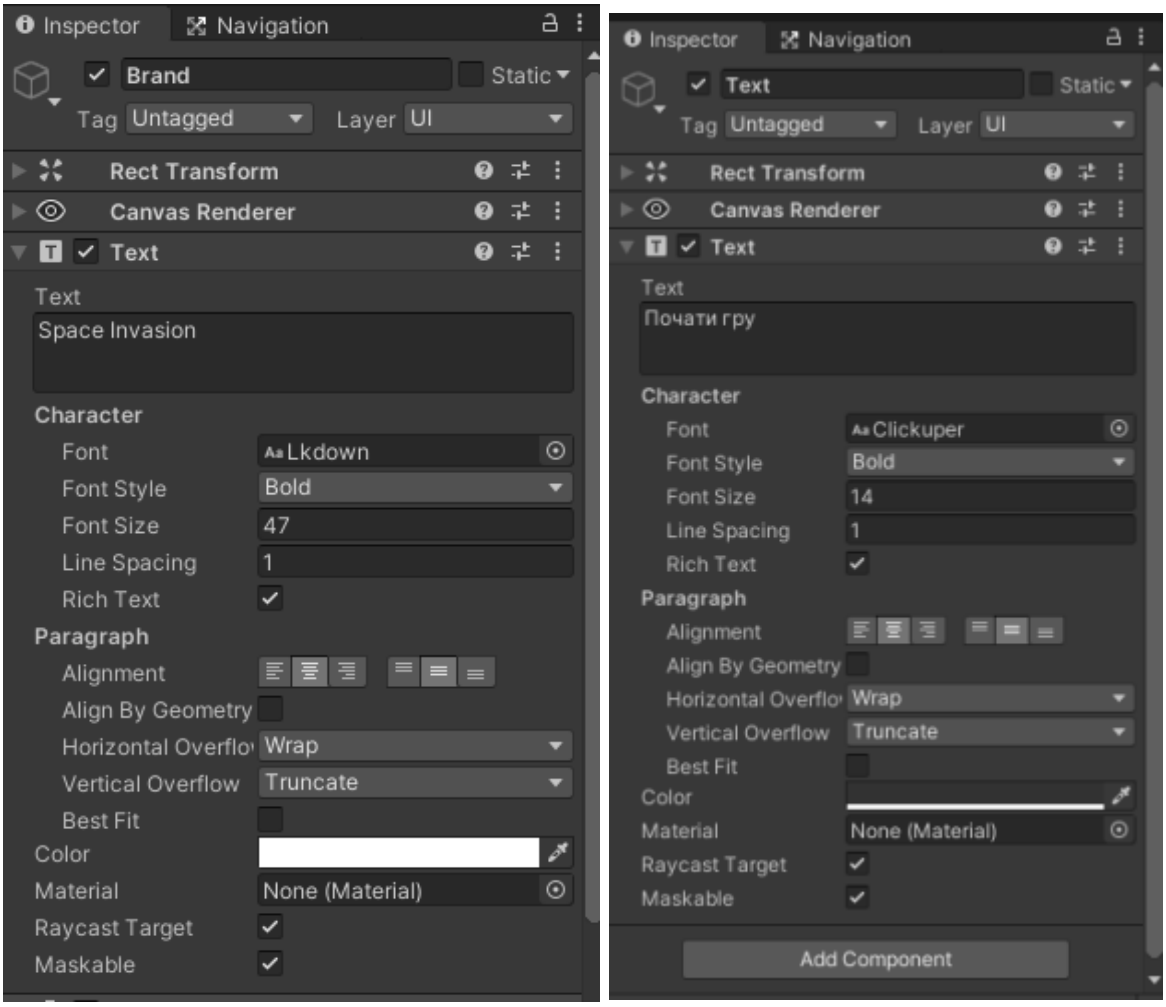

Рисунок 4.5 – Параметри тексту для назви гри (зліва) та кнопок (справа)

Далі створено інтерфейс для меню Налаштування. Для цього приховано об’єкт MainMenu, натисканням на checkbox у вікні Inspector, поруч з назвою об’єкту. Далі у Panel додано три текстових поля (Назва проекту, підпис до слайдеру керування гучністю звуків у грі, та підпис для керування гучністю музики у грі), два Ui -> Slider та кнопку з підписом «Назад» для повернення до головного меню. Після цього розміщено елементи на інтерфейсі та вказано для них якорі прив’язки (Anchor Presets). Зовнішній вигляд вікна Hierarchy та самого інтерфейсу на цьому етапі представлено на рис. 4.6

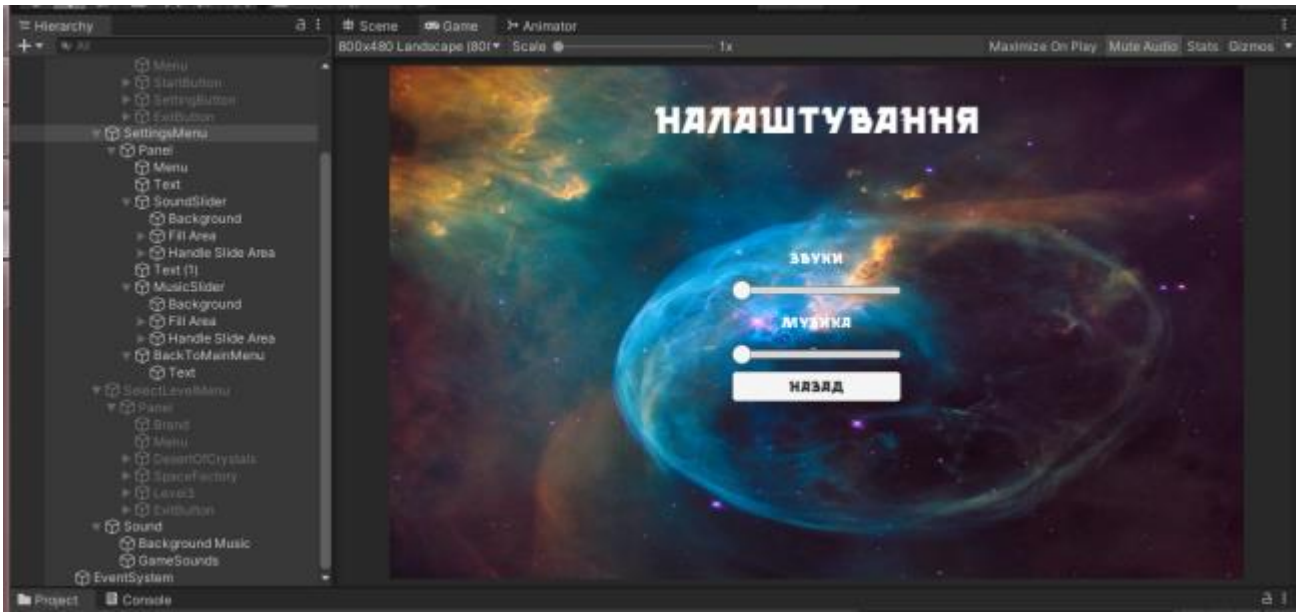

Рисунок 4.6 – Зовнішній вигляд дерева меню «Налаштування»

Останнім інтерфейсом цієї сцени налаштовано вікно вибору ігрового рівня. Для цього зроблено копію об’єкту SettingMenu та його видозмінено: назву гри та кнопку «Назад» залишили без змін, а додали елементи інтерфейсу для вибору ігрового рівня, відображення результату їх проходження і підпис поточного меню.

Для відображення ігрового рівня та його деталей додали Image для картинки ігрового рівня, Image та Text для відображення кращого часу проходження рівня, Text для відображення кращого результату балів за проходження рівня та Button для запуску бажаного рівня.

Кнопки для запуску рівня копійовано з кнопки «Назад» та змінено їх підпис на назву ігрових рівнів, зображення рівня додано у проект після завантаження його з

інтернету. Під зображенням ігрового рівня розташовано зображення іконки годинника у раніше доданому Image для цього. Текстове поле для значення часу розташували з лівого боку, а з іншого боку додано текстове поле для значення балів за ігровий рівень.

Останнім елементом розташовано кнопку для запуску рівня. Після налаштування тексту, розташування всіх елементів та зображень раніше додані елементи об’єднано під одним об’єктом з назвою ігрового рівня. На цьому етапі інтерфейс для вибору ігрових рівнів має результат який зображено на рис. 4.7.

Рисунок 4.7 – Результат налаштування вікна вибору ігрових рівнів

Також елементи вибору ігрового рівня додано у Prefab проекту. Prefab - особливий тип асетів, що дозволяє зберігати весь GameObject з усіма компонентами та значеннями властивостей. Prefab виступає у ролі шаблону для створення екземплярів зберігаємого об'єкту в сцені. Будь-які зміни в Prefab негайно відображаються і на всіх його копіях, при цьому можна перевизначати компоненти і налаштування для кожного екземпляра окремо [29].

Щоб створити Prefab достатньо створити каталог для них у проекті, потім натиснувши ліву кнопку миші перетягнути потрібний GameObject з вікна Hierarchy створений каталог проекту, решту Unity зробить самостійно. Створений Prefab відображено на рис. 4.8.

Рисунок 4.8 – Prefab інтерфейсу вибору ігрового рівня

Подальша розробка ігрового додатку полягає у створенні скриптів для коректної роботи інтерфейсу. Згідно з документації Unity поведінка ігрових об'єктів контролюється за допомогою компонентів, які приєднуються до них. Unity дозволяє створювати свої компоненти, використовуючи скрипти. Вони дозволяють активувати ігрові події, змінювати параметри компонентів, і відповідати на введення користувача яким завгодно способом [30].

Для початку налаштовано перемикання вікон інтерфейсу додатку. Для цього на кнопках головного меню додано функцію для опрацювання натискання на відповідну кнопку. Це зроблено завдяки переміщенню кнопки StartButton з вікна ієрархії у потім у вікно Inspector, у частину OnClick(). Далі додано, під написом Runtime Only, об’єкт меню SelectLevelMenu, потім обрано поруч функцію GameObject – SetActive(bool) та позначено активною позначкою checkbox який з’явився поруч. Результат цих дій відображено на рис. 4.9.

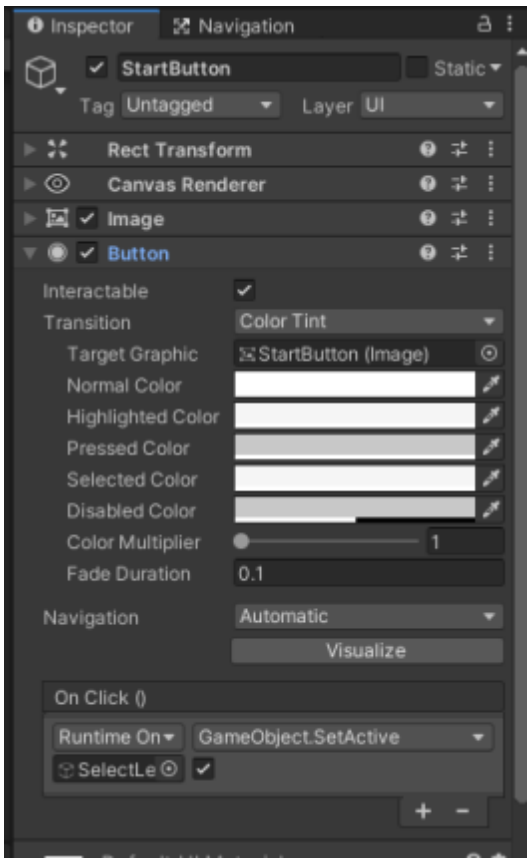

Рисунок 4.9 – Налаштування функції зміни вікон інтерфейсу головного меню

Після цих дій запущено ігровий додаток та перевірено, що при натисканні на кнопку «Почати гру», відображається вікно вибору ігрових рівнів, але йому заважає інтерфейс головного меню. Для того, щоб це виправити у кнопку StartButton, як і попереднього разу, додано функцію SetActive, але вже для об'єкту MainMenu, з не активною відміткою checkbox. Таким чином при натисканні на кнопку «Почати гру» приховає елементи інтерфейсу головного меню, та відобразить інтерфейс меню вибору ігрових рівнів.

Таким самим способом налаштовано кнопку «Назад» в інтерфейсі вибору ігрових рівнів, але вже checkbox позначено навпаки, щоб приховувати інтерфейс вибору рівнів та відобразити кнопки головного меню. Налаштування кнопки «Назад» зображено на рис. 4.10.

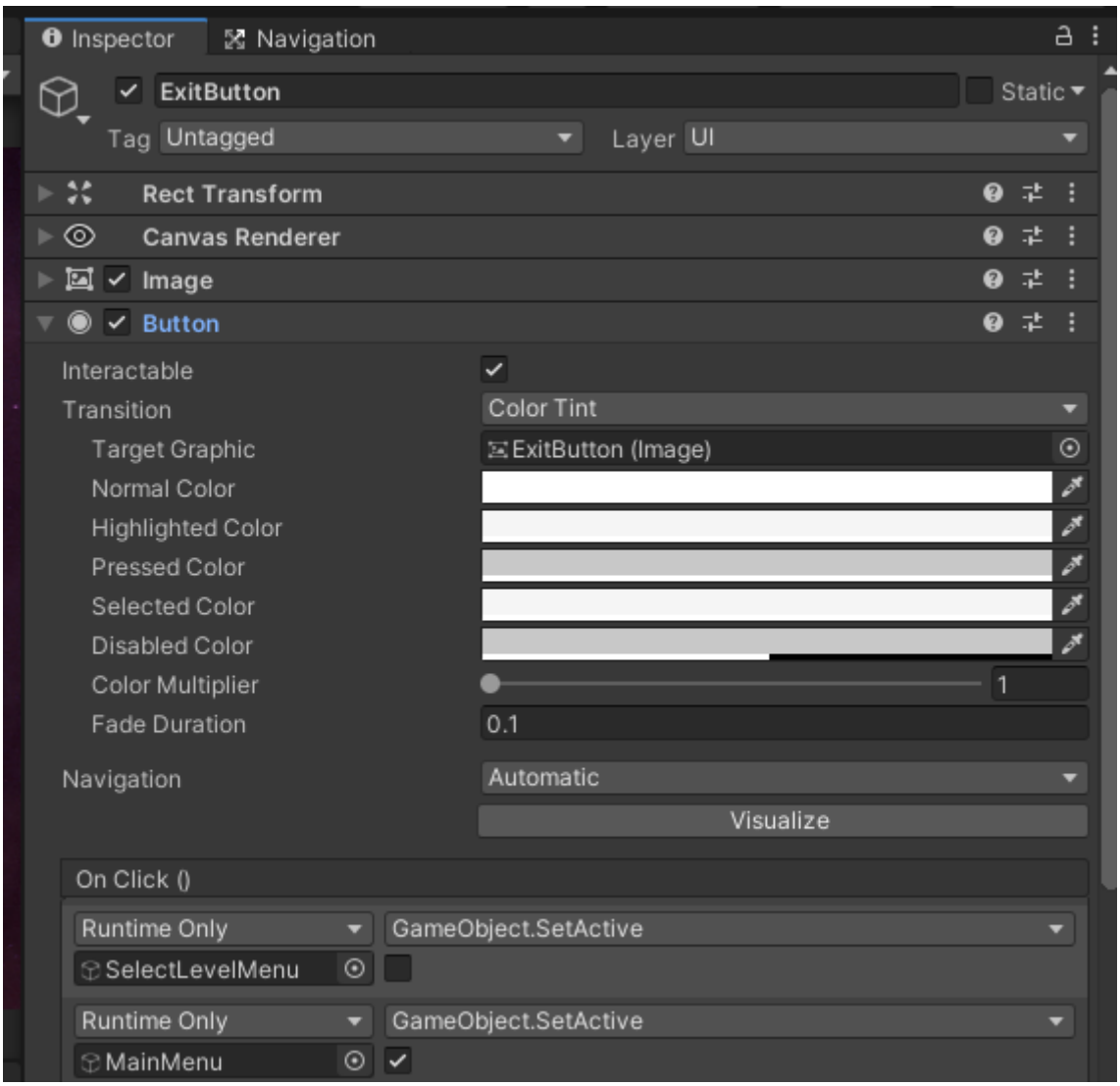

Рисунок 4.10 – Налаштування функції повернення до головного меню

Аналогічним чином налаштовано кнопки SettingButton в об’єкті MainMenu та кнопку «Назад» у об’єкті SettingMenu. Після такого налаштування натискання на кнопку «Налаштування» у головному меню, відкриє вікно налаштувань ігрового додатка, у якому натискання на кнопку «Назад» поверне гравця у головне меню.

Далі у об’єкті SettingMenu налаштовано роботу елементу Slider для зміни гучності ігрових звуків та фонової музики. Для цього у вікні проекту в Unity створено каталог Sound для майбутніх звуків. У цьому каталозі через натискання на праву кнопку миші та вибір Create – AudioMixer створено аудіо змішувач. AudioMixer в Unity дозволяє змішувати різні джерела звуку, застосовувати до них ефекти та виконувати мастеринг [31].

У цьому проекті створено два елементи з назвами GameSounds і SoundtrackSetting. Для їх налаштування вони відкриті натисканням два рази лівої кнопки миші. Потім натиснуто на мікшер Master і у вікні Inspector на параметрі Volume обрано «Expose ‘Volume of (Master)’to script» як на рисунку 4.11 натисканням правої кнопки миші.

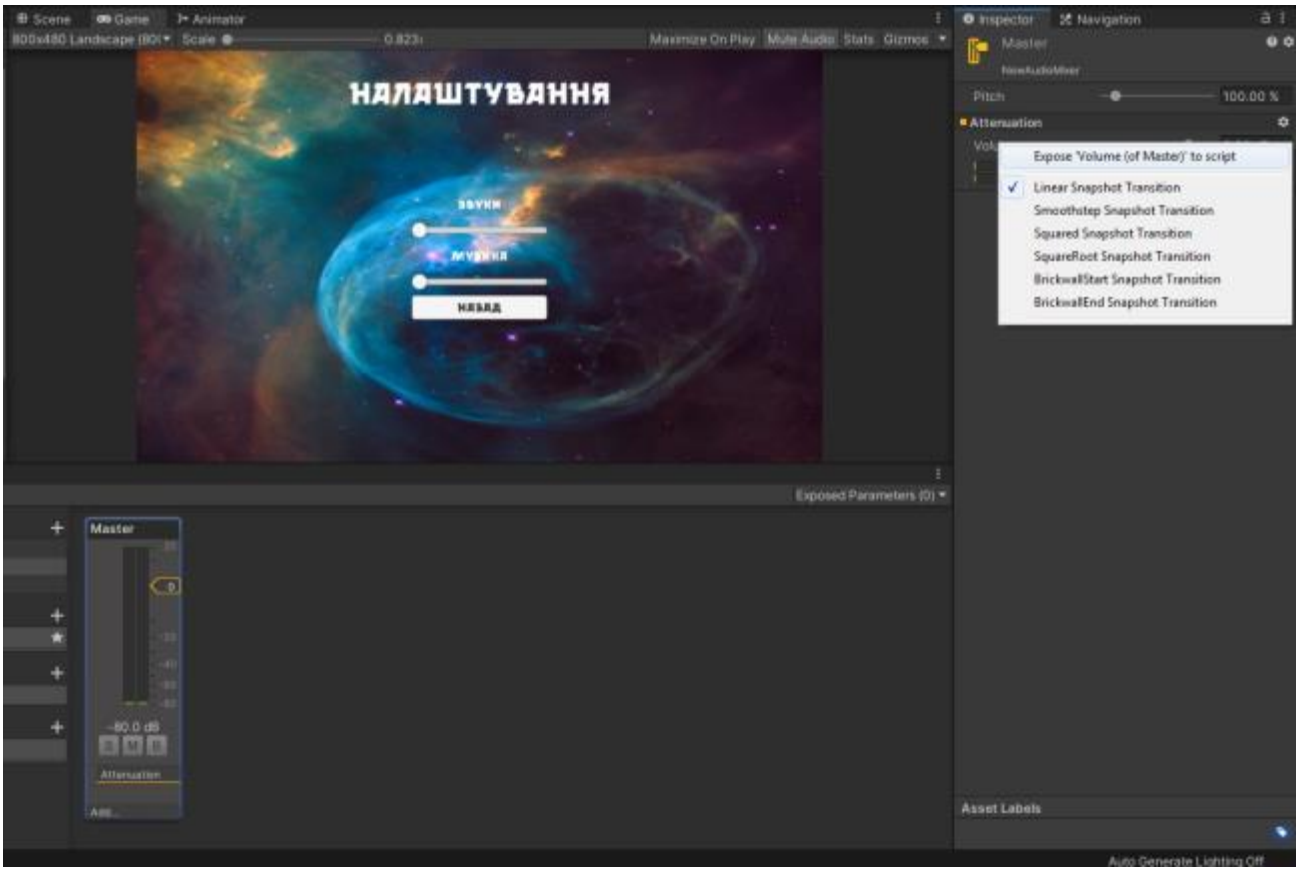

Рисунок 4.11 – Надання скриптам доступу до мікшеру гучності.

Після цього у правому верхньому вікні файлів проекту з’явився параметр у списку Exposed Parameters для доступу до нього скриптам. Далі змінено параметру у кожного мікшера. Для налаштування ігрових звуків вказано назву gameSoundVolume, а для зміни гучності фонової музики – SoundtrackVolume. Налаштування цих параметрів зображено на рис. 4.12.

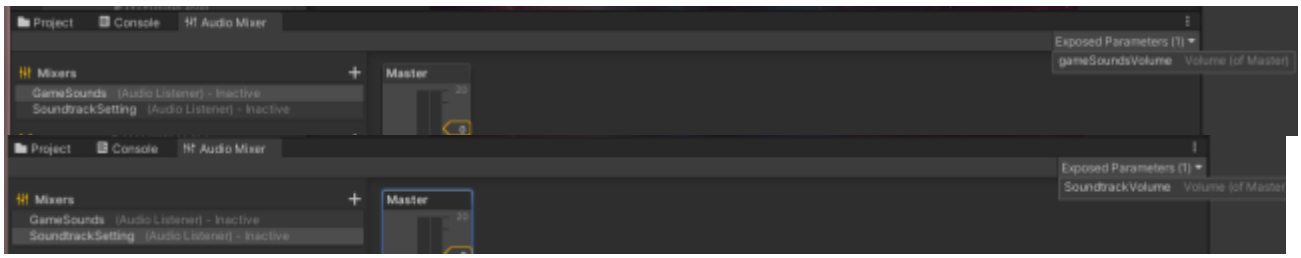

Рисунок 4.12 – Зміна назви параметрів гучності для мікшерів звуку

Мікшери для звуку налаштовані, тому створено власний скрипт мовою C# для зміни параметрів слайдеру та зберігання значення рівня гучності звуків у грі. Для цього створено каталог Scripts у файлах проекту, у цьому каталозі створено новий каталог з назвою SoundScript. Після цього створено два скрипти (Create – C# Script) з назвами GameSoundsSetting та GameSoundtrackSetting, їх повний програмний код наведено у додатку Б.

Після створення скриптів їх додано до відповідних слайдерів та обрано у параметрах скрипту необхідний AudioMixer та слайдер. Параметри слайдеру з необхідними скриптами зображено на рис. 4.13.

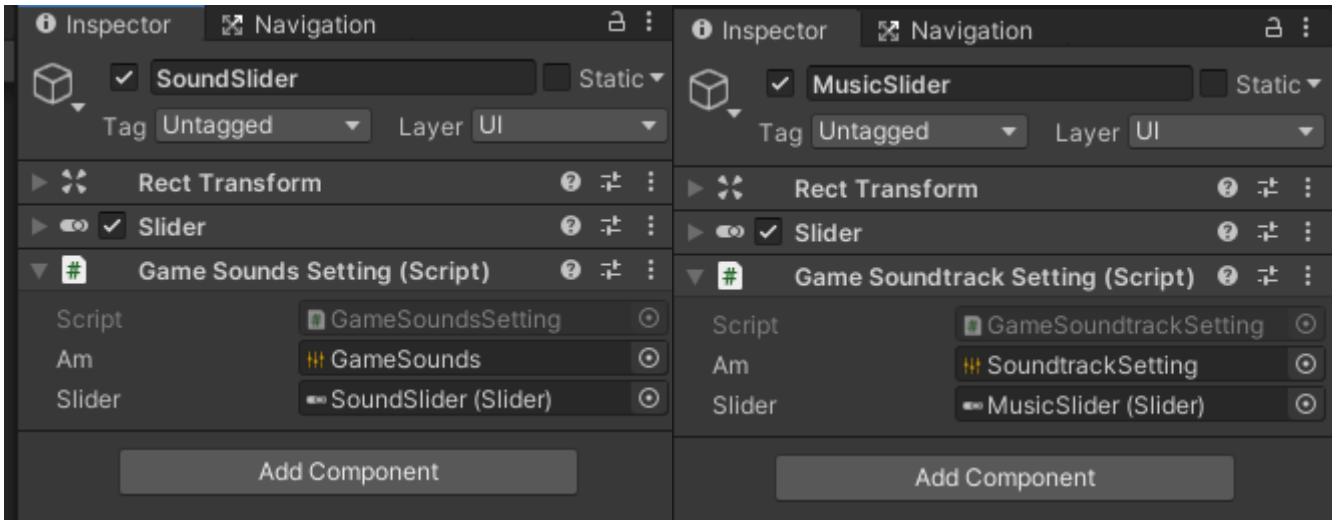

Рисунок 4.13 – Налаштування параметрів слайдерів гучності звуків додатку

Для виконання скриптів у компоненті Slider, у тригері OnValueChanged (Single), вказано посилання на потрібний слайдер та викликано метод з відповідного скрипту. Детальніше це зображено на рис. 4.14.

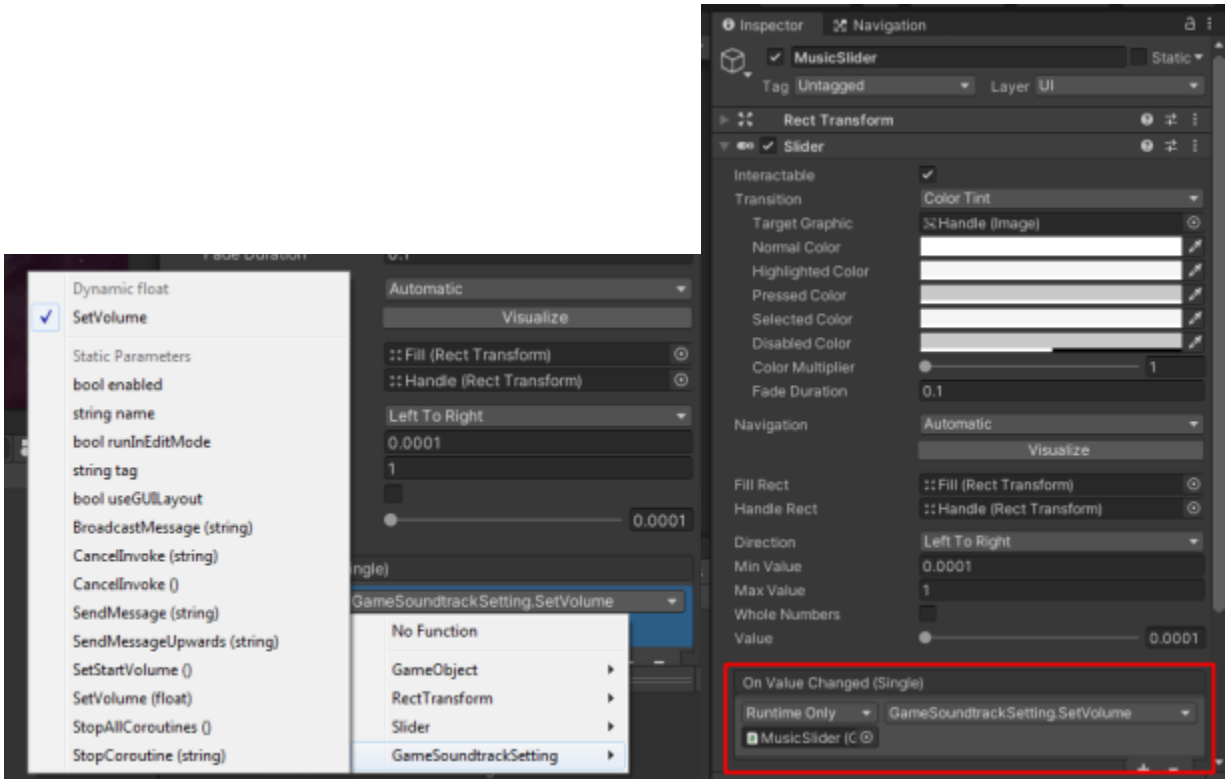

Рисунок 4.14 – Встановлення методу для зміни значення слайдеру

Для встановлення початкових значень слайдерів гучності ігрових звуків створено новий скрипт з назвою InitializeOnLoad, програмний код якого наведено в додатку Б. Для виконання скрипту додано його, як компонент, до об’єкту MainCamera та вказано відповідні параметри. Таким чином після запуску ігрового додатка з файлів гри буде завантажено значення гучності для мікшерів ігрових звуків та фонової музики. Зображення налаштування параметрів скрипту об’єкта MainCamera наведено на рис. 4.15.

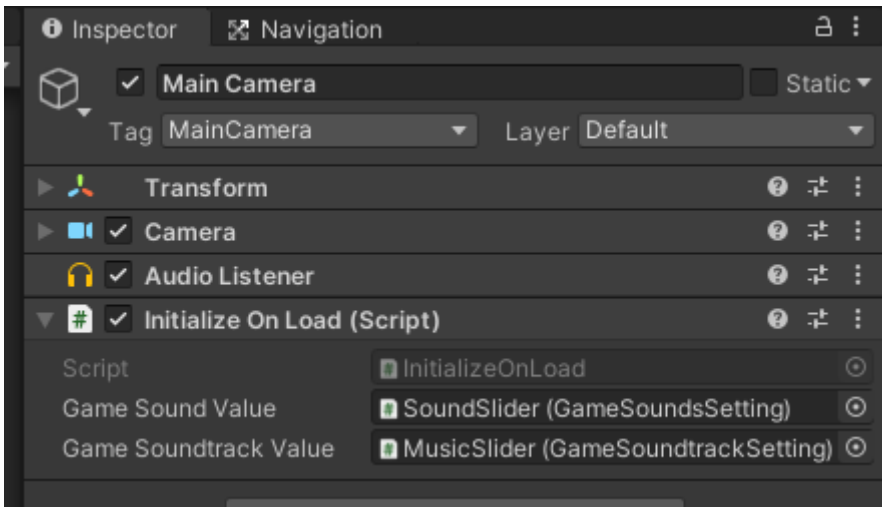

Рисунок 4.15 – Налаштування скрипту

Після створення інтерфейсу головного меню створимо перший ігровий рівень. Для цього створено нову сцену з назвою DesertOfCrystal. Після її створення до неї додано об’єкт Terrain (3D Object – Terrain). Це ландшафт по якому пересувається гравець. Після цього у його компоненті Terrain, у параметрах Mesh Resolution, вказано ширину та довжину об’єкту зі значенням 200 одиниць у меню іконки із зображенням шестерні. На цьому етапі сцена матиме вигляд який зображено на рис. 4.16.

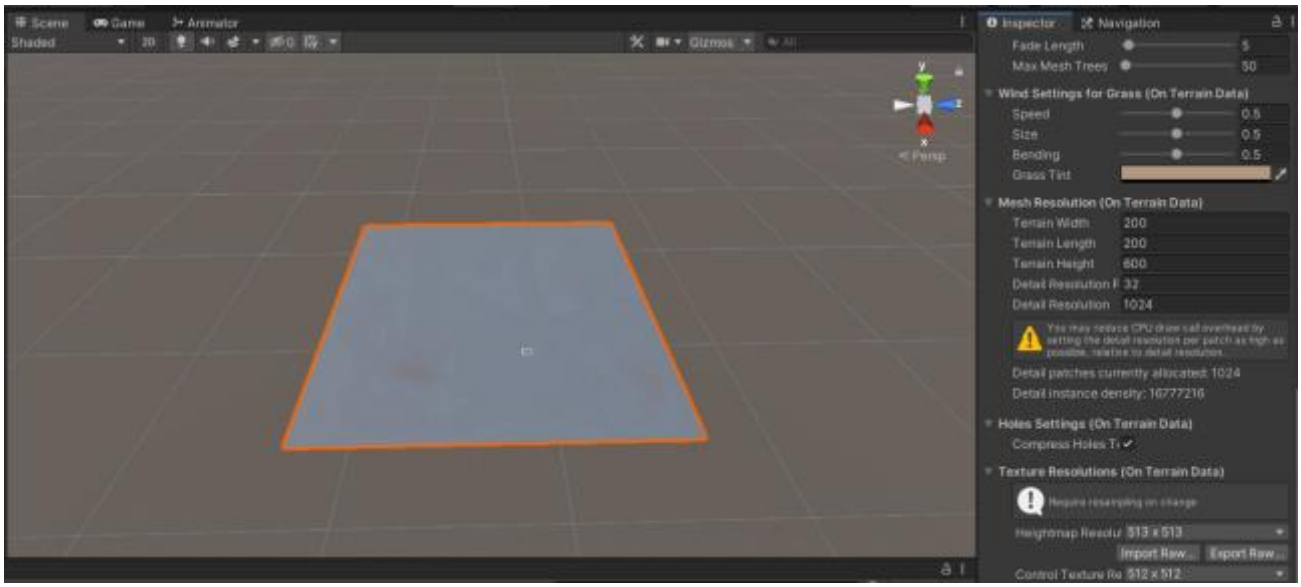

Рисунок 4.16 – Terrain з розмірами 200x200

Далі Terrain розфарбовано в інший колір. Для цього натиснуто на іконку пензлика у компоненті Terrain, після цього у випадаючому списку обрано Paint Texture і додано Terrain Layers з текстурою льоду. Саму текстуру завантажено з інтернету та додано до проекту у каталог з текстурами.

4.2 Налаштування ігрового інтерфейсу

Наступним кроком налаштовано інтерфейс ігрової сцени. Для цього додано до об'єкта MainCamera Canvas, з назвою UI. У ньому створено два Canvas та вказано його батьківський об’єкт у параметрі RenderCamera та значення 5 у параметрі Plane Distance. Назва першого канвасу GameMenu, він буде містити в собі інтерфейс для ігрового процесу, другий PauseMenu вміщує інтерфейс для меню паузи.

Також до MainCamera додано компонент AudioSoure для програвання фонової музики ігрового додатка. Для цього у проект завантажено необхідну мелодію, яку далі додано до параметра AudioClip, а для контролю звуку обрано раніше створений AudioMixer SaundtrackSetting. Налаштування параметрів MainCamera зображено на рис. 4.17.

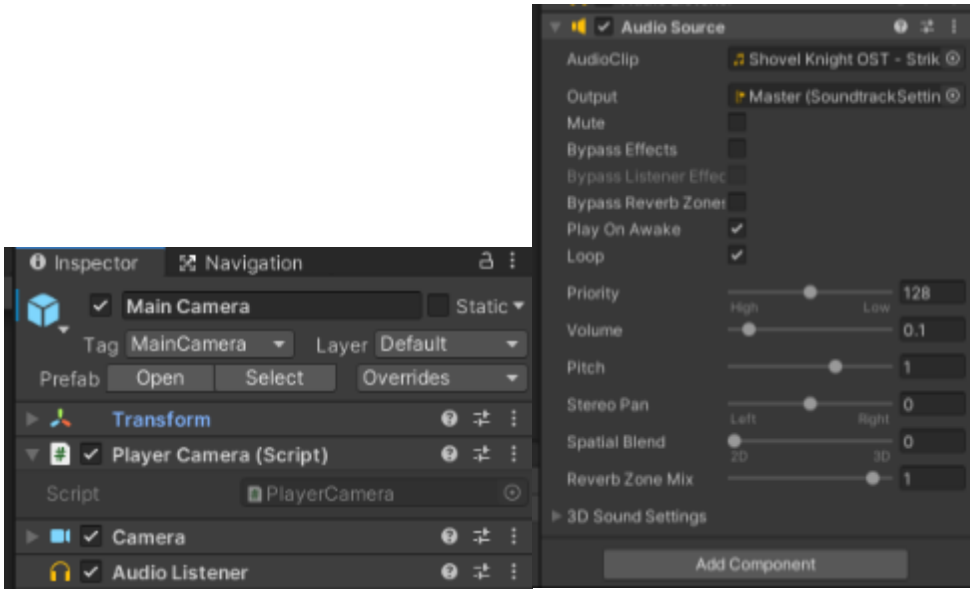

Рисунок 4.17 – Налаштування параметрів MainCamera

Для налаштування інтерфейсу ігрового процесу до GameMenu додано текстове поле для відображення набраних балів у правому верхньому куті екрану, текстове поле для відображення часу від початку проходження рівня та об’єкт Image для іконок секундоміру у лівому верхньому куту. Також додане зображення кнопки паузи у верхньому правому куті біля текстового поля набраних балів. На цьому етапі інтерфейс його ієрархія зображена на рис 4.18.

Рисунок 4.18 – Зображення інтерфейсу ігрового додатка під час ігрового процесу

Для керування гравцем передбачено два джойстики, вони складаються з двох об’єктів Image. Перший для відображення контурів джойстика дочірній йому Image для відображення рухомого кола. Розташовано їх у правому та лівому нижньому куті інтерфейсу.

По центру внизу додано Slider який відображає значення кількості здоров’я гравця. Для дочірнього об'єкта Background змінено Anchor Presets на stretch з обох боків для максимального заповнення простору батьківського елементу. Колір фону слід встановлено на відтінок сірого, а значення Left, Right, Top, Bottom встановлено 0 одиниць.

Далі в об’єкта Fill Area та Handle Slide Area також потрібно змінити Anchor Presets на stretch і вказати значення 0 одиниць для параметрів Left, Right, Top, Bottom. Останнім кроком налаштовано зображення для джойстиків створених в Adobe Illustrator. Після цього остаточний ігровий інтерфейс буде мати зовнішній вигляд який зображено на рис. 4.19.

Рисунок 4.19 – Ігровий інтерфейс додатку

Далі було налаштовано інтерфейс меню паузи. Він містить об’єкт Panel у якому розташовано об’єкти Timer та UiScore з попереднього меню, а також додано дві кнопки з текстом «Продовжити гру» та «Завершити гру». Додатково для Panel у компоненті Image встановлено чорний колір зі значенням прозорості в 100 одиниць. У результаті цього меню паузи матиме вигляд який зображено на рис. 4.20.

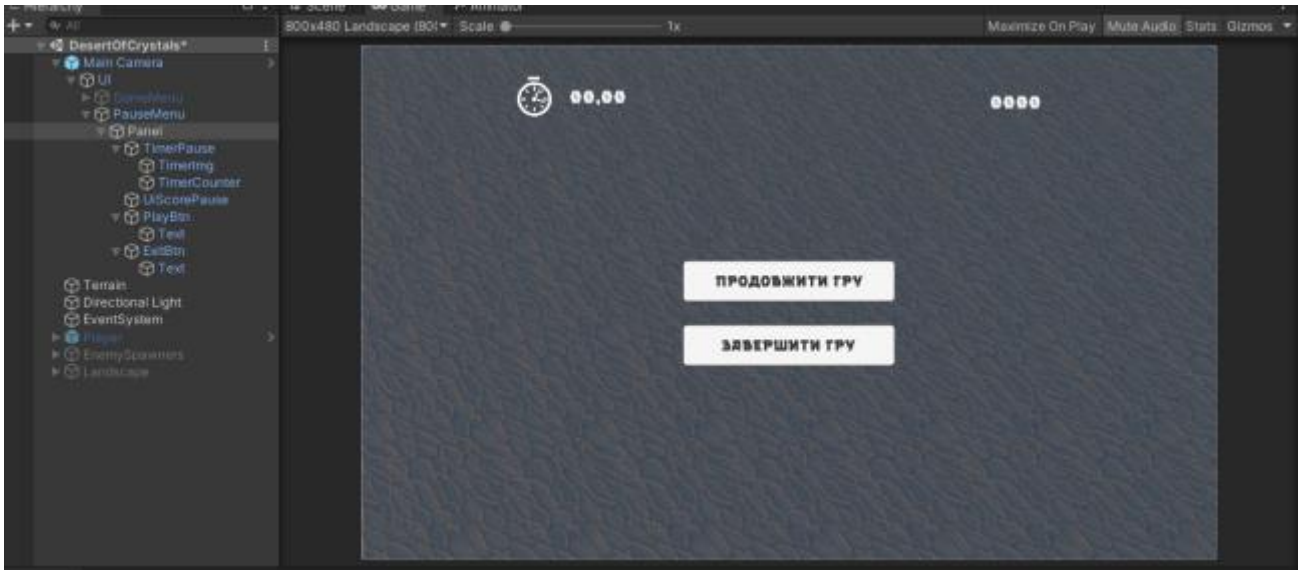

Рисунок 4.20 – Інтерфейс меню паузи

Далі був розроблений власний скрипт PlayerCamera (рис. 4.21) для поведінки камери та її елементів інтерфейсу під час гри, повний програмний код скрипту наведено у додатку Б. Далі скрипт додано до об'єкта MainCamera.

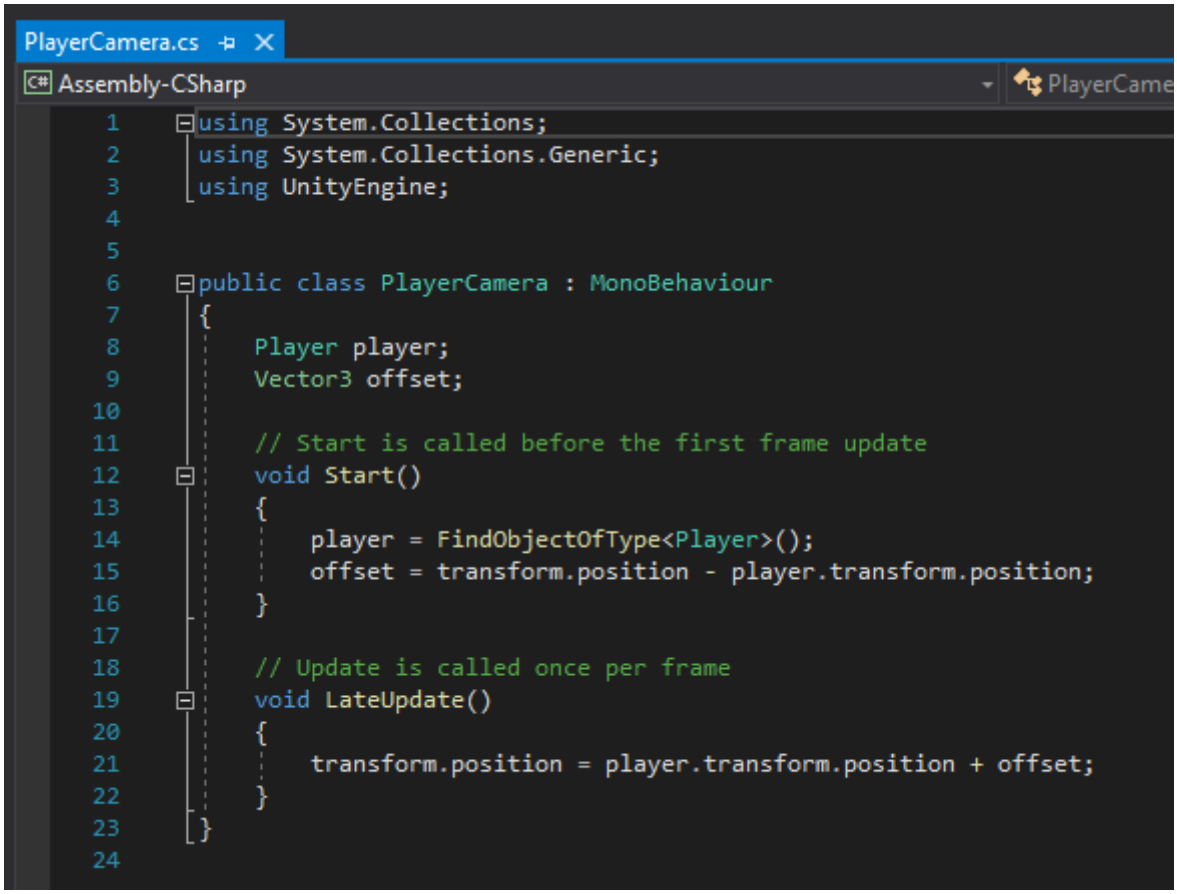

Рисунок 4.21 – Програмний код для поведінки MainCamera

Після цього створено скрипт UiPauseBtn (рис. 4.22) для опрацювання дій з кнопкою паузи в ігровому інтерфейсі, повний програмний код якого наведено у додатку Б. Цей скрипт додано до об'єкта PauseBtn та вказано необхідні параметри які зображені на рис. 4.23. Також до цієї кнопки додатно компонент Event Trigger для опрацювання натискань на кнопку.

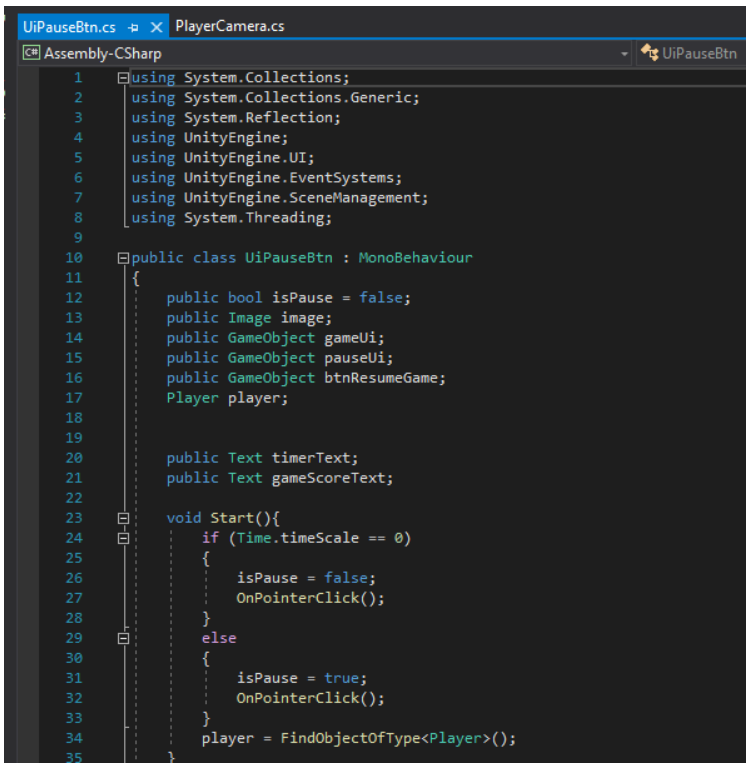

Рисунок 4.22 – Код скрипта для кнопки паузи

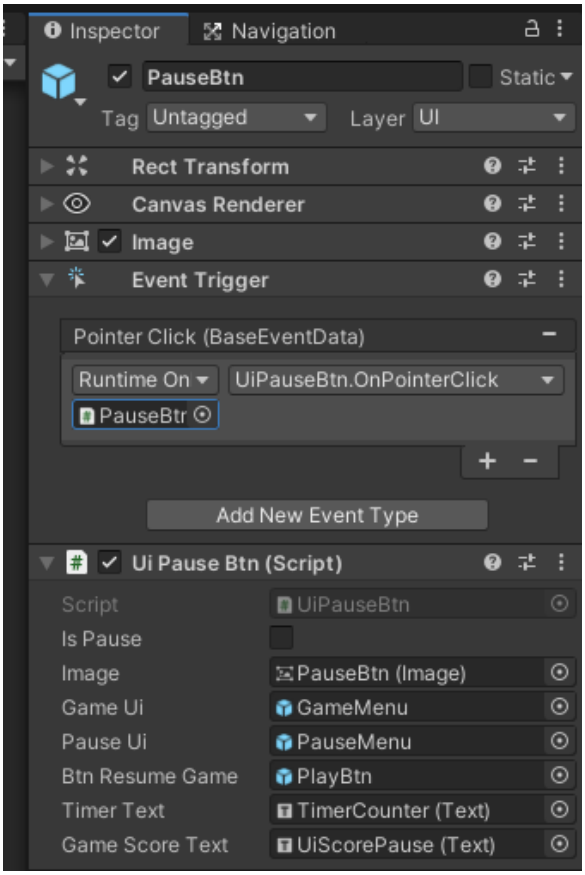

Рисунок 4.23 – Параметри кнопки паузи ігрового інтерфейсу

Після цього створено скрипт UiGameScore (рис. 4.24) для підрахунку та відображення набраних балів за вбивство ворогів, повний програмний код якого наведено у додатку Б. Скрипт додано до текстового поля UiScore та вказано необхідні параметри, які зображено на рис. 4.25.

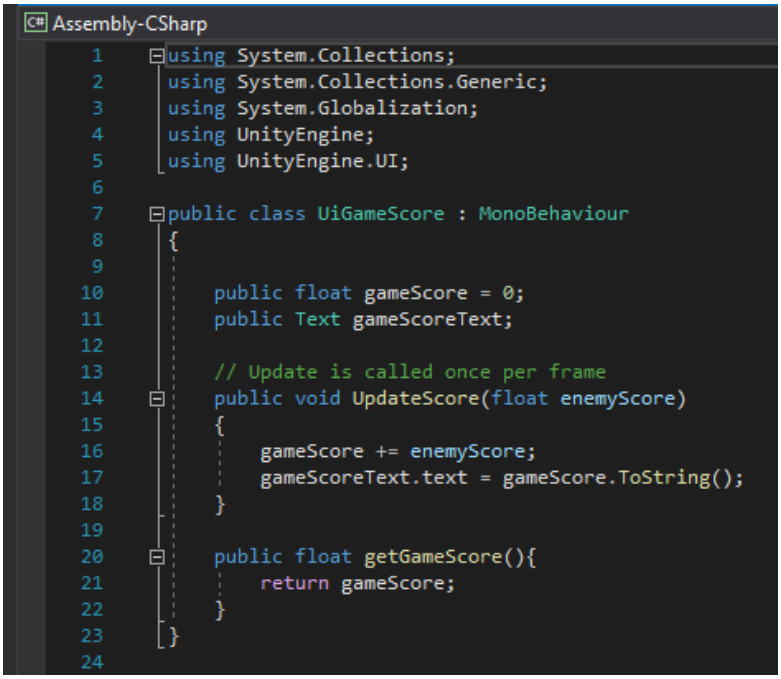

Рисунок 4.24 – Код для підрахунку набраних балів

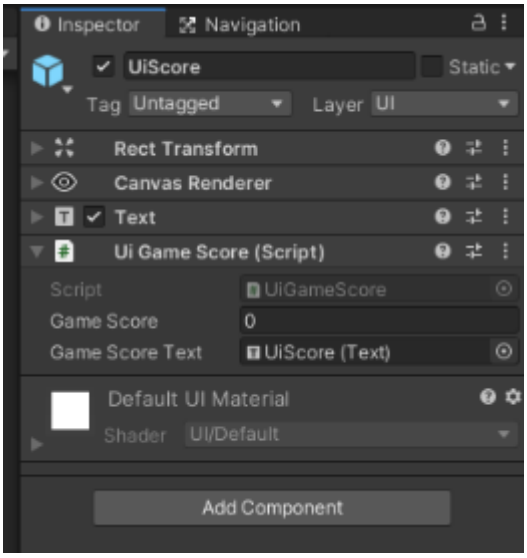

Рисунок 4.25 – Налаштування скрипту для UiScore

Далі було створено скрипт UiTimer (рис. 4.26) для підрахунку часу проведеного від старту ігрового рівня, повний програмний код якого наведено у додатку Б. Скрипт додано до текстового поля TimerCounter та вказано необхідні параметри які відображені на рис. 4.27.

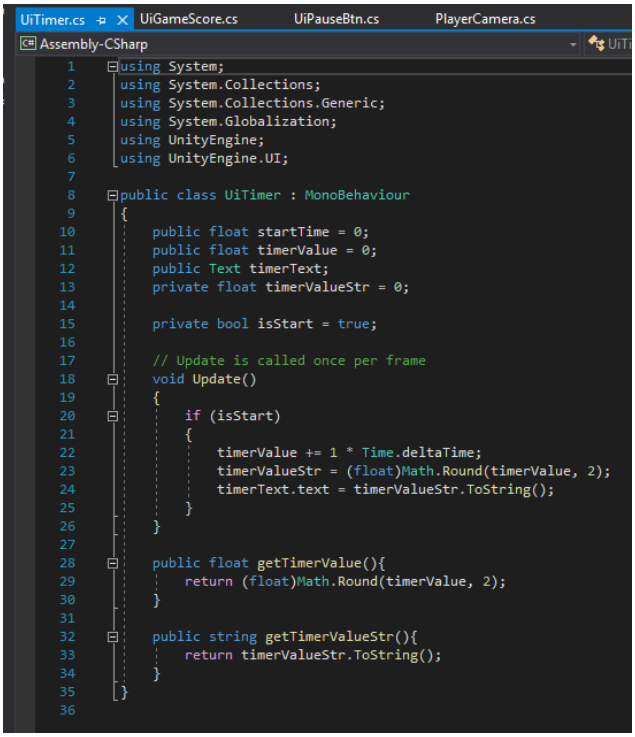

Рисунок 4.26 – Код скрипту для підрахунку часу гри

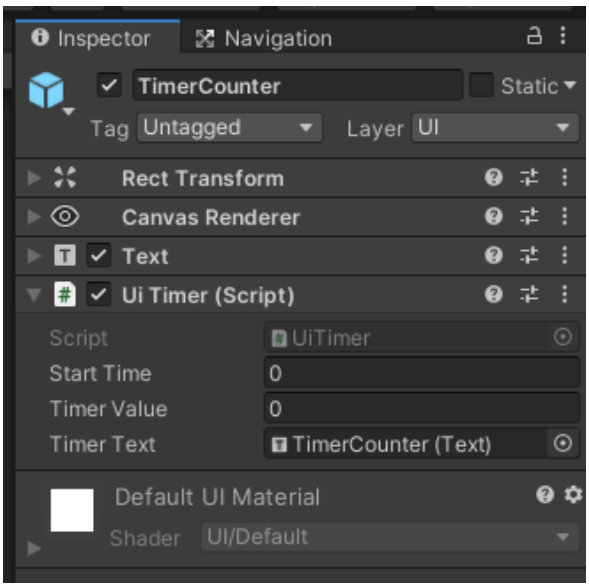

Рисунок 4.27 – Налаштований скрипт у TimeCounter

Після цього створено скрипт HealthBar (рис. 4.28) для візуального відображення кількості здоров'я ігрового персонажа, повний код якого наведено у додатку Б. Далі скрипт додано до повзунка HealthBar та вказано необхідні параметри. Результат налаштування відображено на рис. 4.29.

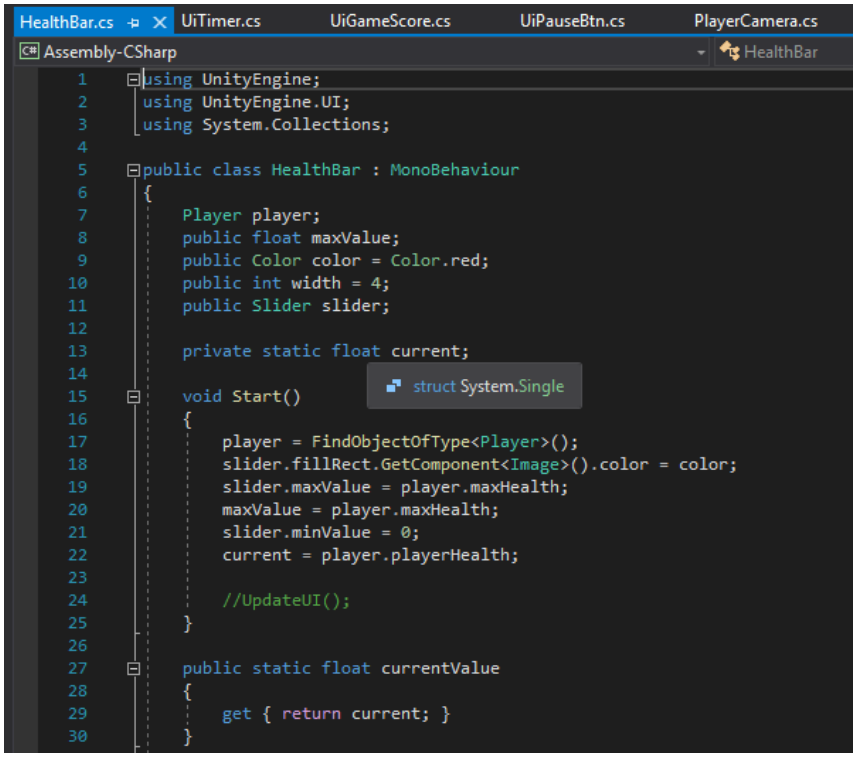

Рисунок 4.28 – Код скрипту поділок здоров'я персонажу

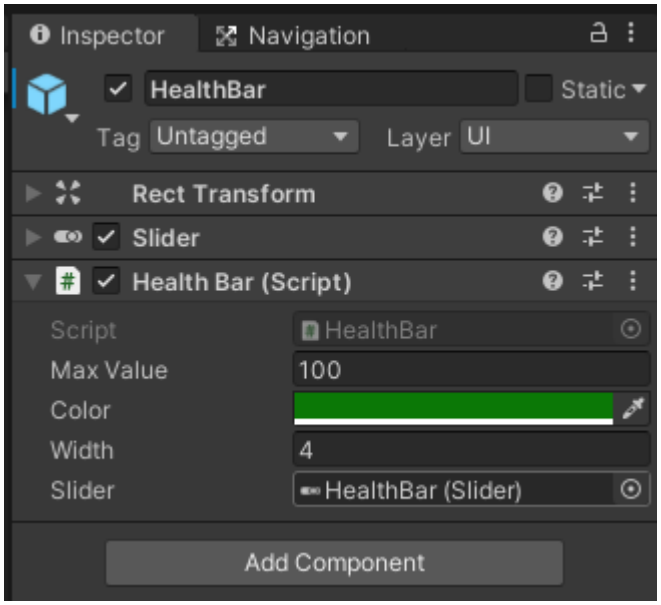

Рисунок 4.29 – Параметри скрипту у слайдера HealthBar

Далі було створено скрипти для опрацювання взаємодії гравця з джойстиком руху персонажу (JoystickMove) та його повороту (JoystickRotate). Спочатку створено скрипт з назвою Joystic_controller (рис. 4.30) для пересування гравця завдяки зміні положення внутрішнього кола, повний код якого наведено у додатку Б. Далі скрипт додано до JoystickMove.

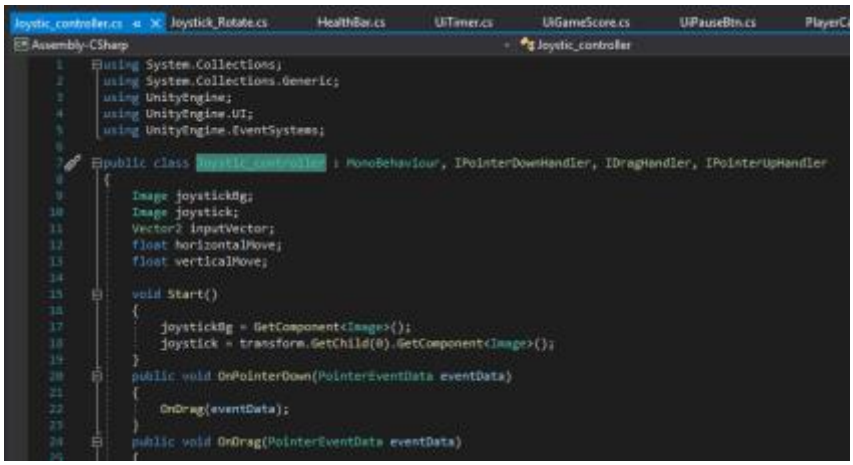

Рисунок 4.30 – Код керування гравцем завдяки джойстику

Після цього створено скрипт JoystickRotate (рис. 4.31) для опрацювання повороту гравця, повний код його наведено у додатку Б. Далі скрипт додано до JoystickRotate.

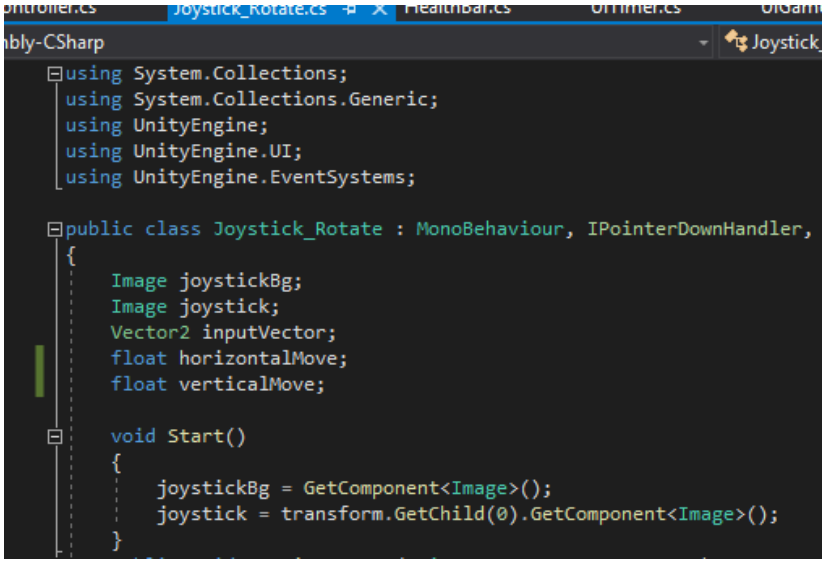

Рисунок 4.31 – Код керування поворотом гравця завдяки джойстику

Далі для налаштування повернення у гру з вікна паузи для кнопки PlayBtn додано у методі OnClick об’єкт PauseBtn у якого викликається виконання методу OnPointerClick. Налаштування кнопки PlayBtn зображено на рисунок 4.32.

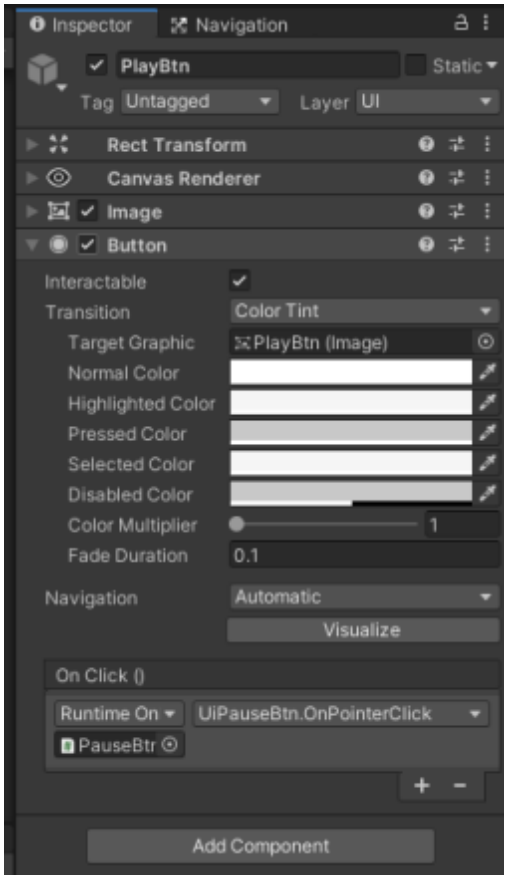

Рисунок 4.32 – Налаштування кнопки повернення у гру з вікна паузи

Далі налаштовано функціонал кнопки повернення у головне меню. Ці дії виконує простий скрипт ExitBtnScript (рис. 4.33) який додано як компонент до кнопки ExitBtn, яка своєю чергою додана до методу OnClick() у власному компоненті Button. Результат цих дій дозволяє викликати метод зі скрипту для зміни сцени, налаштування параметрів кнопки ExitBtn зображено на рис. 4.34.

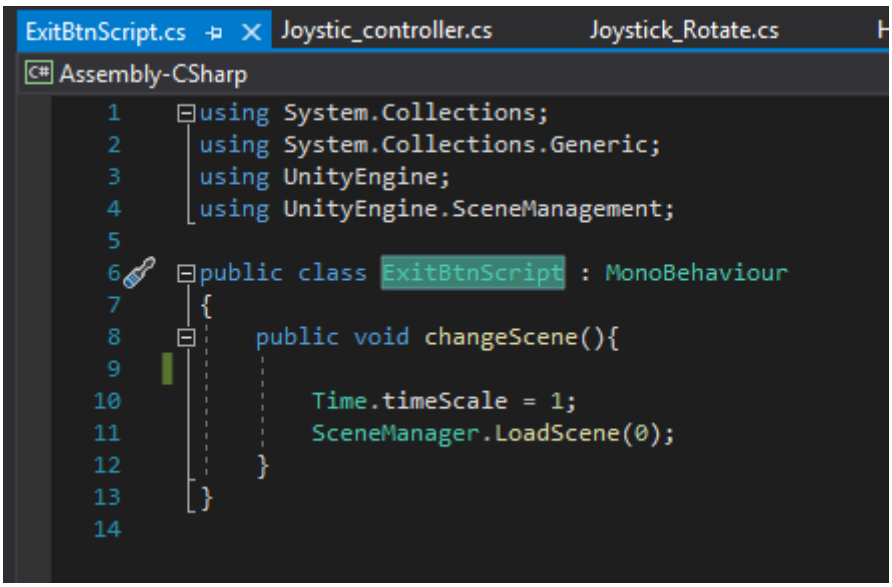

Рисунок 4.33 – Код скрипту переходу в головне меню додатку

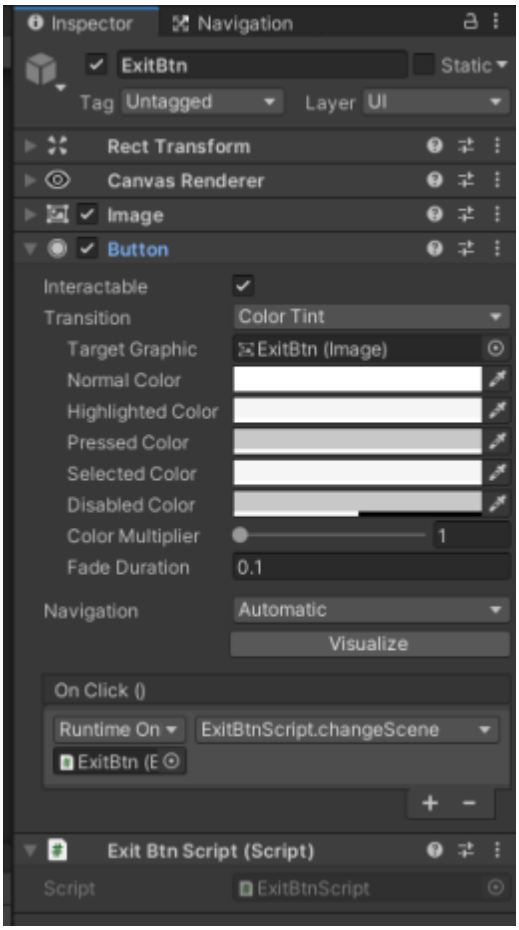

Рисунок 4.34 – Параметри кнопки ExitBtn

На цьому етапі налаштування інтерфейсу ігрового рівня завершено, можна запустити проект, щоб перевірити працездатність додатку. Результат роботи

ігрового інтерфейсу зображено на рис. 4.35. Результат роботи інтерфейсу меню паузи зображено на рис. 4.36. Для прискорення подальшої розробки майбутніх ігрових рівнів камеру MainCamera додано до каталогу Prefab проекту.

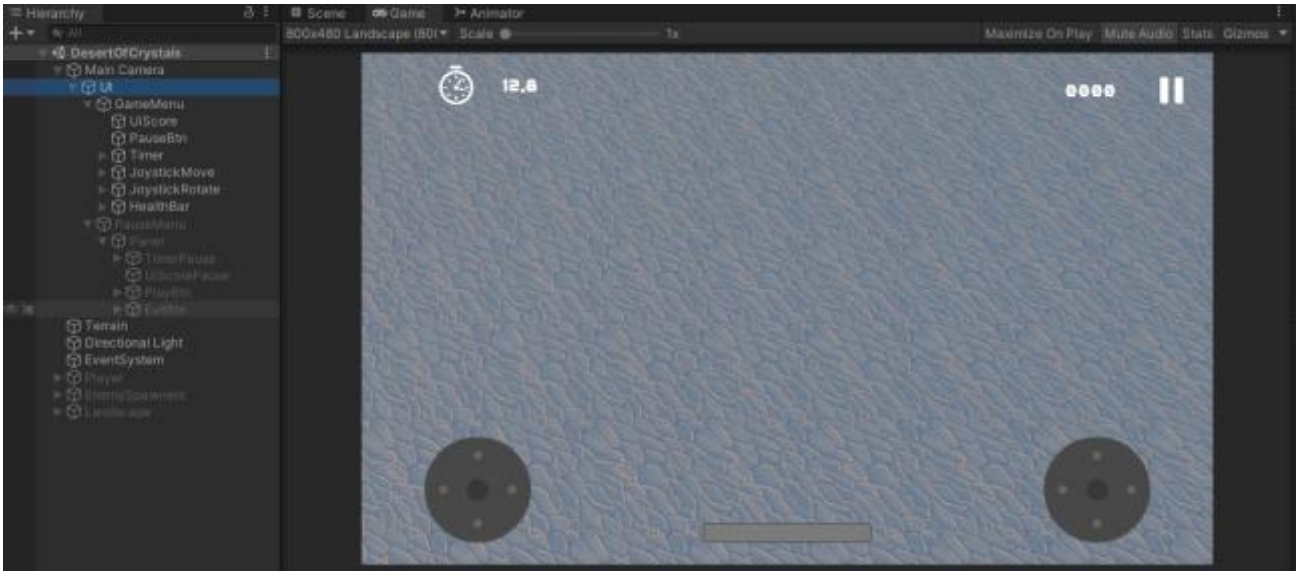

Рисунок 4.35 – Інтерфейс додатку під час проходження рівня

Рисунок 4.36 – Інтерфейс меню паузи додатку

4.3 Створення ігрових персонажів

Наступним важливим кроком буде створення ігрового персонажа для гри. Для цього була використана модель гравця з веб ресурсу AssetStore Unity. Обрану

модель додано до персональних шаблонів та імпортовано через вікно ProjectManager до каталогу проекту з вибором необідних текстур та матеріалів. Вікно імпорту з параметрами зображено на рис. 4.37.

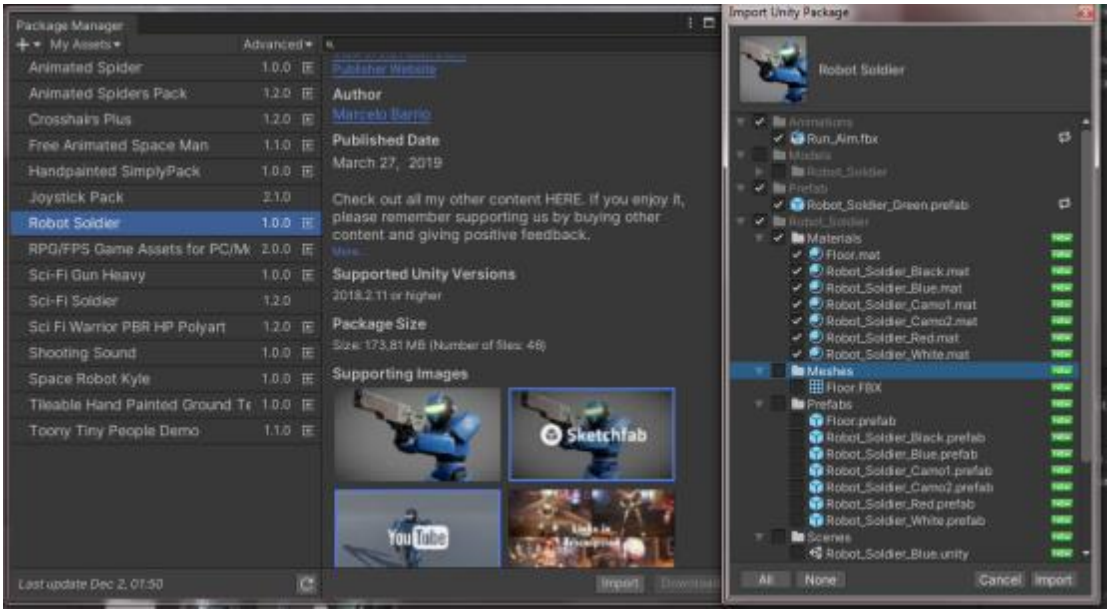

Рисунок 4. 37 – Вікно імпорту моделі у проект

Після імпорту персонажу можна додати його на ігровий рівень, додати до нього компоненти Rigidbody, CapsuleCollider та вказати тег Player. Додана модель персонажу має бути дочірнім об’єктом групи Player. Параметри налаштування моделі персонажу зображено на рисунку 4.38.

Рисунок 4.38 – Модель персонажу та її параметри

Після цього до об’єкта Player додано Particle System з назвою hitPlayer. Цей генератор часток буде використано для відображення пошкодження персонажу від атак ворогів. Розміщено цей об’єкт у середині персонажу з налаштуваннями у компоненті Transform з параметрами які зображено на рис. 4.39.

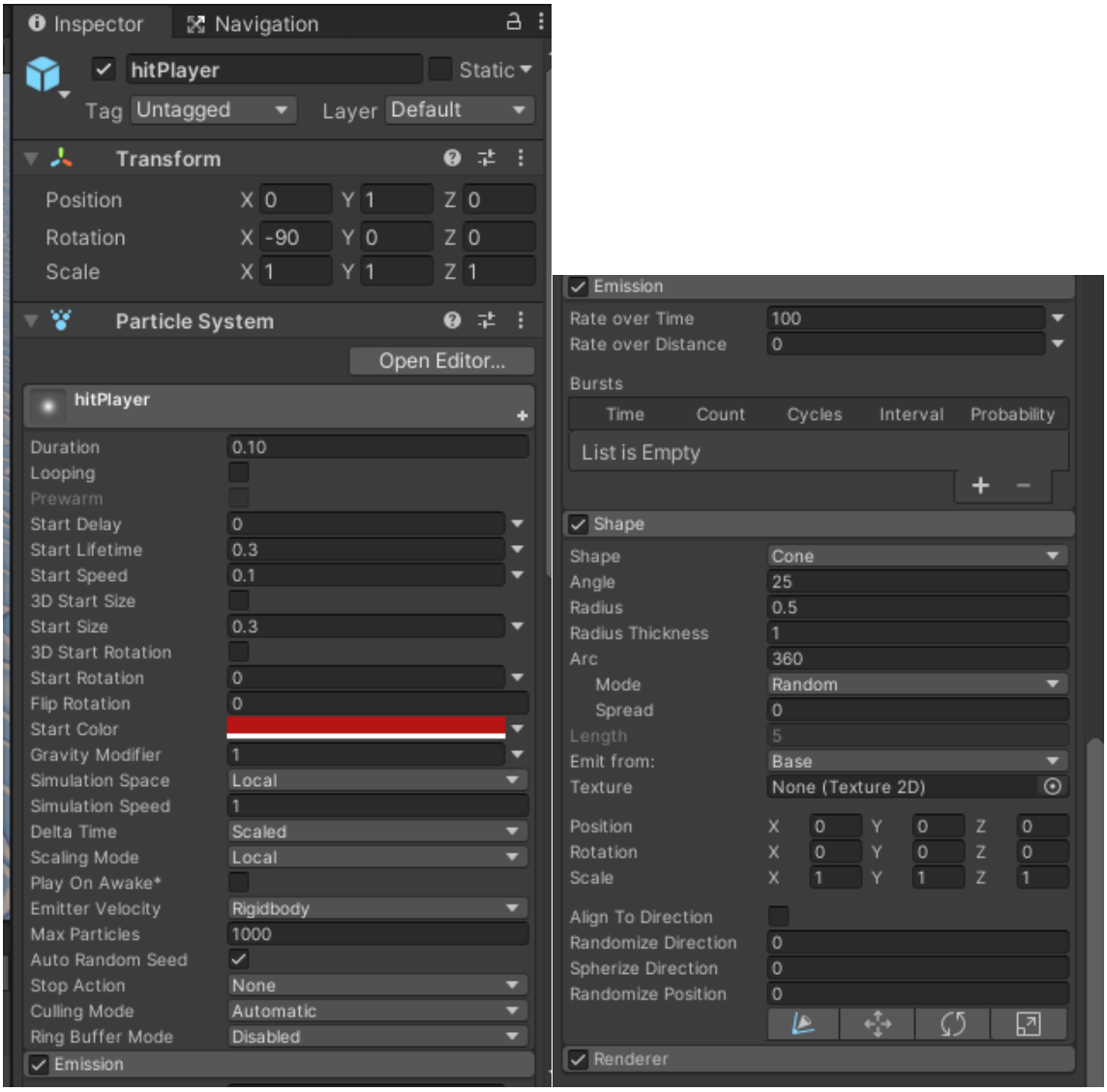

Рисунок 4.39 – Параметри генератору часток

Після цього до об’єкта Player було додано UI – Image для відображення прицілу у напрямку якого персонаж робить постріли, його параметри зображено на рисунку 4.40. Спрайт прицілу створено в Adobe Illustrator, результат роботи зображено на рис. 4.40.

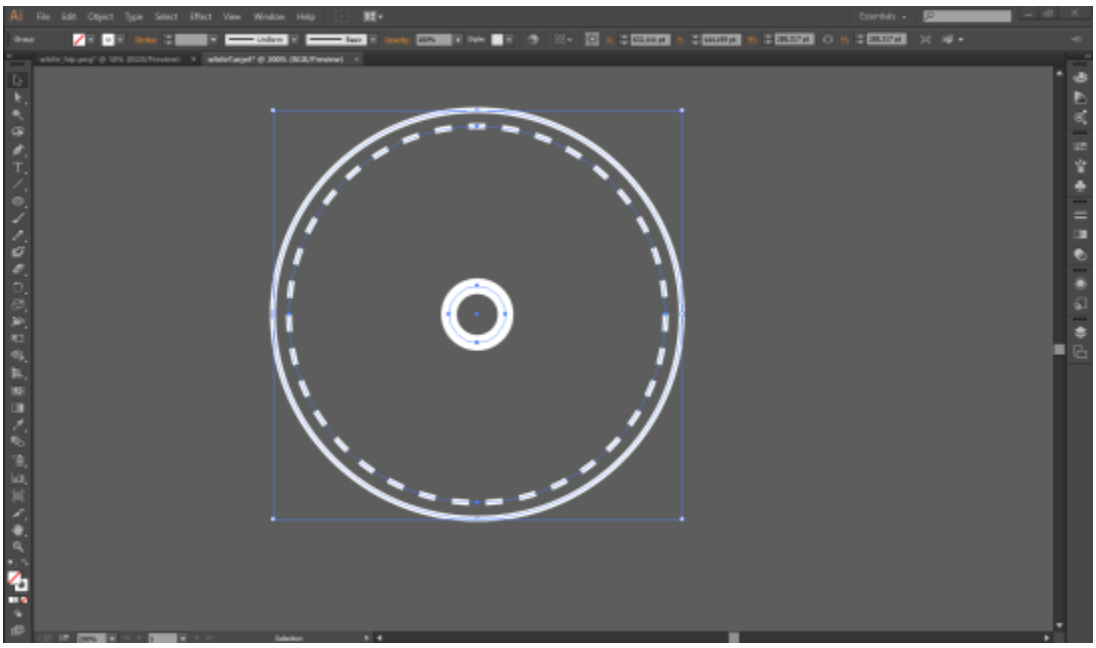

Рисунок 4.40 – Створення спрайту ігрового прицілу

Останнім дочірнім елементом групи Player буде зображення пострілу, для цього було створено порожній GameObject з назвою Shot, до якого додали компонент Line Renderer. За допомогою поля Width у редакторі змінимо його ширину на 0.04.

Далі для цього об’єкту створено матеріал з назвою ShotCollor. Тип шейдеру для матеріалу слід обрати Unlit/Color. Цей стандартний шейдер дозволить не враховувати освітлення, тому нашу постріл буде видно навіть у темноті, а колір слід обрати жовтий. Налаштування параметрів матеріалу зображено на рис. 4.41, а об’єкту Shot на рис. 4.42.

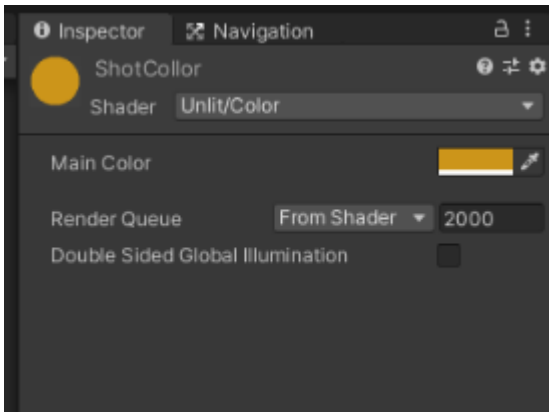

Рисунок 4.41 – Параметри матеріалу ShotColor

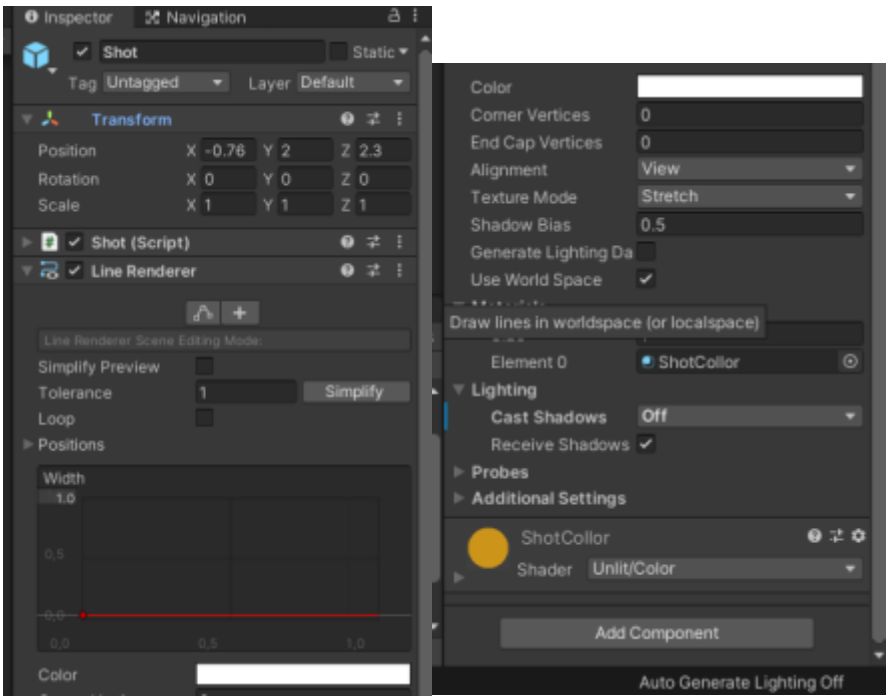

Рисунок 4.42 – Параметри об’єкту Shot

Після цього створено скрипт Shot (рис. 4.43) для відображення лінії пострілу у потрібний момент у заданому напрямку. Повний програмний код скрипту наведено у додатку Б.

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

public class Shot : MonoBehaviour
{
    LineRenderer lineRenderer;
    bool visible;

    // Start is called before the first frame update
    void Start()
    {
        lineRenderer = GetComponent<LineRenderer>();
    }

    // Update is called once per frame
    void FixedUpdate()
    {
        if (visible)
            visible = false;
        else
            gameObject.SetActive(false);
    }

    public void Show(Vector3 from, Vector3 to)
    {
        lineRenderer.SetPositions(new Vector3[]{ from, to });
        visible = true;
        gameObject.SetActive(true);
    }
}
```

Рисунок 4.43 – Код відображення лінії пострілу.

Після цього можна перейти до налаштування самого об’єкту Player, для його додамо компоненти NavMeshAgent та AudioSoure, параметри яких зображено на рис. 4.44.

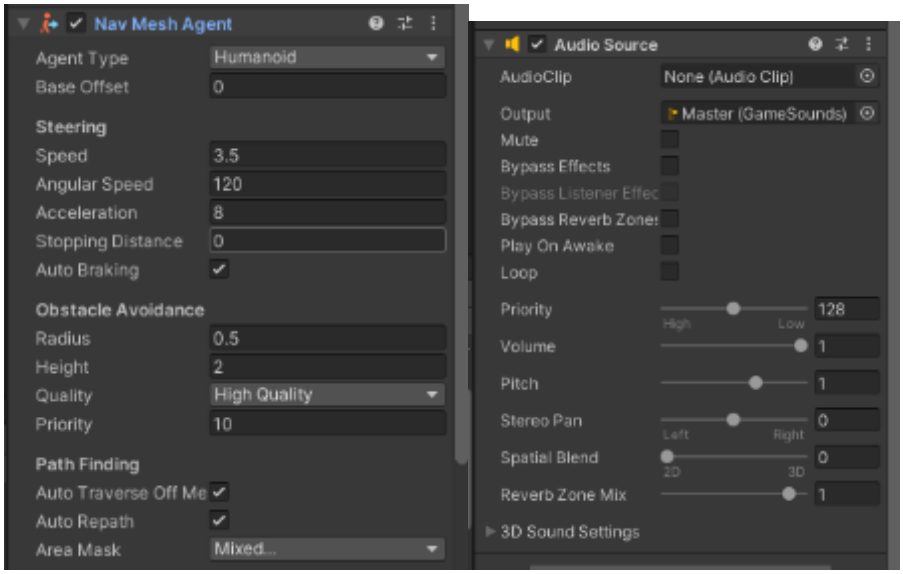

Рисунок 4.44 – Параметри компонентів для Player

Після цього було створено скрипт MovementAnimator (рис. 4.45) для контролю анімацій гравця та Player (рис. 4.46) для опису можливостей персонажу, визначення параметрів швидкості, кількості здоров’я та інших. Повний програмний код скрипту наведено у додатку Б, а його параметри зображено на рис. 4.45.

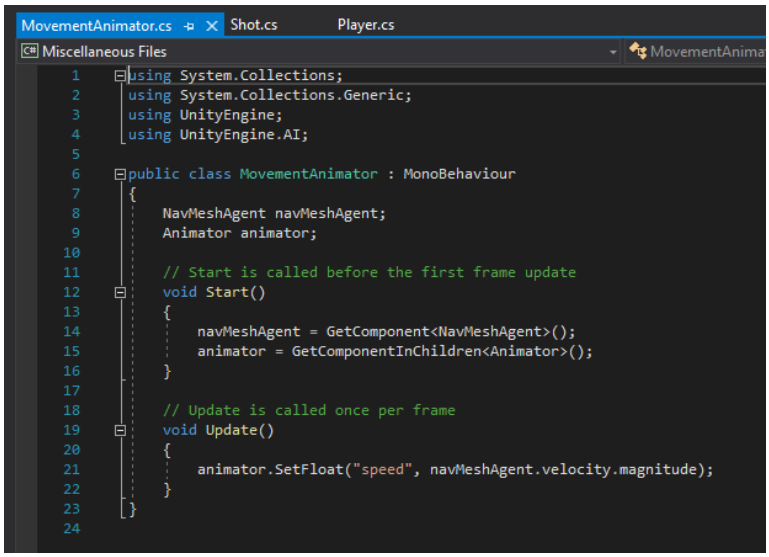

Рисунок 4.44 – Код керування анімацією гравця

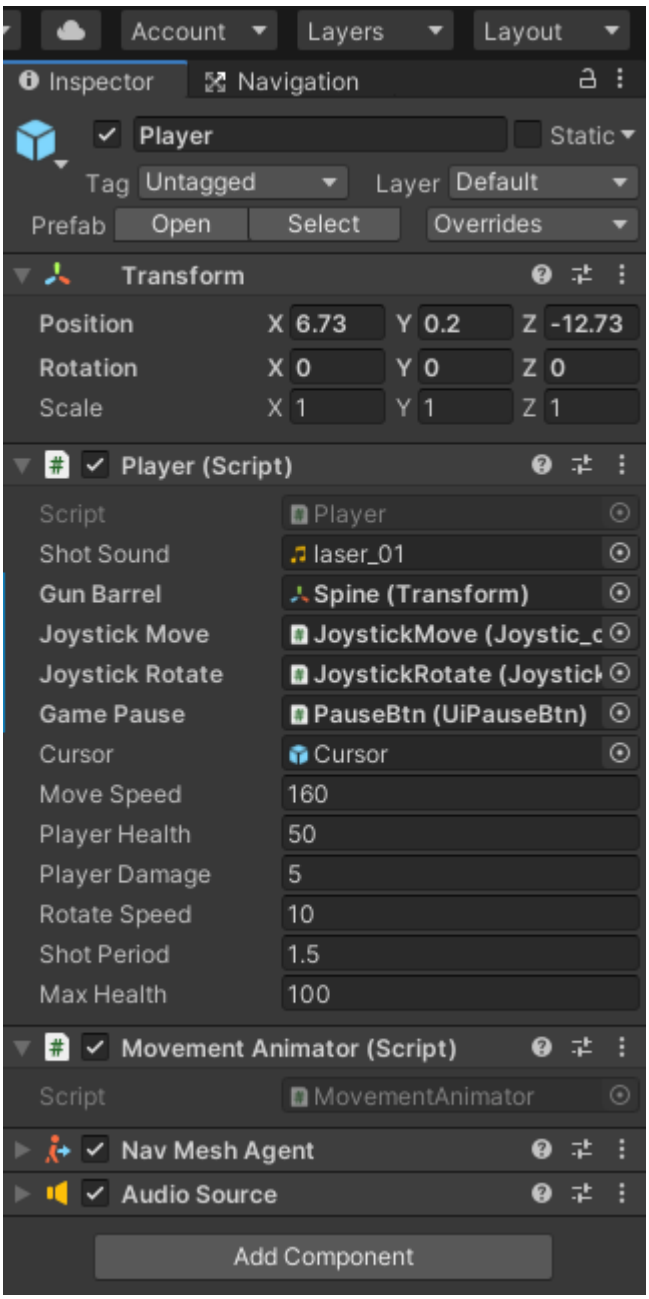

Рисунок 4.45 – Параметри Player

Останнім кроком налаштування персонажу було створення контролеру анімацій для персонажа з назвою Robot. Для цього створено каталог Animation у якому виконано команди Create – Animation Controller. Після створення його в ньому у вікні Animator було додано параметр типу float з назвою speed який приймає значення швидкості гравця зі скрипту Movement Animator.

Далі було налаштовано анімації та переходи між ними. У персонажа задіяно дві стани анімації з назвами idle та walk, але в кожному зі станів виконується

анімація Run_Aim. Різниця в тому, що у стані idle швидкість анімації встановлена зі значенням 0.0001 одиниць, таким чином гравець ніби завмирає.

Переходи між станами залежать від значення speed. Якщо значення змінної більше за 0, то відбувається перехід до анімації walk, а навпаки анімація змінюється за умови коли speed менше за 0.0001. Зображення налаштованої анімації наведено на рис. 4.46.

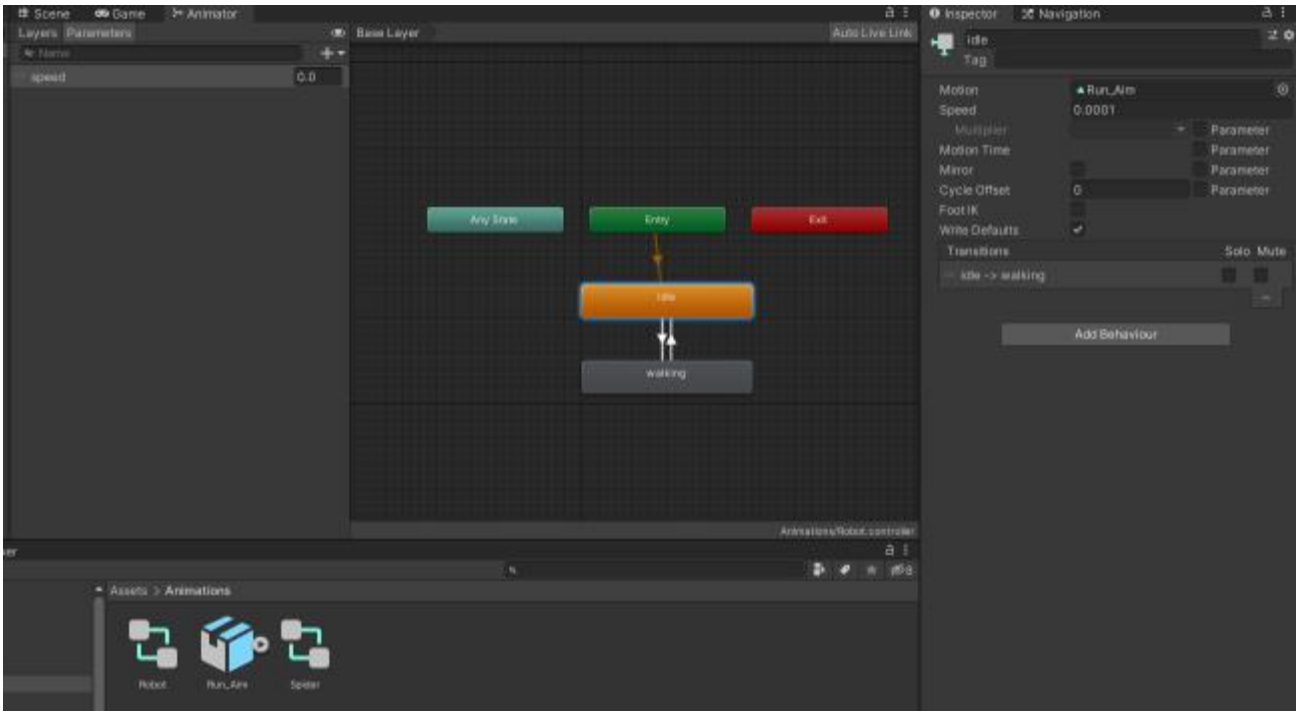

Рисунок 4.46 – Вікно налаштувань аніматора персонаж

На цьому створення персонажу можна вважати закінченим, об’єкт Player було додано в каталог Prefabs, щоб у майбутньому при створенні нових сцен з ним було простіше працювати.

Для того, щоб гравець міг рухатися по ігровій сцені, було прораховано територію на якій він може це робити. Спочатку було відкрито вікно навігації (Window – AI – Navigation) завдяки якому було прораховано зону можливого руху. Для її розрахунку у закладці Bake потрібно вказати допустимі розміри агентів (персонажів та ворогів), щоб Unity прорахував доступну для руху територію. Після натискання кнопки Bake відбувається розрахунок території, результат цієї операції зображено на рис. 4.47.

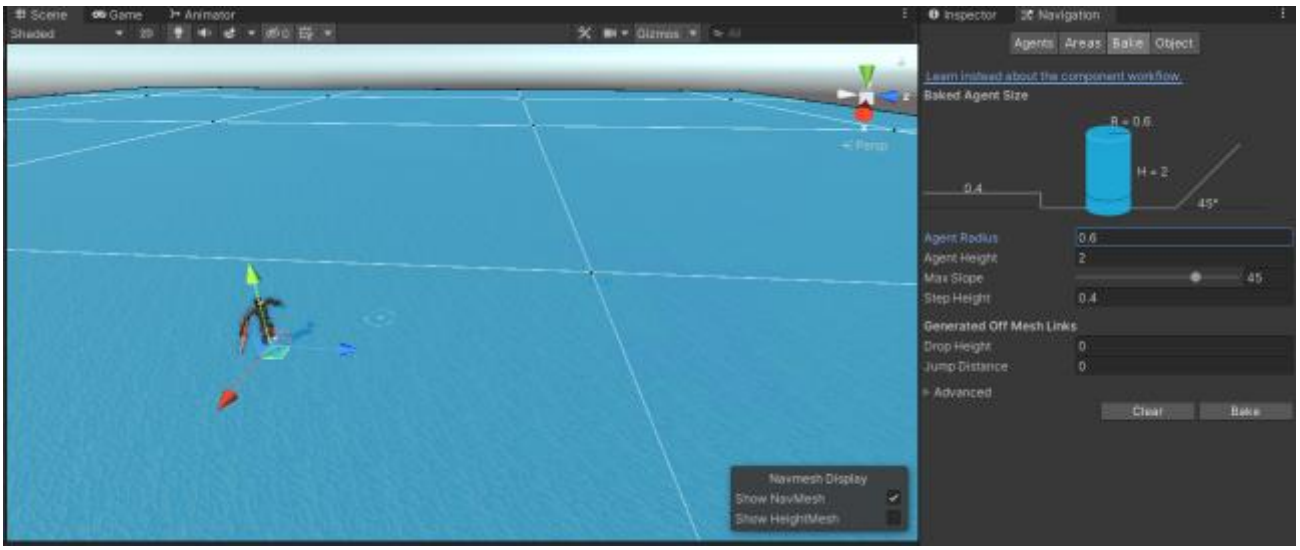

Рисунок 4.47 – Прорахування доступної для пересування зони

Наступним кроком буде створення ворога. Для цього до проекту було імпортовано модель паука з анімаціями, текстурами та матеріалами із веб-ресурсів. Після цього ця модель була додана на сцену під назвою spider з батьківським об’єктом SpiderEnemy.

До SpiderEnemy було змінено тег на Enemy та додано компоненти AudioSource, Nav Mesh Agent, Capsule Collider та скрипт Movement Animator для контролю станів анімації. Додатково до проекту було імпортовано звуки для паука, а саме його атаки, смерті та отримання пошкоджень. Зображення параметрів об’єкту SpiderEnemy наведено на рис. 4.48.

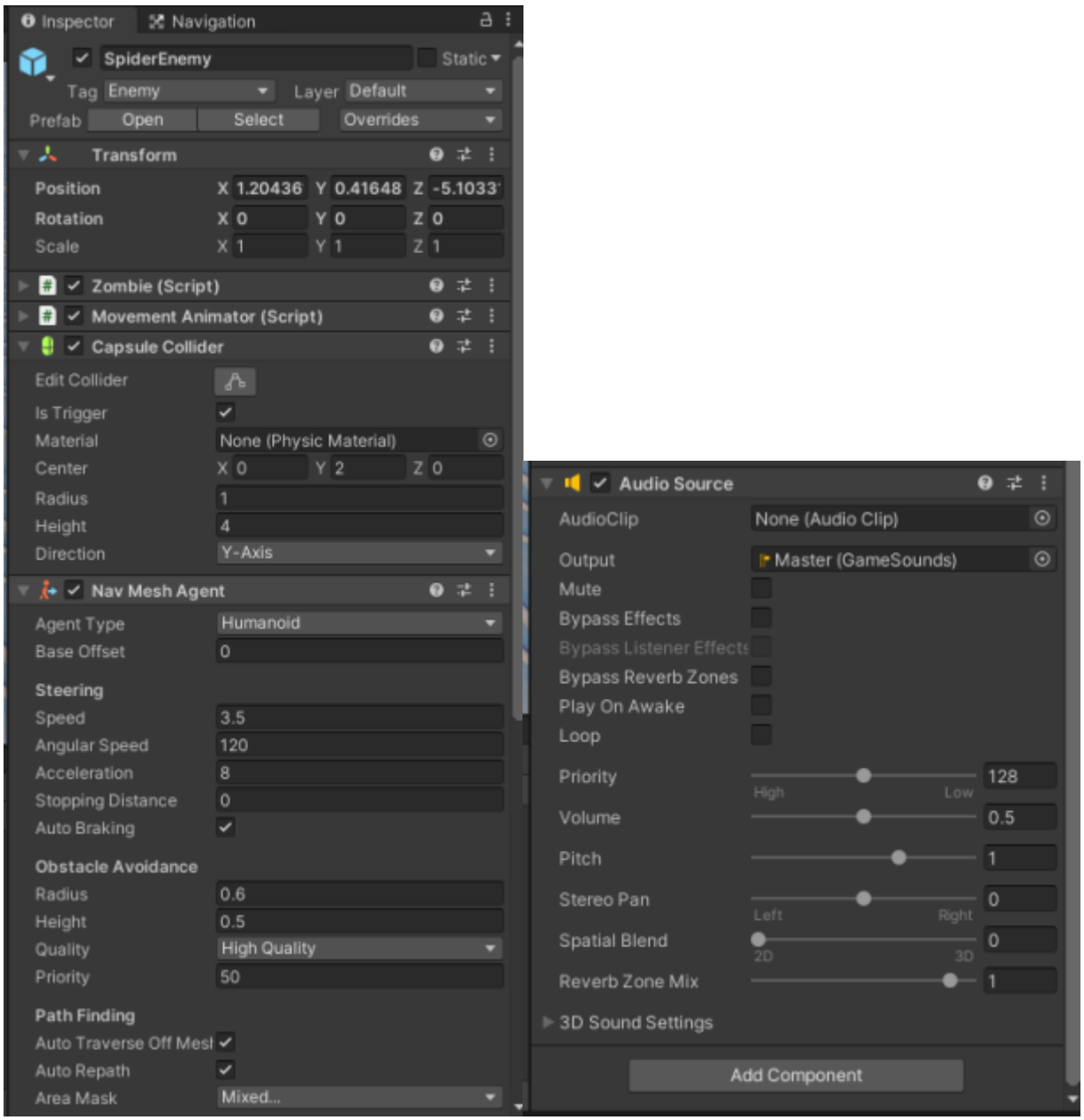

Рисунок 4.48 – Параметри об’єкту SpiderEnemy

Після цього для SpiderEnemy було створено основний скрипт Zombie (рис. 4.49) який відповідає за параметри ворога, механіки пошуку шляху до гравця, атаку та решту його дій. Параметри скрипту зображено на рис. 4.50.

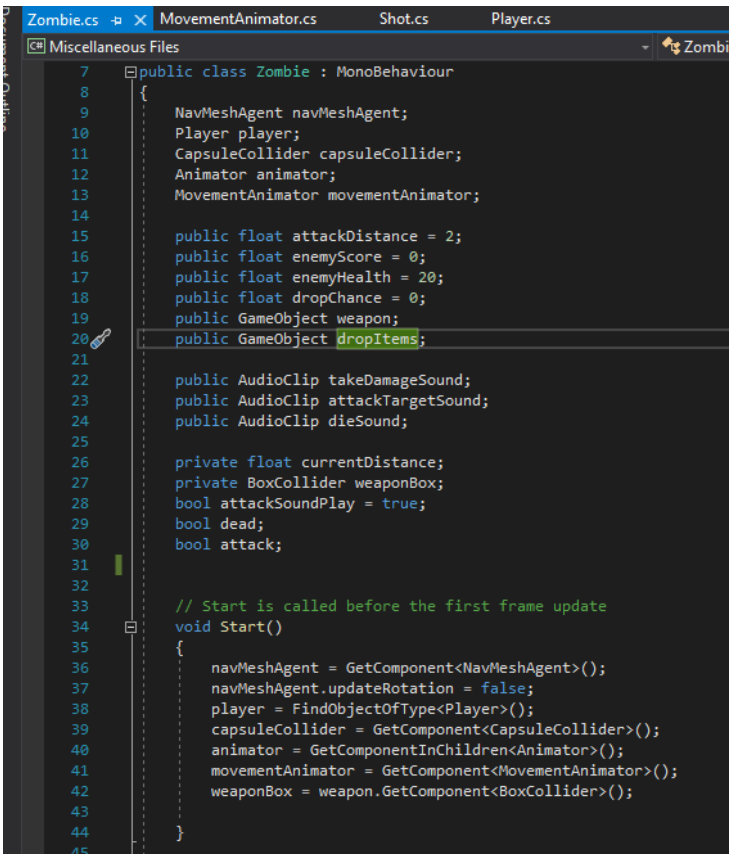

Рисунок 4.49 – Основний скрипт поведінки ворога

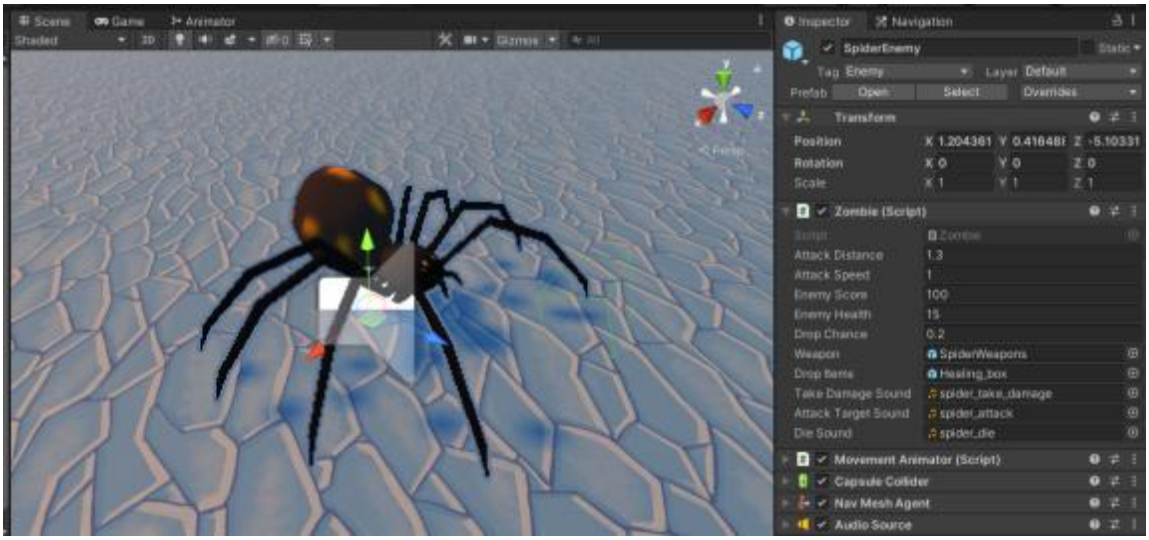

Рисунок 4.50 – Параметри ворога

Після цього додатково був створений допоміжний об’єкт SpiderWeapons який імітує зброю ворога. Додати його потрібно до дочірніх елементів моделі spider, а саме як зображено на рис. 4.51. Також до SpiderWeapons додано BoxCollider та

скрипт HitPlayer (рис. 4.52). Загалом параметри SpiderWeapons мають вигляд як зображено на рис. 4.53.

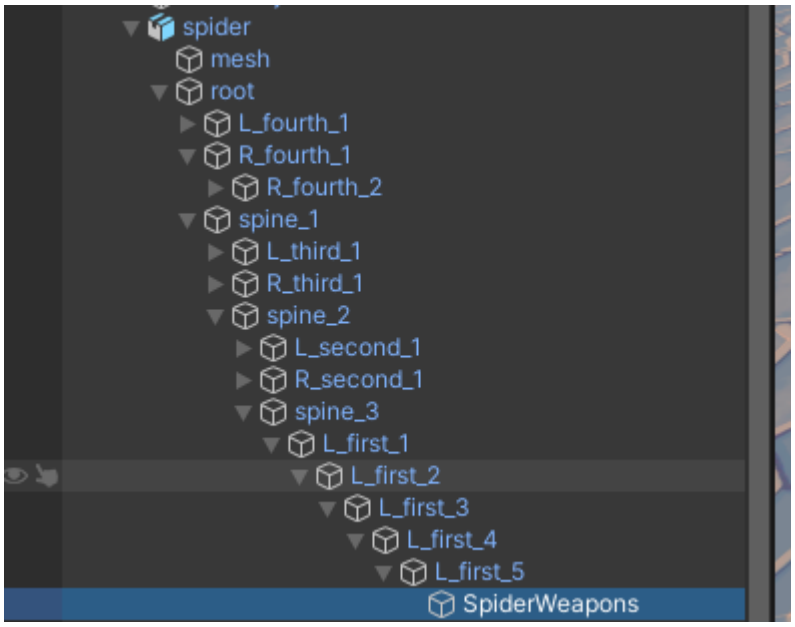

Рисунок 4.51 – Розташування об’єкту на моделі

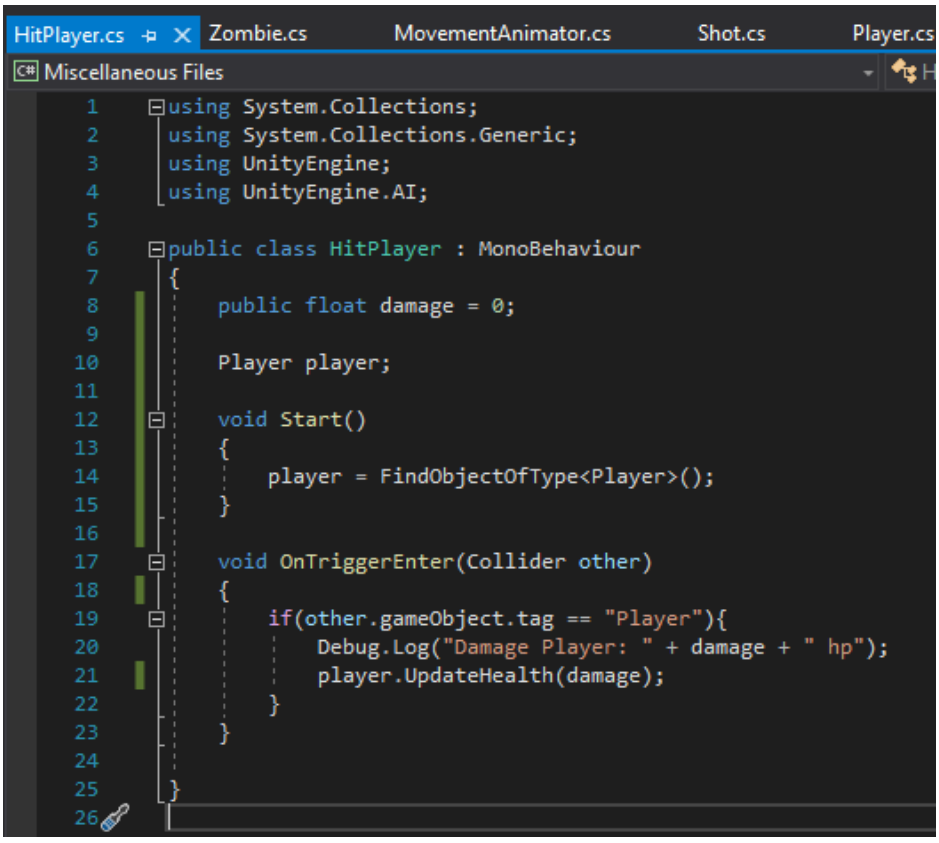

Рисунок 4.52 – Код нанесення пошкоджень гравцю

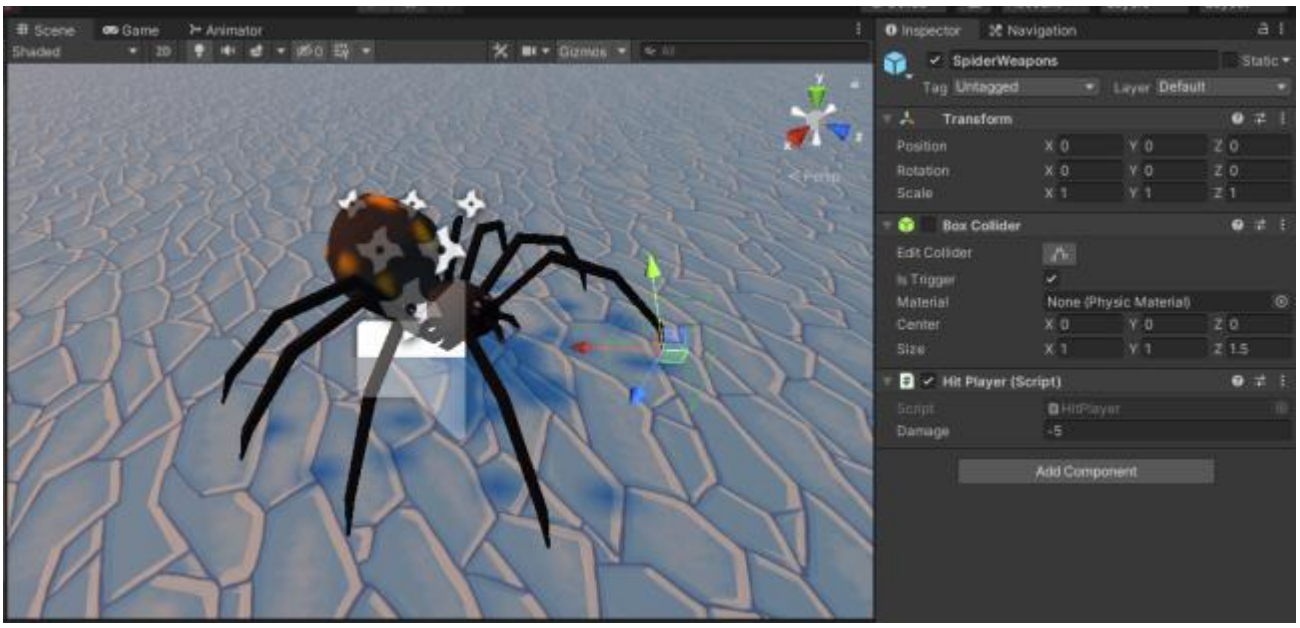

Рисунок 4.53 – Параметри об’єкту що імітує зброю ворога

Наступним кроком були задані налаштування анімації для ворога. Для цього створено аніматор Spider у якому додали змінну типу float з назвою speed, та додали ще дві змінні – bool з назвою died та bool з назвою attack. У вікні станів анімації було додано анімації з моделі spider з назвами die, idle, walk, attack. Далі їх було розмішено як зображено на рис. 4.54, а також для анімації walk змінили значення speed на 3 одиниці, щоб прискорити анімацію до більш натуральної.

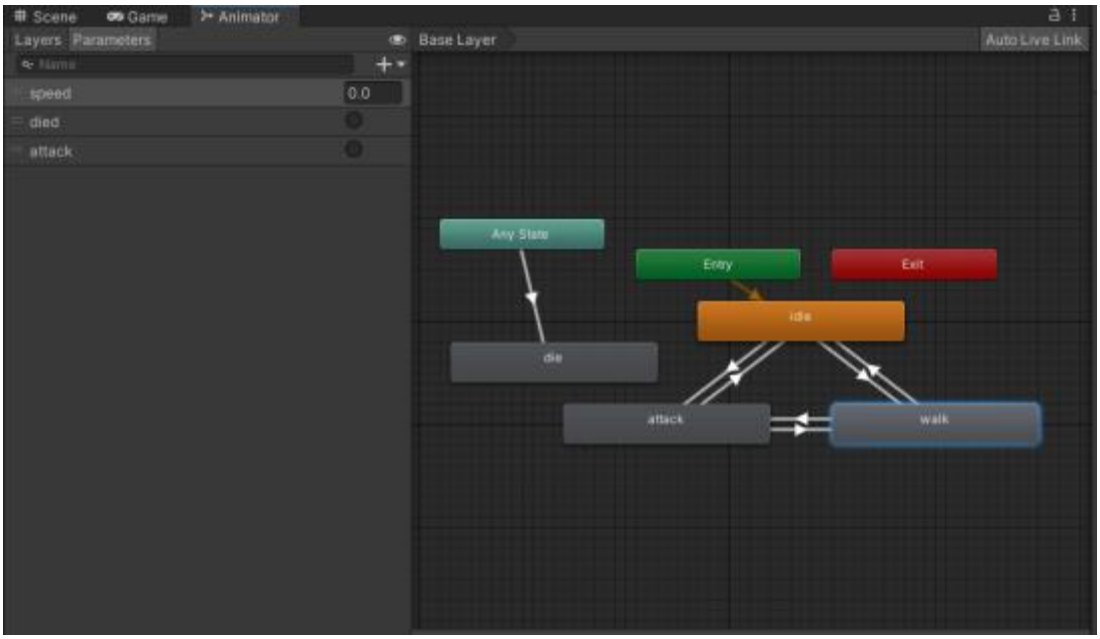

Рисунок 4.54 – Вікно аніматора Spider

Після додавання анімацій було налаштовано переходи між ними. Від стану Any State до анімації die перехід відбувається після зміни стану змінної died. Для переходу від idle до attack за умови зміни стану змінної attack, навпаки перехід відбувається якщо значення змінної speed менше за 0.0001. Перехід від idle до walk спрацьовує тоді, коли значення змінної speed більше за 0, а навпаки коли значення змінної менше за 0.0001. Перехід від walk до attack після зміни стану змінної attack, а навпаки якщо speed більша за 0.

Останнім кроком було продубльовано об’єкт hitPlayer від гравця до об'єкта Spider Enemy та змінено його назву на hitEnemyParticle. У компоненті Particle System було змінено значення Start Color, а також параметри компоненту Transform. Оновлений генератор частинок для ворога зображено на рис. 4.55. Після цього створення ворога завершено, далі було створено з нього Prefab з метою подальшого використання у грі.

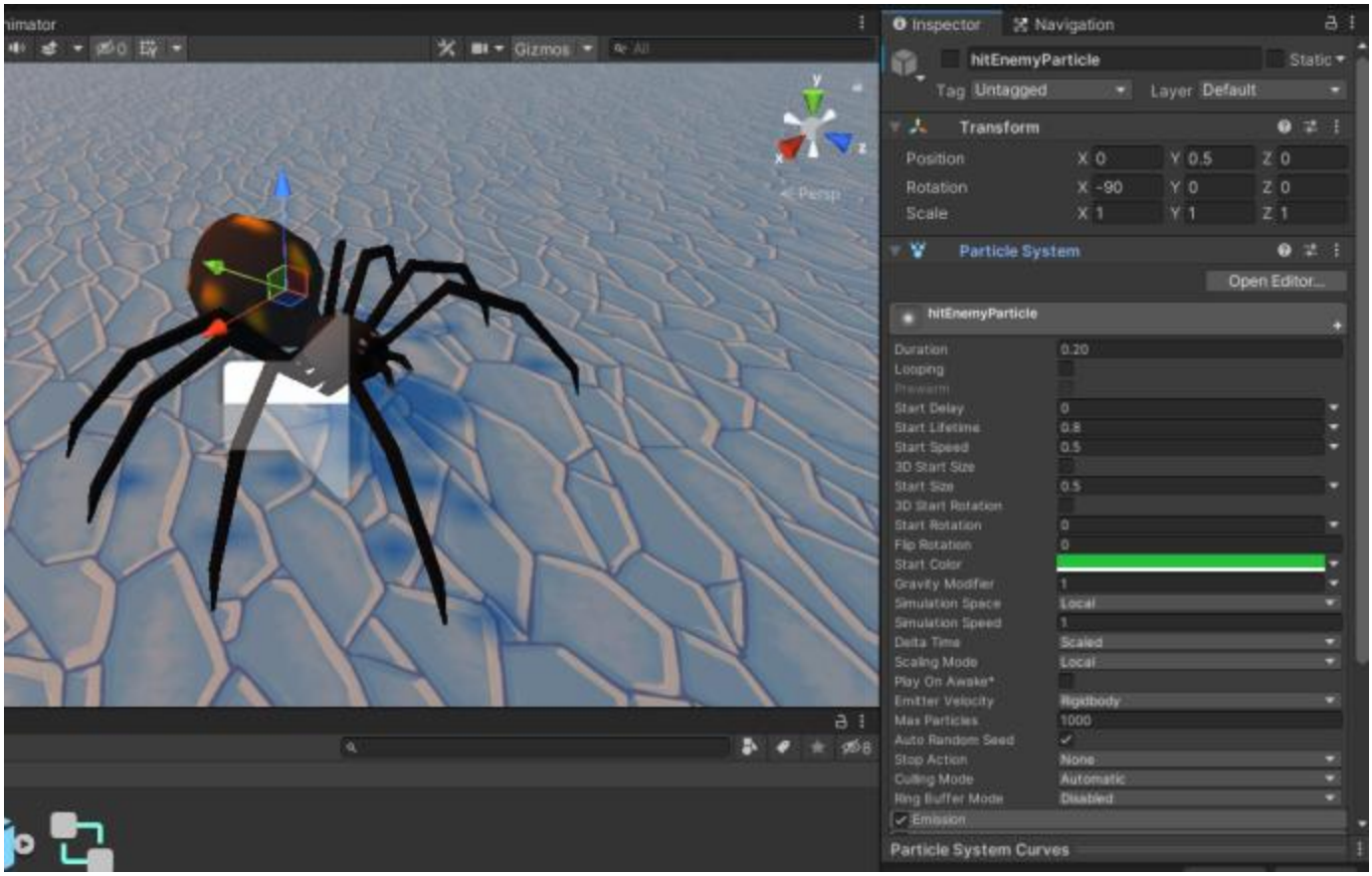

Рисунок 4.55 – Параметри генератора частинок для ворога

4.4 Створення тривимірних моделей ігрового оточення

Наступним кроком було створення об’єктів для наповнення ігрової сцени деталями. Для початку у Blender було створено модель каменю, для цього спочатку потрібно активувати додаткові примітиви Edit – Preferences – Add-ons – відмітити групу Add Mesh: Extra Objects. Далі на сцені створюємо Mesh – Round Cube. Зображення фігури з параметрами на цьому етапі відображено на рис. 4.56.

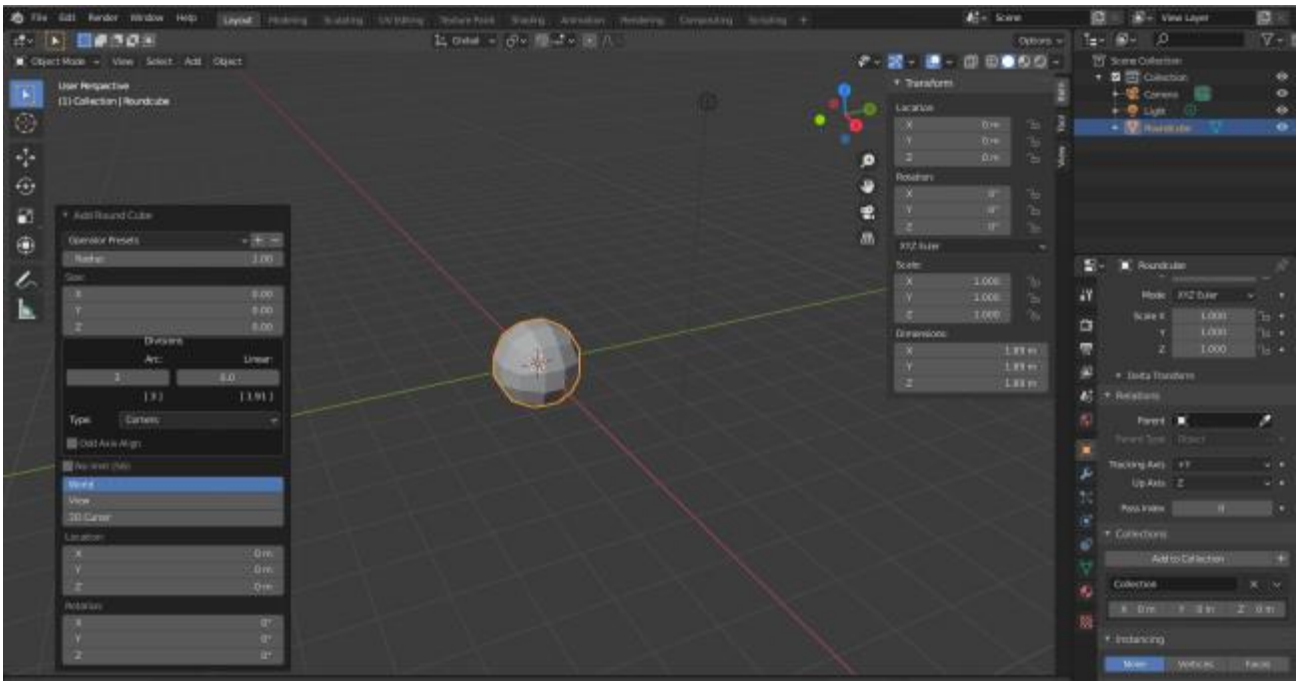

Рисунок 4.56 – Roundcube з початковими налаштуваннями

Далі було змінено пропорції по осі Z до значення 0.4. Після цього потрібно перейти в Edit Mode, обрати всі вершини та інструментом Bisect відсікати грані фігури, щоб досягти бажаного зовнішнього вигляду каменя. Відразу було вирівняно нормалі моделі, щоб майбутня текстура відображалася коректно, для цього потрібно виділити всі елементи моделі у Edit Mode та натиснути комбінацію клавіш Shift + N. Результат таких операцій відображено на рис. 4.57.

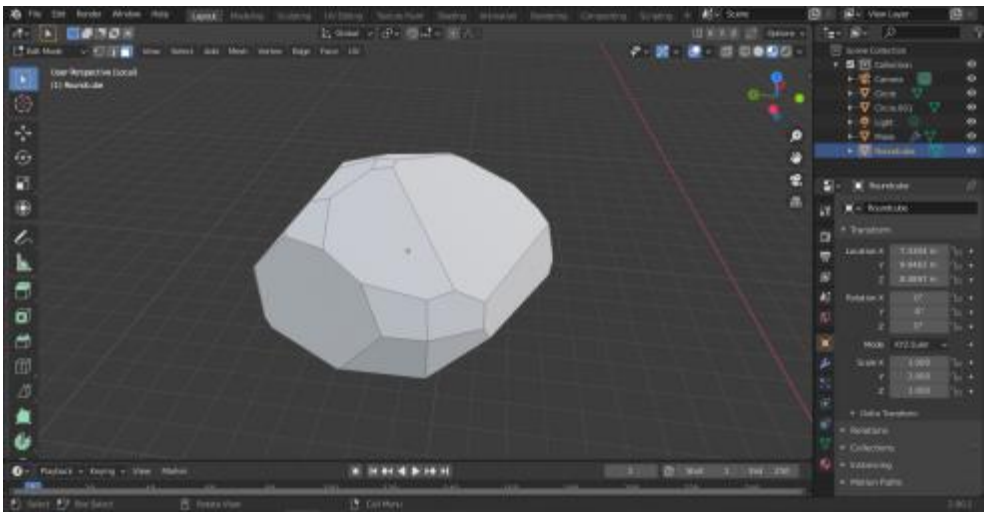

Рисунок 4.57 – Модель каменя

Для експорту моделі в Unity було повернуто фігуру по осі X на -90 градусів, але в Unity та Blender різні системи координат, тому таким чином експорт відбудеться з правильною орієнтацією. Далі потрібно натиснути Ctrl + A та обрати All transform, щоб встановити всі зміни масштабування, повороту та переміщення фігури як стандартні. Далі за шляхом File – Export – FBX (.fbx) вказати назву моделі та в параметрах експорту обрати параметри експорту як зображено на рис. 4.58.

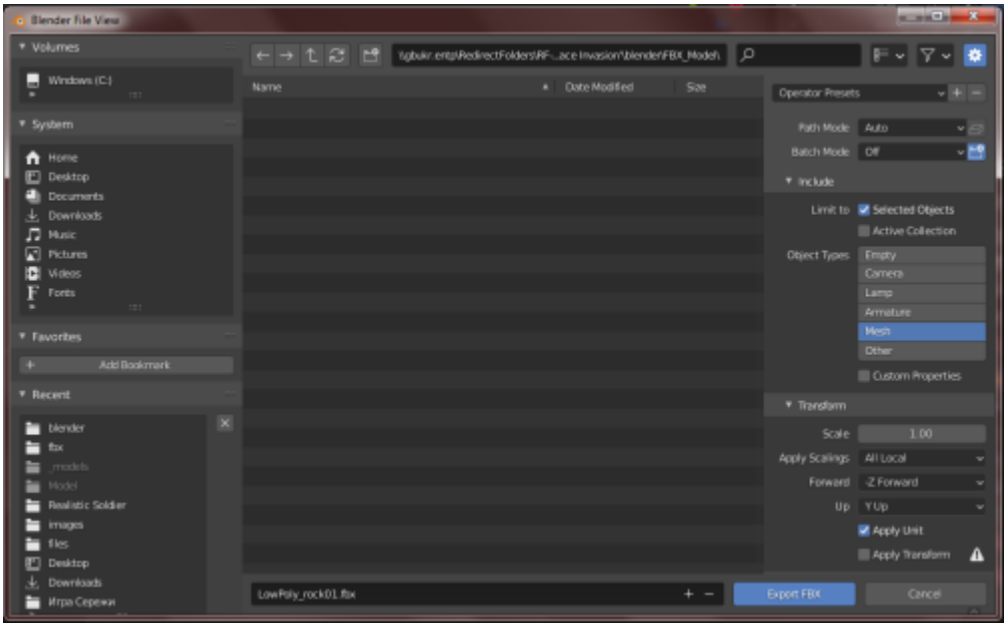

Рисунок 4.58 – Експорт моделі у форматі FBX

Наступним кроком було створено модель кристалу. Для початку було виконано створення основи для кристала за командою Shift + A – Mesh – Cylinder, у параметрах фігури вказати значення вершин 6 одиниць. Потім до фігури виконали масштабування для того, щоб вона стала вищою та вужчою, результат зображено на рис. 4.59

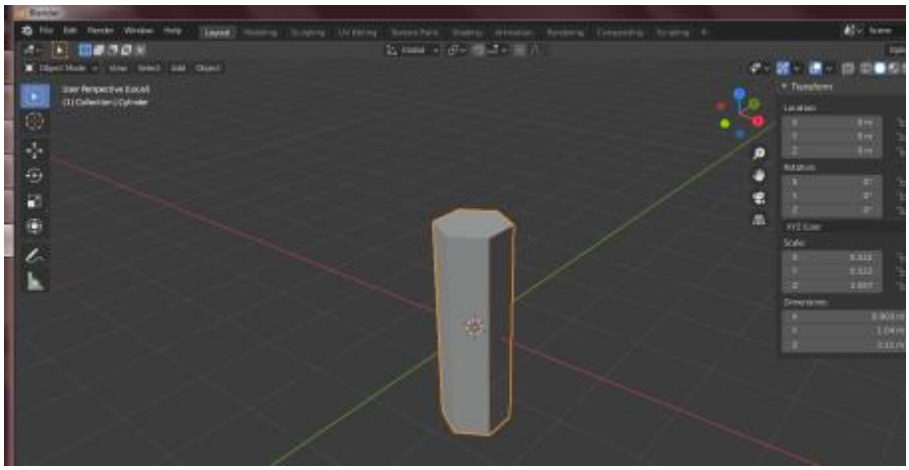

Рисунок 4.59 – Зображення основи для моделі кристалу

Далі було виконано команду Insert для верхнього полігону і вершини що утворилися було об’єднано в одну точку командою Merge – At Center. Далі за допомогою команди Bevel можна згладити кути вершин, щоб утворити щось схоже на кристал. Після цих дій модель виглядала так, як зображено на рис. 4.60.

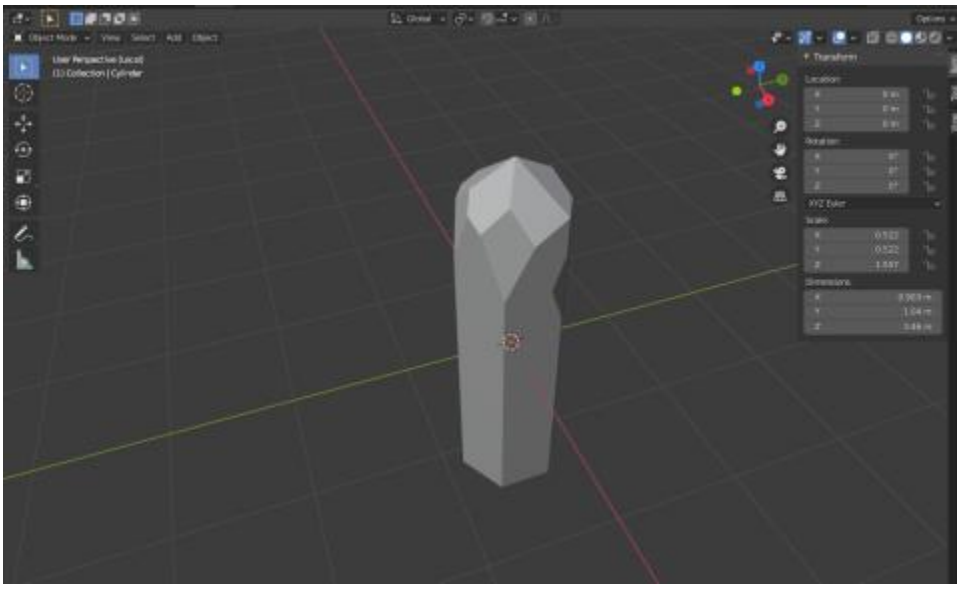

Рисунок 4.60 – Модель кристалу на поточному етапі

Наступним кроком було створення матеріалу кристалу, для цього було створено текстуру з назвою M_Crystal у якої змінили значення Base Color, Subsurface, Subsurface Color, Spectacular, Roughness й Transmisson. Після відповідних налаштувань кристал має вигляд який зображено на рис. 4.61. Далі було створено декілька варіантів кристалу, щоб обрати кращий серед них, який буде експортовано у форматі FBX до проекту Unity.

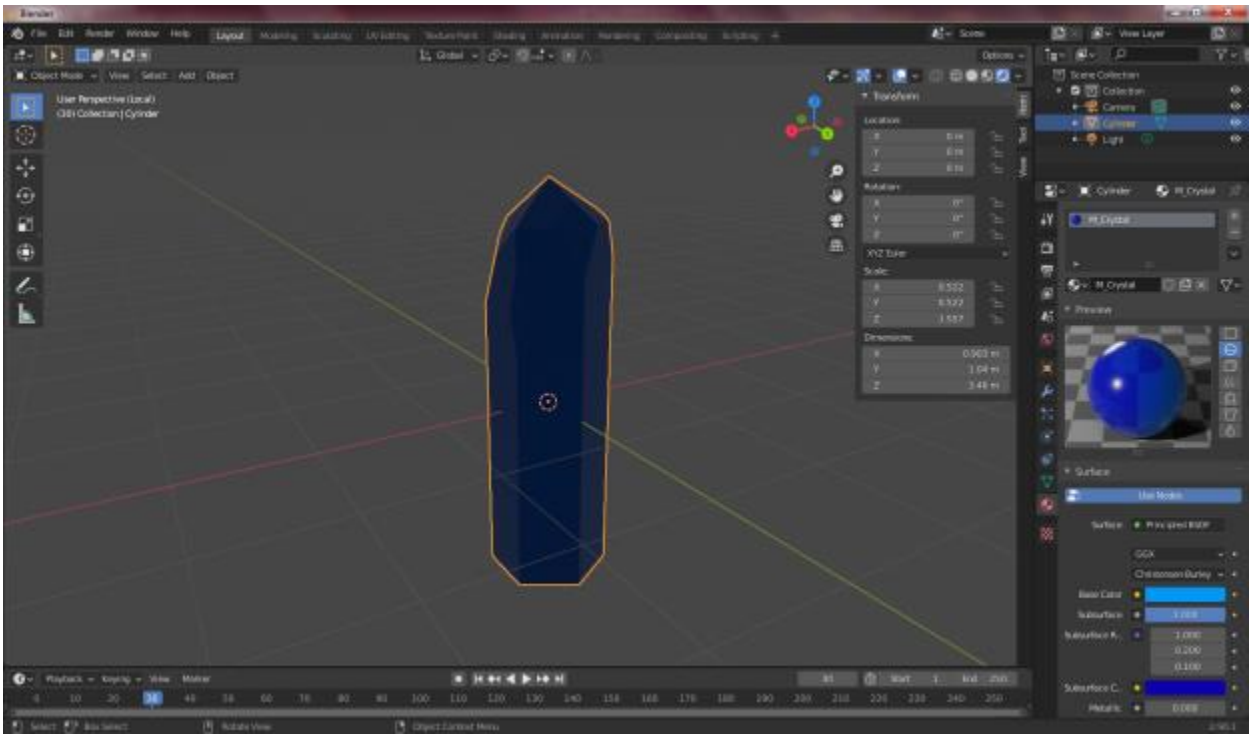

Рисунок 4.61 – Модель кристалу та його матеріал.

Наступною створювалася модель об’єкту для появи нових ворогів. Для цього спочатку створили елемент Circle, простір якого за допомогою клавіші F заповнили, потім командою Insert зробили внутрішнє коло та відразу клавішею X видалили нові Faces. Далі клавішею E видавили полігони вгору, виділили грані внутрішнього кола та подвійним натисканням G його трохи розширили. У результаті таких дій модель матиме вигляд який зображено на рис. 4.62.

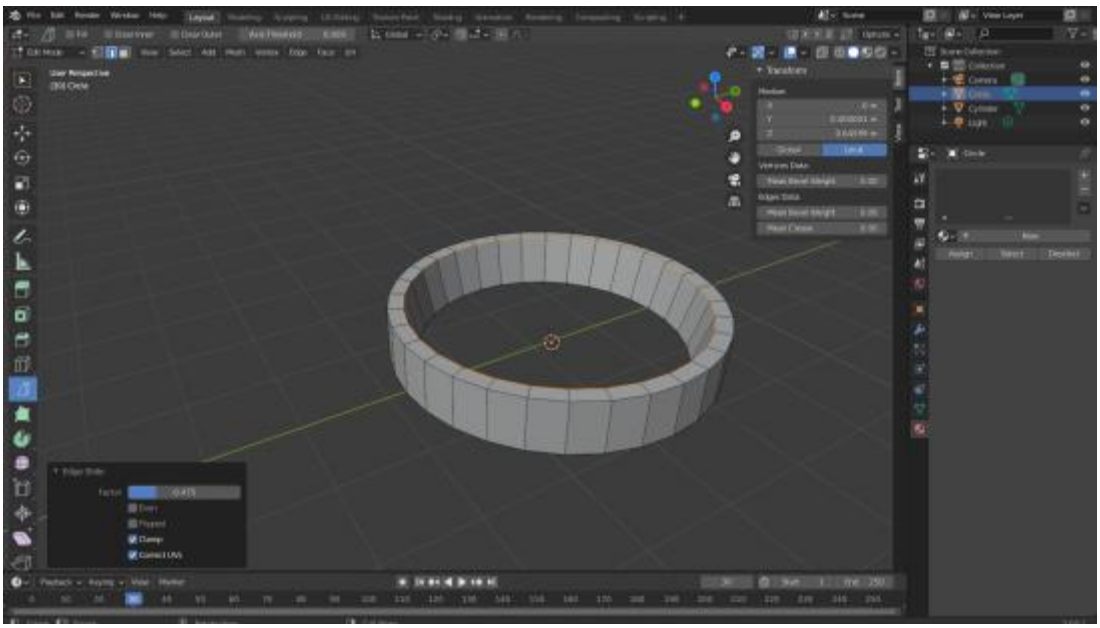

Рисунок 4.62 – Модель об’єкту на поточному етапі

Після цього комбінацією Ctrl + R створили грань у вигляді кола в середині фігури. Далі обрали полігони по колу у нижній частині бокової сторони та витіснили їх у напрямку нормалей. Потім обрали по кругу верхнє ребро і натиснувши 2 рази G зробили невелику фаску у моделі. Далі додали створений раніше кристал у середину цієї моделі отримавши результат з рис. 4.63.

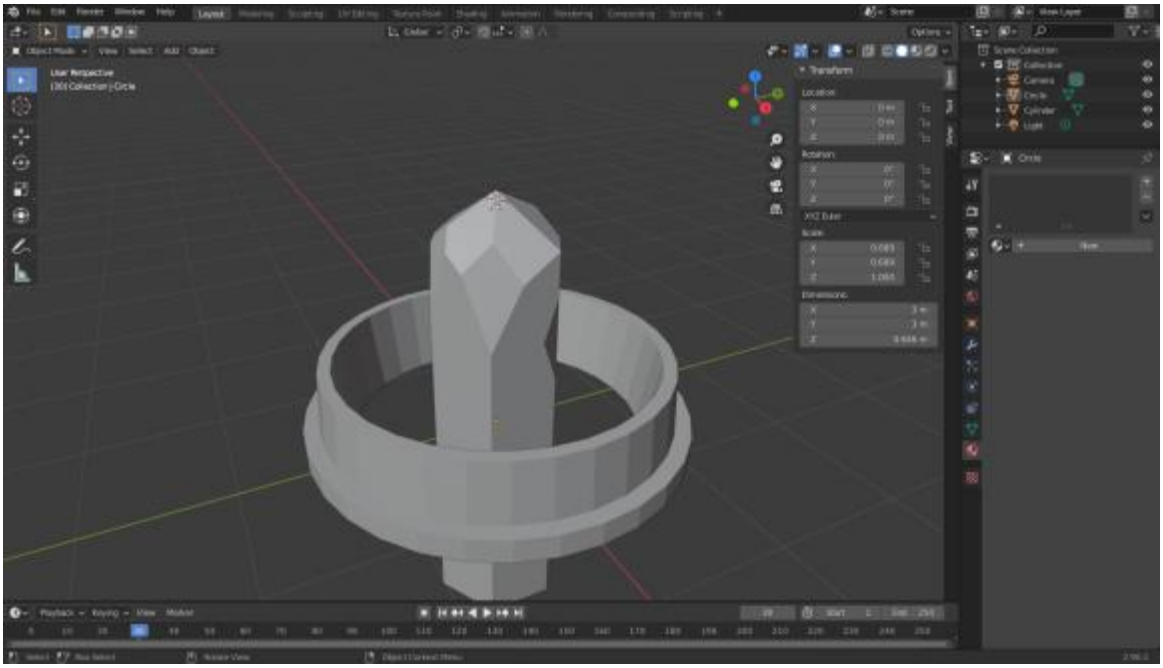

Рисунок 4.63 – Модель об’єкту на поточному етапі

Наступним кроком було додано Circle з 12 ребрами в середину моделі між кристалом та зовнішнім колом. Потім в Edit Mode комбінацією клавіш витягуємо грані по осі Z. Потім частину полігонів фігури було видалено, а решту витягнуто у напрямку нормалей. Далі було зроблено дублікат нової фігури, збільшено її в масштабі та повернуто на деякий кут. У результаті модель має вигляд як на рис. 4.64.

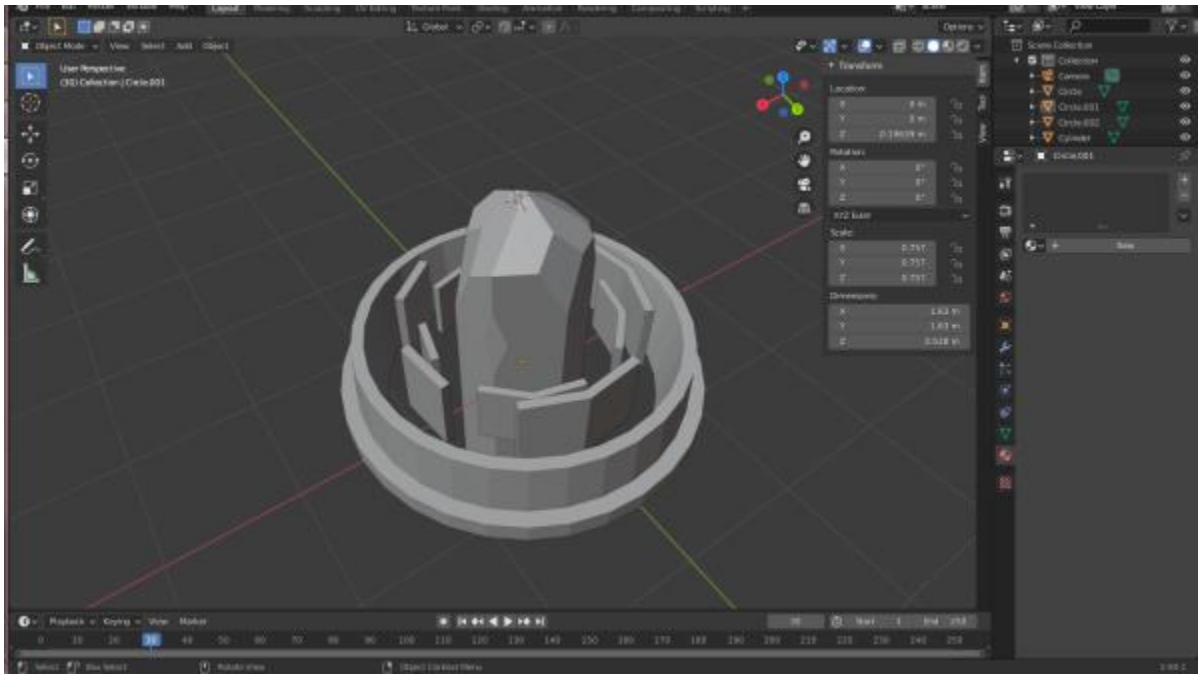

Рисунок 4.64 – Варіант моделі після до надання текстур

Останнім кроком моделювання залишається додати текстури, повернути модель перед експортом по осі X, об’єднати фігуру в одне ціле комбінацією клавіш Ctrl + J та експортувати у формати FBX. Зображення фігури після надання текстур наведено на рис. 4.65.

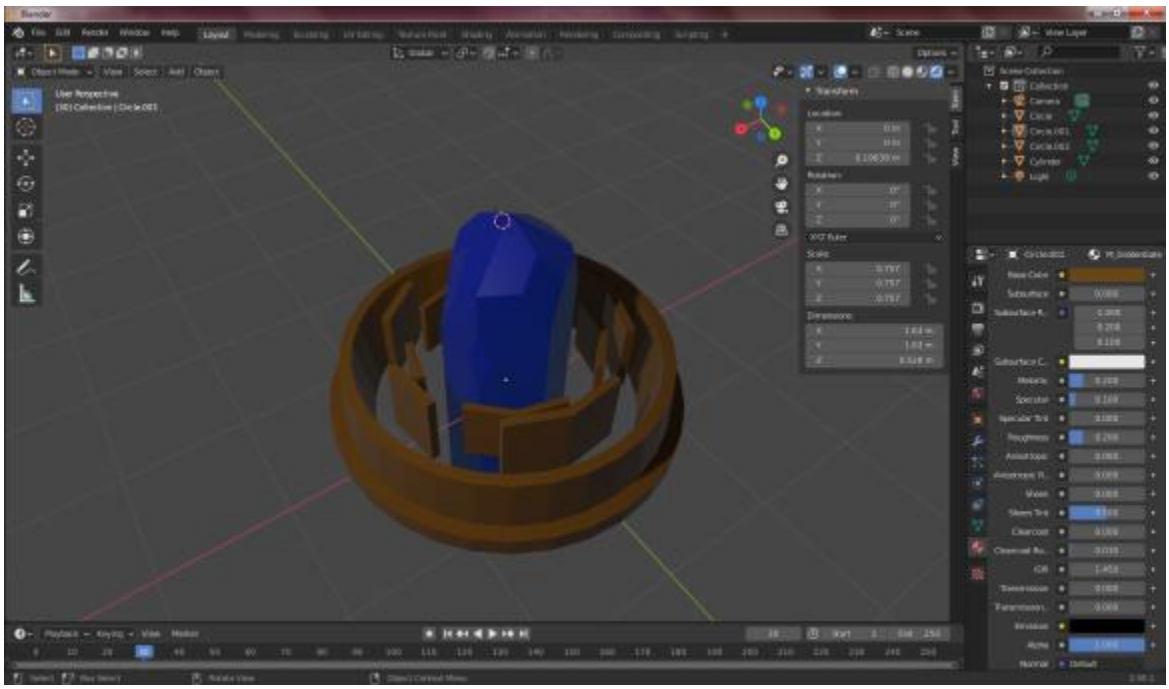

Рисунок 4.65 – Вигляд моделі після надання текстур

Потім була створена модель низько полігонального дерева. Для цього після створення нового файлу Blender виділили куб, який зазвичай створюється в сцені по замовчуванню, та командою Merge – At Center об’єднали всі вершини в одну точку. Після цього клавішею E можна витягувати нові ребра від точки, у майбутньому це будуть гілки дерева. Після закінчення моделювання гілок дерева до моделі було додано модифікатор Skin. Модель після додавання модифікатора зображена на рис. 4.66.

Рисунок 4.66 – Модель дерева після додавання модифікатора

Після додавання модифікатора деякі вершини були занадто великими, тому їх було змасштабовано до меншого розміру і потім додано модифікатор Subdivision Surface, таким чином дерево стало реалістичнішим. Далі на вершини гілок було додано об’єкт Icosphere різного масштабу який моделює листя. Після цього було створено текстури для нього та експортовано модель у файли проекту. У результаті текстурування дерево набуло вигляду із зображення на рис. 4.67.

Рисунок 4.67 – Тривимірна модель дерева

Після створення моделей оточення повернемося у проект Unity для наповнення сцени DesertOfCrystal об’єктами оточення. Для початку додамо на сцену створену модель кристалу с кільцем навколо нього, яка буде дочірнім елементом об’єкту EnemySpawner. До цього об’єкту додали компонент Capsule Collider та створили скрипт EnemySpawner (рис. 4.69), який буде створювати нових ворогів через задані часові інтервали. Його повний програмний код наведено у додатку Б.

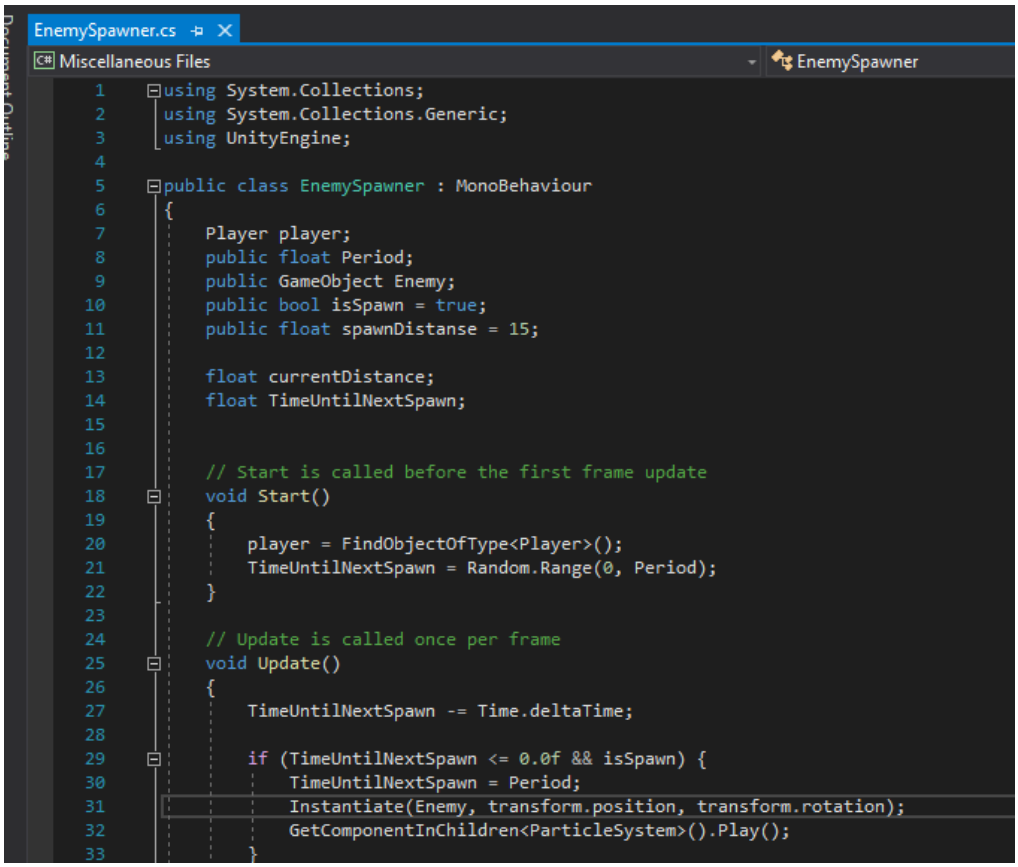

Рисунок 4.69 – Код скрипту для генерації нових ворогів

Також до об’єкта EnemySpawner додано дочірній генератор частинок, який можна продублювати у гравця. Після розташування генератору у потрібному місці йому було змінено базовий колір частинок на синій. Partycle System буде спрацьовувати під час генерації нового ворога. Параметри об’єкту EnemySpawner зображено на рис. 4.70. Після цього з об’єкта EnemySpawner створено Prefab.

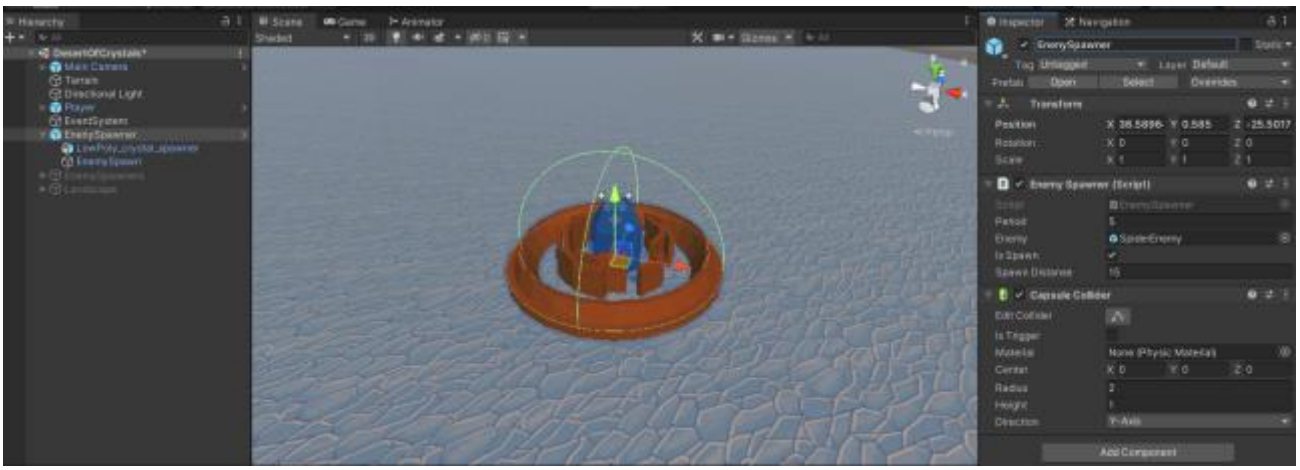

Рисунок 4.70 – Параметри об’єкта EnemySpawner

Також у проект було імпортовано модель футуристичної аптечки з інтернету разом з текстурами, вона буде з визначеною ймовірністю залишатися після смерті ворога надаючи можливість гравцю відновити частину свого здоров’я.

Для подальшого налаштування її було додано на сцену з назвою Healing_Box, а потім додали компоненти BoxCollider, AudioSource та було створено власний скрипт HealthBox, (рис. 4.71) повний скрипт якого включено до додатка Б. Далі з цього об’єкту створено Prefab у каталогах проекту, а параметри його компонентів зображено на рис. 4.72.

HealthBox.cs | EnemySpawner.cs

Miscellaneous Files | HealthBox

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

public class HealthBox : MonoBehaviour
{
    BoxCollider boxCollider;
    Player player;
    public AudioClip pickupSound;
    public float valueHealth;

    // Start is called before the first frame update
    void Start()
    {
        player = FindObjectOfType<Player>();
        boxCollider = GetComponent<BoxCollider>();
    }

    void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.tag == "Player")
        {
            GetComponent<AudioSource>().PlayOneShot(pickupSound, 0.5f);
            player.UpdateHealth(valueHealth);
            Debug.Log("HealthBox: pickup - OK");

            Destroy(boxCollider);
            Destroy(gameObject);
        }
    }
}
```

Рисунок 4.71 – Програмний код компоненту Health_Box

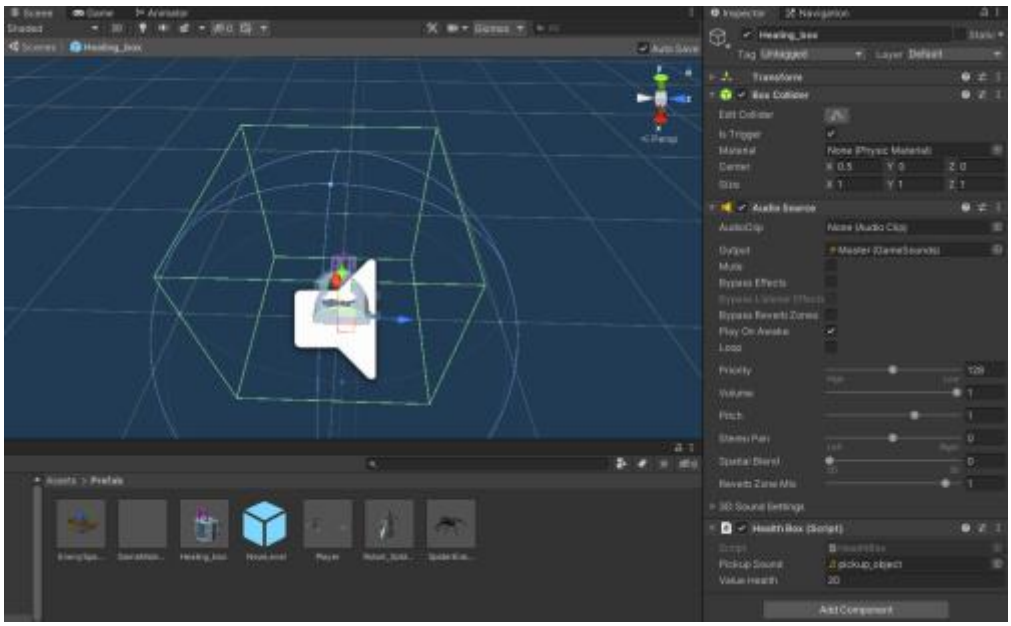

Рисунок 4.72 – Параметри об’єкта Healing_Box

4.5 Завершення налаштування додатку

Наступним кроком було наповнення ігрового рівня DesertOfCristals ігровим оточенням. Для цього на сцену додали раніше імпортовані моделі кристалів та каменів. Після їх розмноження на ігровій сцені для них додали колайдер, встановили значення Layer Ground та позначили що вони статичні об’єкти для прорахунку навігації по рівню.

Також створено примітиви кубу з назвою barrier, які виступають у ролі обмеження ігрової території з кожного боку Terrain. У barrier видалено компонент Mesh Filter для того, щоб він став прозорим. Зовнішній вигляд ігрової сцени, після наповнення оточенням, зображено на рис. 4.73.

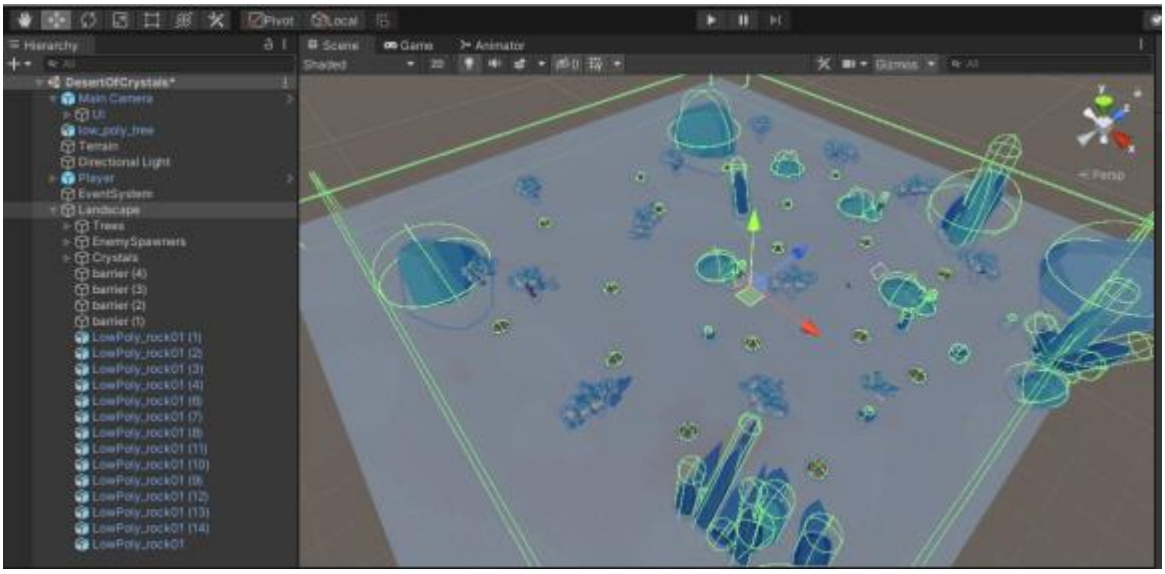

Рисунок 4.73 – Загальне зображення ігрового рівня

Додатково було змінено параметри об’єкту Directional Light, його позиція на ігровому рівні, колір світла, кут нахилу. Детальніше параметри відображені на рис. 4.74.

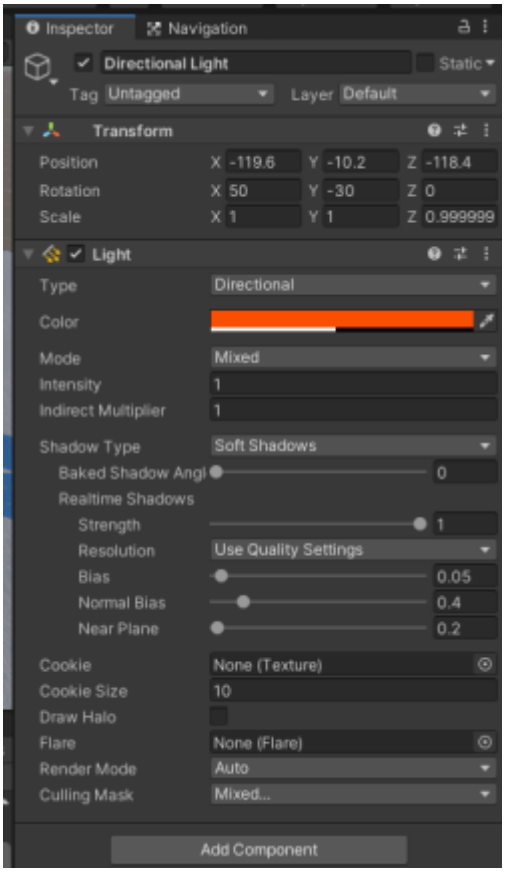

Рисунок 4.74 – Параметри об’єкта Directional Light

На цьому етапі доповнимо конфігурацію інтерфейсу головного меню. Після завантаження сцени, додамо до об'єкта SelectLeveMenu створений скрипт LoadResultOfGame (рис. 4.75) та встановимо необхідні параметри.

Скрипт відповідає за завантаження з пам’яті гри результатів проходження ігрових рівнів, його повний програмний код наведено у додатку Б. Також для цього об’єкту створено скрипт MenuControls (рис. 4.76) для завантаження необхідного рівня. Зображення параметрів об’єкту SelectLeveMenu наведено на рис. 4.77.

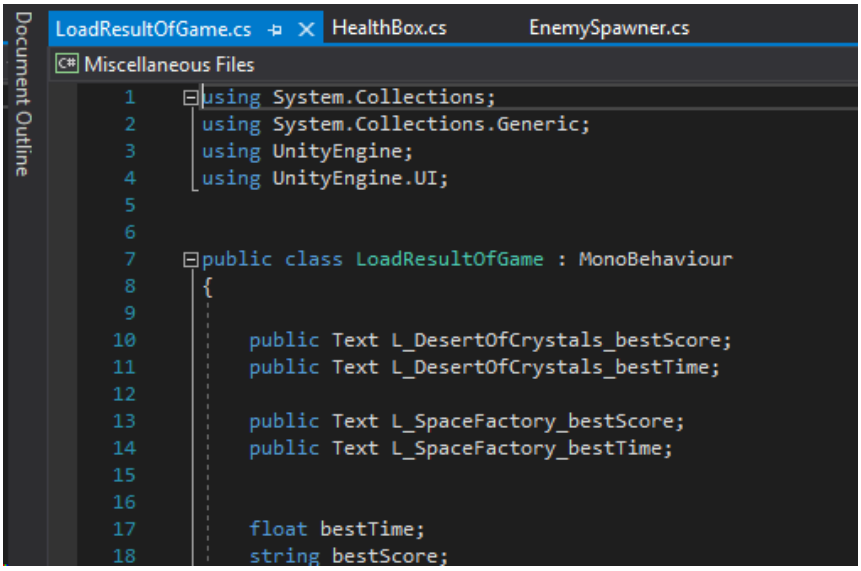

Рисунок 4.75 – Програмний код завантаження кращих результатів

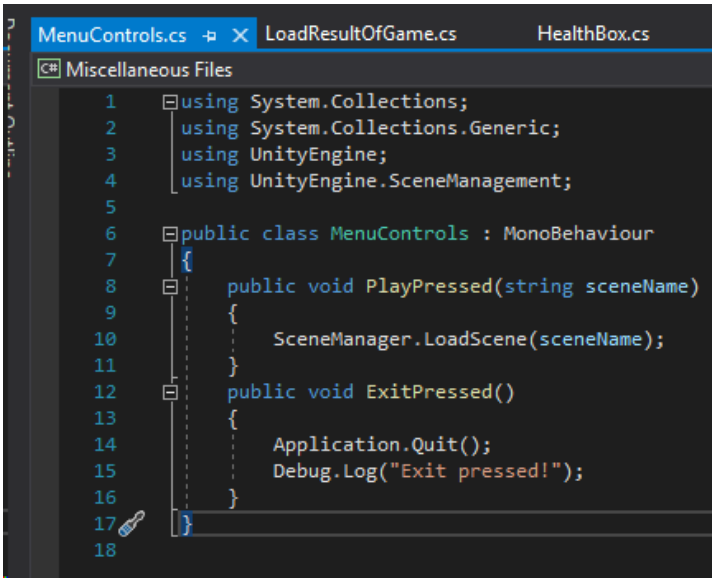

Рисунок 4.76 – Скрипт для завантаження ігрового рівня

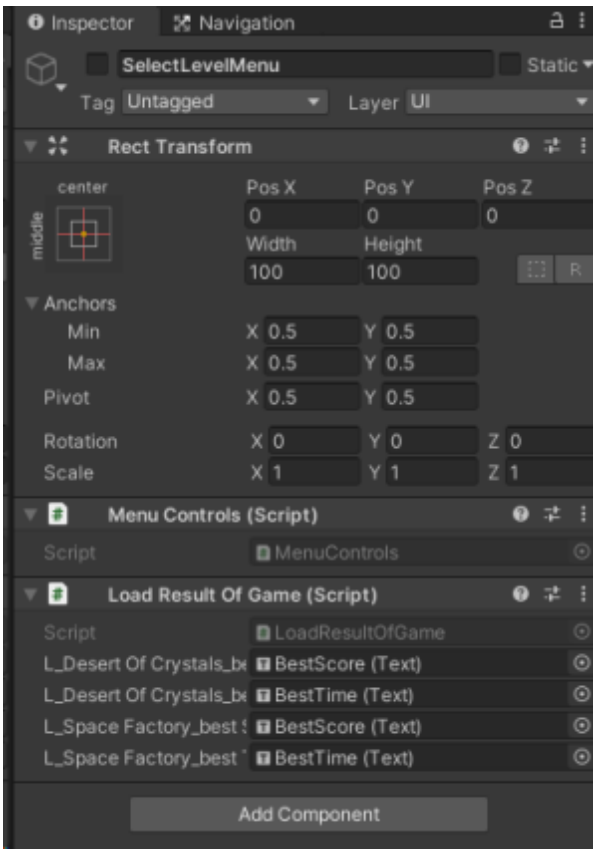

Рисунок 4.77 – Параметри об’єкта SelectLevelMenu

Далі додамо до кнопки StartBtn, в об’єкта MainMenu, виконання методу LoadBestResultOfGames зі скрипту LoadResultOfGame. Для цього додамо об’єкт SelectLevelMenu у метод OnClick() кнопки StartBtn. Також до всіх кнопок меню, у подію OnClick, додамо об’єкт GameSounds та оберемо метод AudioSoure.Play(), щоб при натисканні на кнопки меню звучав відповідний звук. Приклад додавання GameSound та оновленої кнопки StartBtn зображено рис. 4.78.

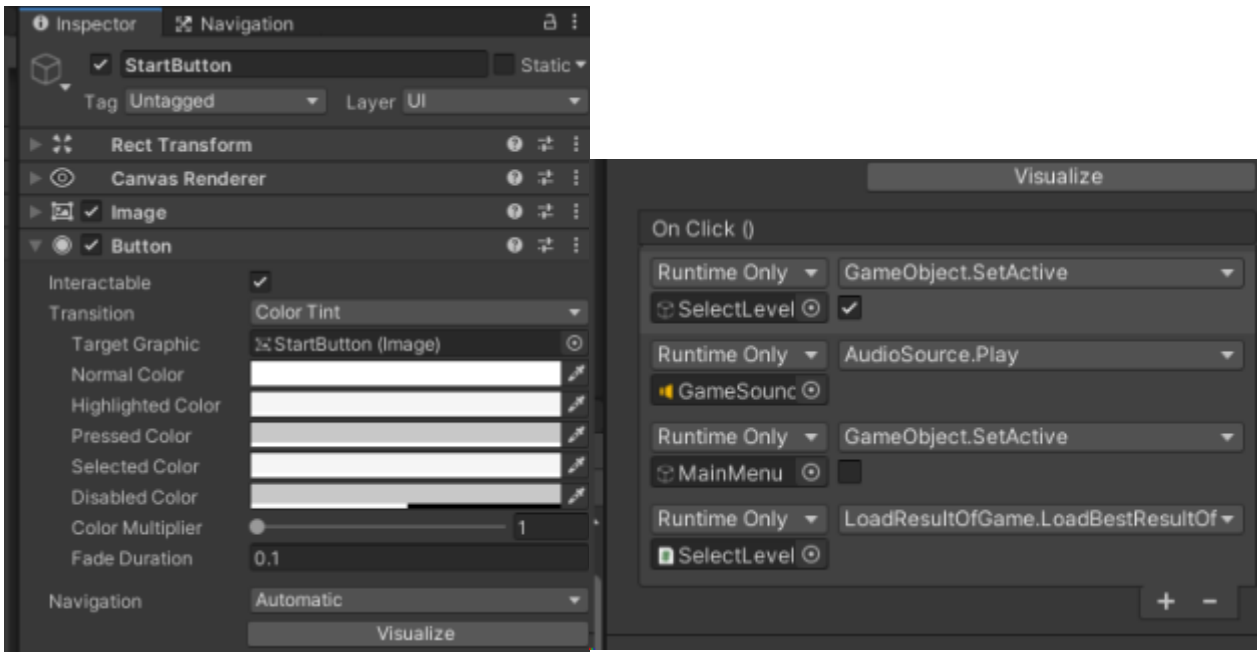

Рисунок 4.78 – Оновлені параметри кнопки StartBtn

Далі було налаштовано завантаження потрібних ігрових рівнів. Для цього у об’єкті SelelctLevelMenu обрано кнопку старту відповідного рівня та додано у її подію OnClick() об’єкт SelelctLevelMenu. У ньому викликано метод PlayPressed зі скрипту MenuControls та у текстовому полі вказано назву рівня який потрібно завантажити. Параметри кнопки після налаштування зображено на рис. 4.79

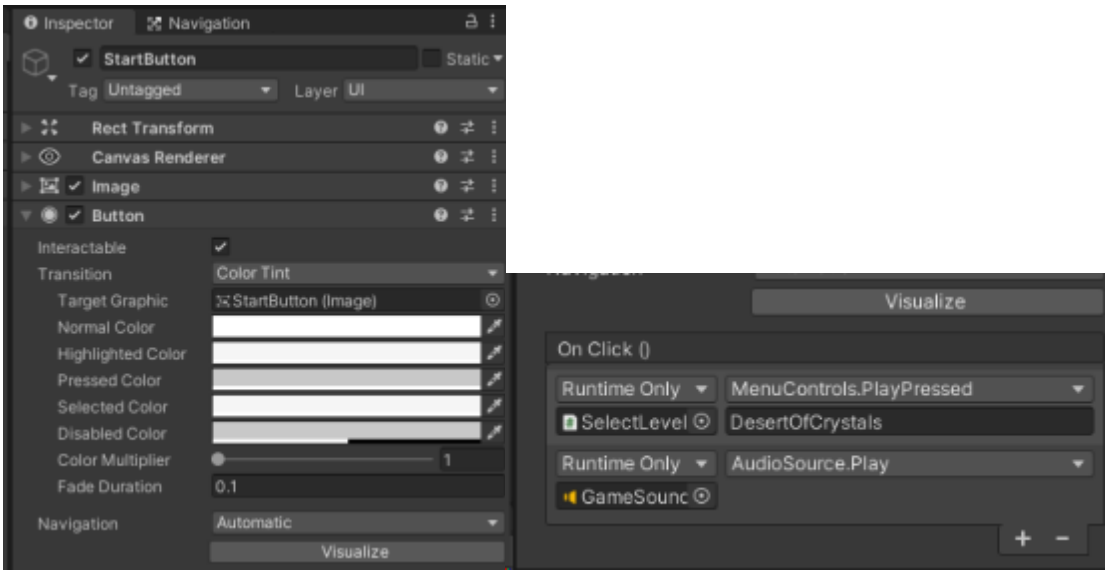

Рисунок 4.78 – Налаштування обробки події натискання на кнопку запуску ігрового рівня

4.6 Налаштування проекту та створення фінальної збірки

Після створення основних елементів гри можна перейти до фінальної частини розробки додатку. До неї можна віднести налаштування збірки проекту (налаштування орієнтації, платформи, іконки гри), додавання нових ігрових рівнів та тестування додатку.

Спочатку була додана іконка ігрового додатка, для цього у налаштуваннях проекту Build Settings у вікні Player Setting у параметри Default Icon обрана іконка яка була створена у Adobe Illustartor (рис. 4.79).

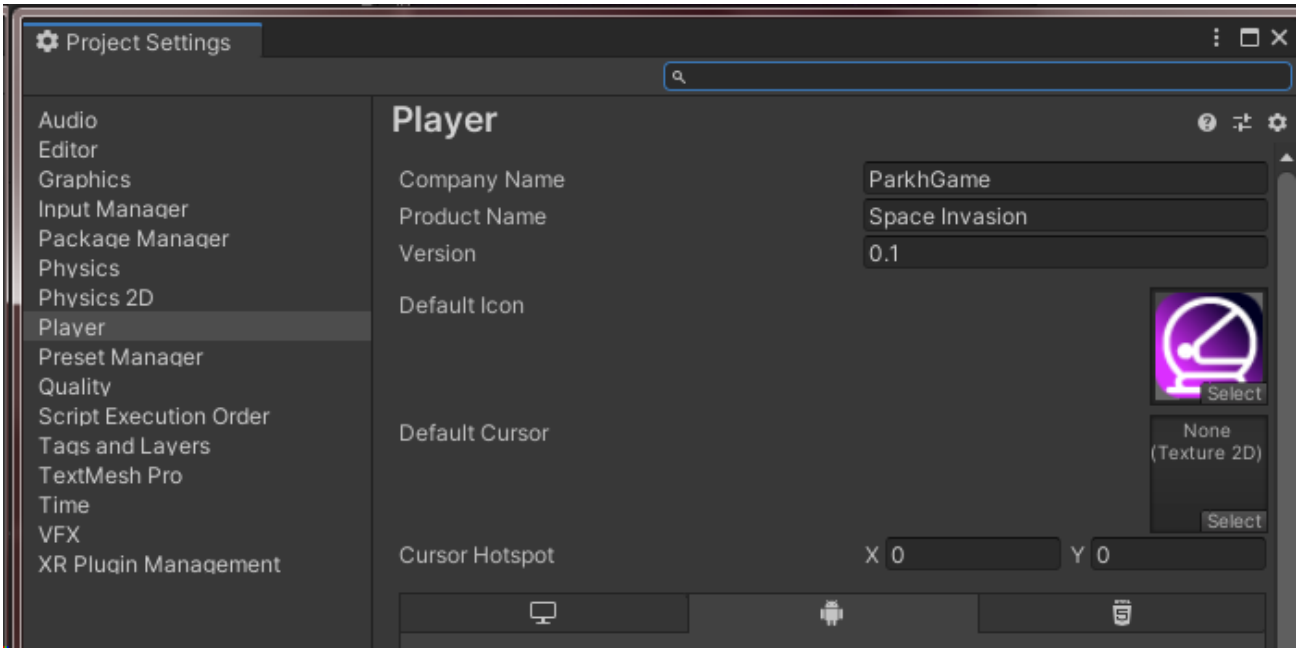

Рисунок 4.79 – Встановлення іконки для додатку

Через те, що гра орієнтована на портретну орієнтацію пристрою, у параметрі Default Orientation вказана стандартна орієнтація саме Portrait без можливості зміни (рис. 4.80).

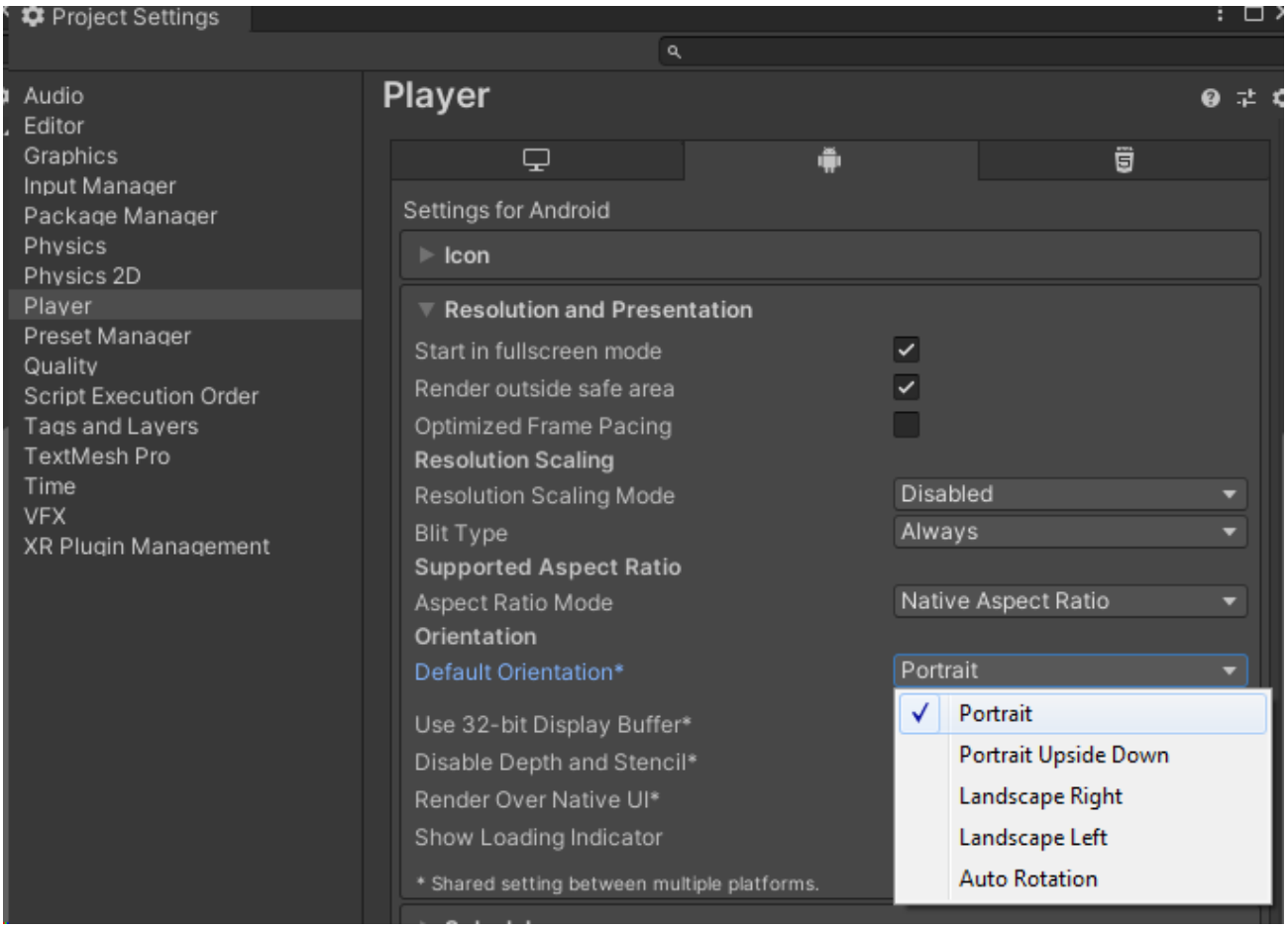

Рисунок 4.90 – Налаштування орієнтації додатку

Через те, що елементи гри були розроблені уніфікованими, створення нових ігрових рівнів вже не займе так багато часу. Для створення нового рівня достатньо зберегти першу сцену методом «Save Scene As» під новою назвою. Далі додати у рівень у менеджер рівнів, додати до сцени інше оточення за бажанням і вказати на нову сцену посилання у кнопках завантаження ігрових рівнів.

Для прикладу до проекту було завантажено з AssetStore asset під назвою RPG_FPS_game_assets_industrial. Створена сцена з назвою SpaceFactory, далі на неї розміщені нові об’єкти оточення з асету та виконано програмування території руху персонажу. На рис. 4.91 зображена нова сцена для гри з використанням створених ключових Prefab та новими об’єктами оточення.

Рисунок 4.91 – Приклад другого ігрового рівня

Після завершення створення ігрових рівнів достатньо натиснути на кнопку «Build» у налаштуваннях збірки, обрати місце та назву для збереження виконуючого файлу (рис. 4.92) та почекати доки Unity скомпілює ігровий додаток (рис. 4.93).

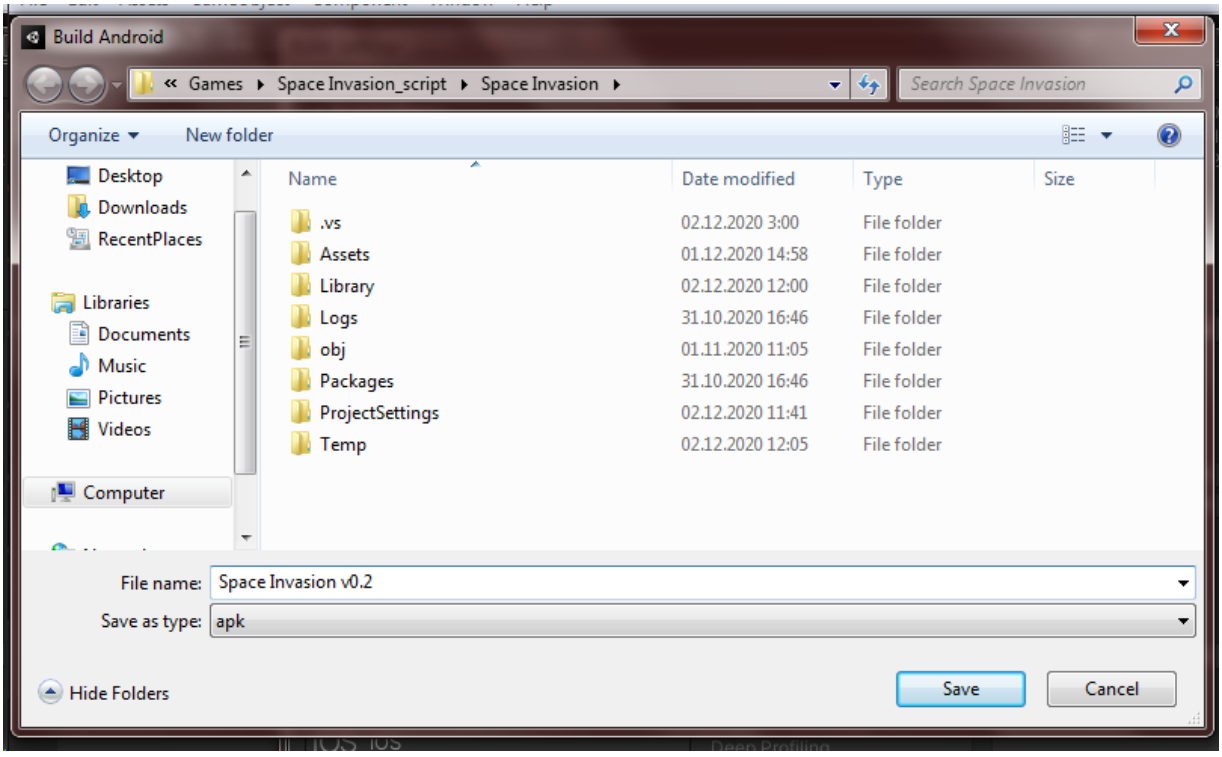

Рисунок 4.92 – Шлях для збереження apk файлу проекту

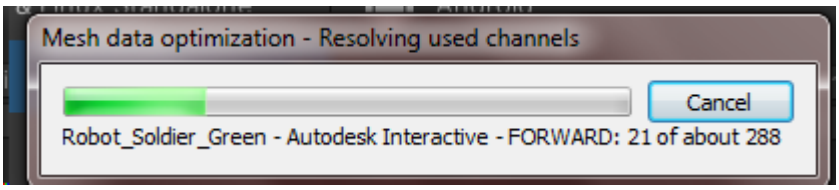

Рисунок 4.93 – Компіляція виконуючого файду проекту

4.7 Тестування мобільного ігрового додатка

Під час розробки проекту, ігровий додаток постійно тестувався, виявлені помилки виправлені.

Спочатку було протестовано робота головного меню, а саме реагування на натискання різних його кнопок. Вони реагують, інтерфейси головного меню змінюються (рис. 4.94).

Рисунок 4.94 – Головне меню

Далі було протестована зміна значення гучності ігрових звуків та музики. Для цього під час запуску додатку значення сладерів змінювались на мінімум, проект перезапускався, значення гучності зберігалися у всіх сценах (рис. 4.95).

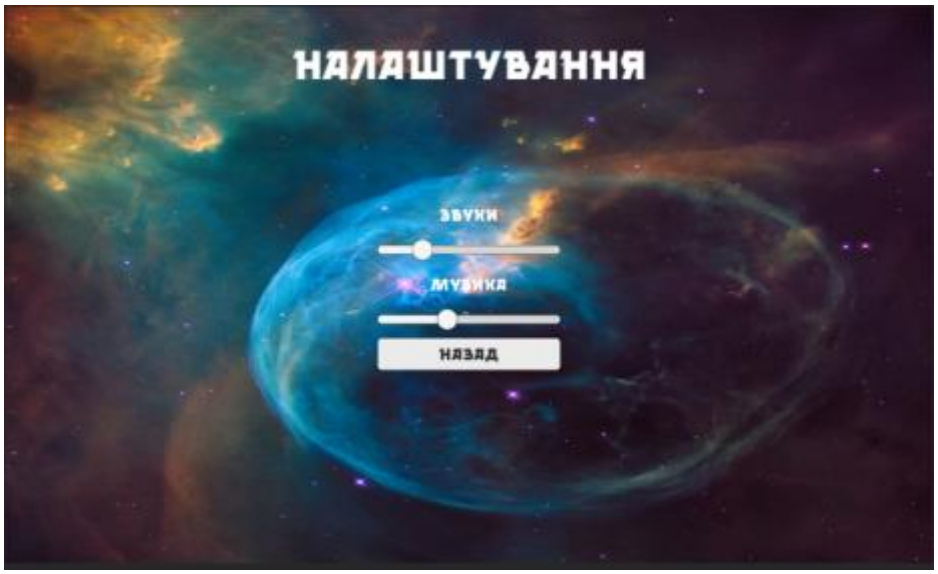

Рисунок 4.95 – Меню «Налаштування»

Далі тестування підлягало меню вибору ігрового рівня, а саме завантаження потрібного рівня, коректність відображення результатів проходження рівня, та поведінка додатку якщо результати проходження відсутні (рис. 4.96).

Рисунок 4.96 – Інтерфейс вибору ігрових рівнів

Після повного тестування інтерфейсів головного меню були протестовані ігрові інтерфейси та ігрові рівні у цілому. Для початку протестовано поведінка гравця на зміну положення джойстиків для руху, повороту та стрільби. Також підрахунок балів за вбивство ворогів та підрахунок часу від моменту початку проходження рівня.

Також в цей час була протестована поведінка ворогів: пошук гравця; атака гравця; смерть ворога; поява лікувального елементу після смерті ворога; відновлення здоров’я персонаж; реагування об’єкту створення ворогів на присутність персонажу; Всі елементи працюють згідно зі створеною логікою (рис. 4.97).

Рисунок 4.97 – Тестування ігрових механік

Далі було протестовано інтерфейс меню паузи (рис. 4.98) та опрацювання подій смерті персонажу (рис. 4.99).

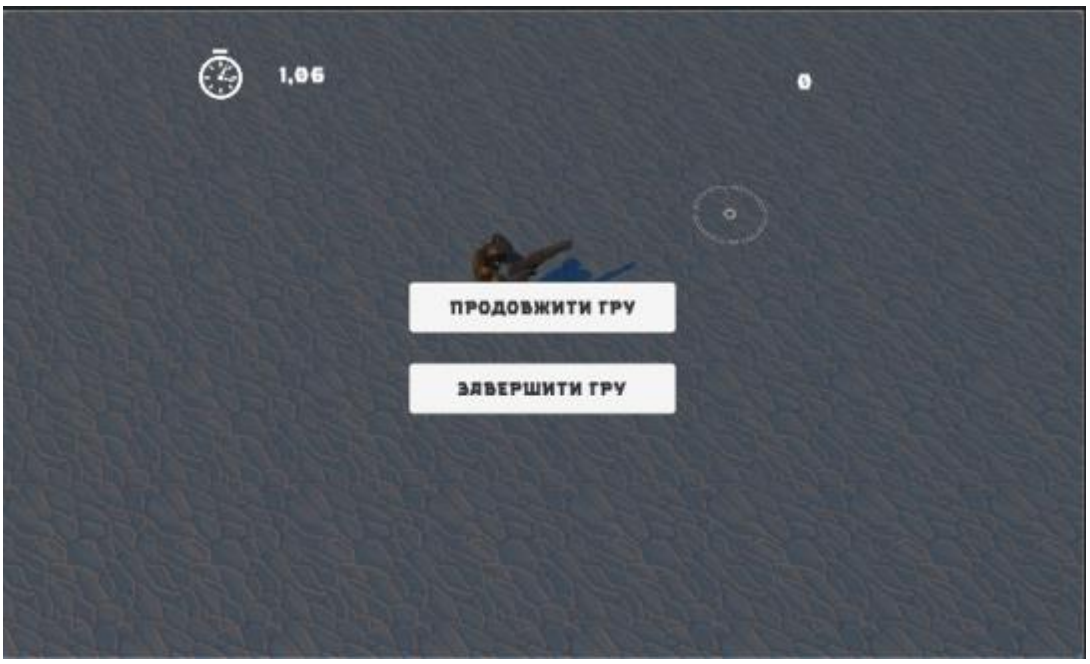

Рисунок 4.98 – Інтерфейс паузи під час гри

Рисунок 4.99 – Інтерфейс після смерті гравця

Останнім етапом тестування перевірено роботу зберігання результатів проходження ігрового рівня. Для цього потрібно набрати більше балів ніж попереднього разу або за умови однакової кількості балів зробити це за швидший

час. Результат тестування зображено на рис. 4.100. Під час тестування ігрового додатка усі анімації, функціональні можливості та інтерфейси працювали без збоїв.

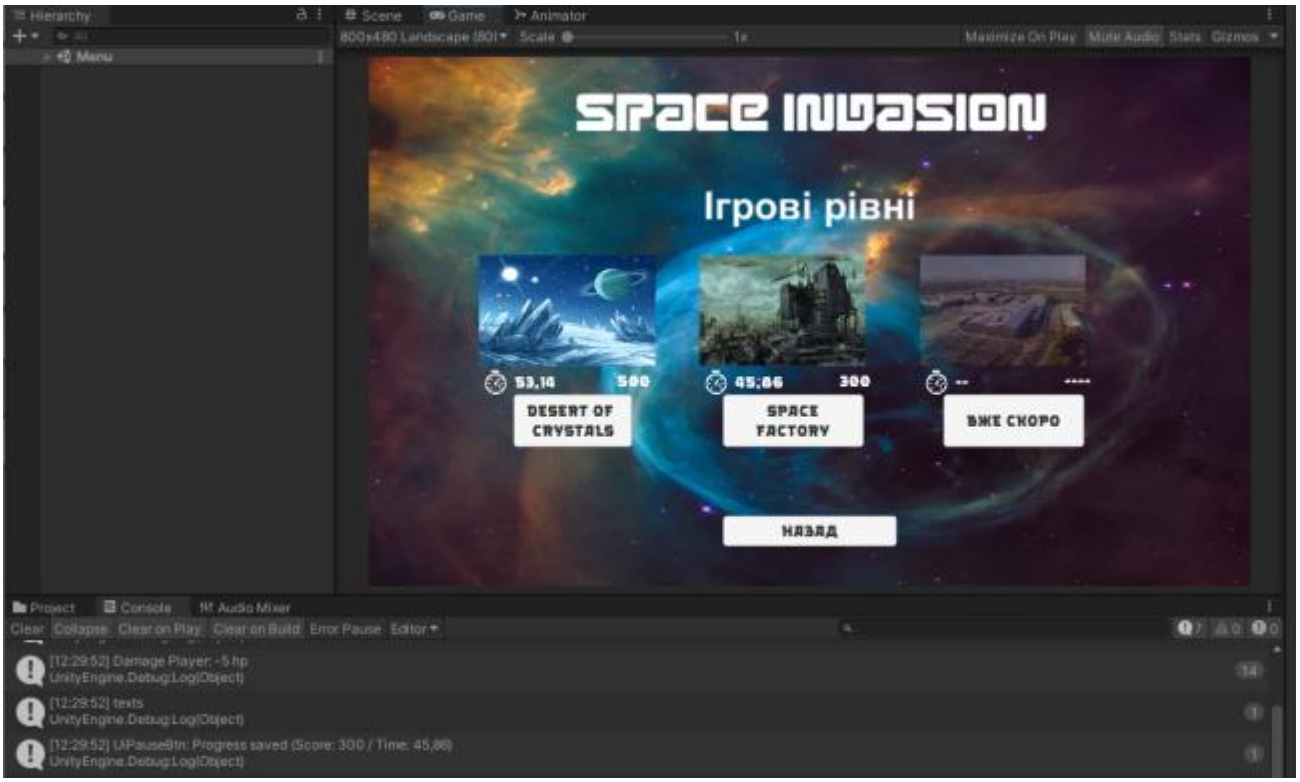

Рисунок 4.100 – Тестування зберігання ігрового прогресу рівня

ВИСНОВКИ

Мобільна ігрова індустрія на сьогодні є вагомим рушієм на ринку розваг. Уже минулого року прибуток від мобільних платформ перевищив дохід комп'ютерного геймінгу. Особливо популярні ігри у жанрі шутерів, вони дають можливість змагатися з іншими людьми та постійно покращувати власні навички, тому було обрано розробку проекту близьким до такого жанру.

Під час виконання роботи була розглянута та досліджена предметна область, розглянуто основні механіки шутерів для мобільних ігор. Також було розглянуто схожі, успішні представники мобільного геймінгу, на основі яких було визначено мету та задачі дослідження, функціональні вимоги до додатка та перелік інструментів та засобів реалізації.

У ході виконання етапу з проектування додатку було проведено структурно-функціональне моделювання процесів. Це дозволило функціонально декомпозувати користування додатком з точки зору гравця. Також відбулося проектування проекту за допомогою UseCase діаграм, що надає можливість описати взаємодію гравця з ігровим додатком, те як він буде реагувати на дії з боку користувача.

У результаті розробки був створений мобільний додаток «Space Invasion». Були створені різні інтерфейси для взаємодії користувача з додатком, власні скрипти для створення логіки та механік гри. Також було створено спрайти, тривимірні моделі об’єктів оточення, налаштовано для них матеріал та використано в оточенні одного з ігрових рівнів.

Були налаштовані моделі для імпорту персонажів, ігрові анімації, сцена, освітлення та звуки. Передбачена можливість самостійного вибору ігрового рівня для проходження, а завдяки уніфікованості елементів, створення нових рівнів не потребує великої кількості часу. Протестований ігровий додаток було скомпільовано у файл з розширенням apk..

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Сравнение производительности ПК и смартфонов, включая iPhone 11: веб-сайт. URL: https://habr.com/ru/post/471018/ (дата звернення 21.10.20).

2. Лучшее в Google Play: веб-сайт. URL: https://play.google.com/store/apps/editorial_collection/promotion_topic_bestof2019_xfn_hub (дата звернення 22.10.20)

3. Цікава монетизація мобільних ігор: веб-сайт. URL: https://dtf.ru/mobile/40947-zanimatelnaya-monetizaciya-mobilnyh-igr/ (дата звернення 29.10.20).

4. Шутеры уже больше спорт, чем игры: веб-сайт. URL: https://habr.com/ru/post/372229/ (дата звернення 22.10.20).

5. Game Engine Architecture, Second Edition: підручник. Jason Gregory, CRC Press; 1st Edition, 2018. 272 с.

6. Приложения в Google Play: веб-сайт. URL: https://play.google.com/store/apps/details?id=com.supercell.brawlstars&hl=ru&gl=US (дата звернення 22.10.20).

7. Приложения в Google Play: веб-сайт. URL: https://play.google.com/store/apps/details?id=com.noodlecake.altosadventure&hl=ru&gl=US (дата звернення 22.10.20).

8. Level Design: Processes and Experiences: підручник. A K Peters/CRC Press; 1st Edition, 2016. 408 с.

9. Hotline Miami в Steam: веб-сайт. URL: https://store.steampowered.com/app/219150/Hotline_Miami/?l=russian (дата звернення 22.10.20).

10. Defense Acquisition University Press. Systems Engineering Fundamentals: підручник. Вірджинія, 2001. 222 с.

11. G. Booch, J. Rumbaugh, I. Jacobson. Unified Modeling Language User Guide, The 2nd Edition: підручник. Addison-Wesley, 2005. 496 с.

12. Ian Sommerville. Software Engineering, 8th edition: підручник. Pearson

Education, 2007. 843 с.

13. Unreal Engine 4 Game Development Essentials: підручник. Бріменхем, 2016. 266 с.

14. Blueprints Visual Scripting for Unreal Engine: підручник. Packt Publishing, 2015. 190 с.

15. Unity in Action: підручник. JOSEPH HOCKING, 2016. 352 с.

16. Getting Started with Unity 2018: підручник. Third Edition: A Beginner's Guide to 2D and 3D game development with Unity, Dr. Edward Lavieri, 2018. 485 с.

17. Timothy A. Budd. C++ for Java Programmers: підручник. Oregon State University, Corvallis, Oregon, 1998. 29 с. URL: http://web.engr.oregonstate.edu/~budd/Books/cforj/info/preface.pdf (дата звернення 23.10.20).

18. Learning C# by Developing Games with Unity 3D Beginner's Guide: підручник. Terry Norton, 2013. 292 с.

19. Microsoft Visual C#. Подробное руководство: підручник. Джон Шарп, 2017. 848 с.

20. Microsoft Launches Visual Studio Code, A Free Cross-Platform Code Editor For OS X, Linux And Windows: веб-сайт. URL: https://techcrunch.com/2015/04/29/microsoft-shocks-the-world-with-visual-studio-code-a-free-code-editor-for-os-x-linux-and-windows/ (дата звернення 28.10.19).

21. Visual Studio Code FAQ: веб-сайт. URL: https://code.visualstudio.com/docs/supporting/faq (дата звернення 24.10.20).

22. Sublime Forum: веб-сайт. URL: https://forum.sublimetext.com/ (дата звернення 24.10.20).

23. Справка по Adobe® Illustrator® CC: підручник, 2016. 734 с. URL: https://illustrator.demiart.ru/books/book_illustrator.pdf (дата звернення 28.10.20).

24. Creativity for all: веб-сайт. URL: https://www.adobe.com/creativecloud.html (дата звернення 26.10.20).

25. Самоучитель Blender 2.7: підручник, БХВ-Петербург, 2016. 400с.

26. Моделювання та аналіз систем. IDEF-технології. Підручник-практикум /

Черемних С.В., Семенов І.О., Ручкін В.С., 2006. 188 с.

27. IDEF0 and SADT: A Modeler's Guide / Девід А. Марка, Клемент Л. Макгоуан. OpenProcess, Inc. 2005. 392 с.

28. Введення в UML від творців мови / Граді Буч, Джеймс Рамбо, Івар Якобсон. - М .: ДМК Пресс, 2015. - 496 с.

29. Prefab: веб-сайт документаціх ігрового рушії Unity. URL: https://docs.unity3d.com/ru/530/Manual/Prefabs.html (дата звернення 25.11.20)

30. Creating and Using Scripts: веб-сайт. URL: https://docs.unity3d.com/ru/2020.2/Manual/CreatingAndUsingScripts.html (дата звернення 25.11.20)

31. Audio Mixer: веб-сайт. URL: https://docs.unity3d.com/ru/current/Manual/AudioMixer.html (дата звернення 25.11.20)

ДОДАТОК А ПЛАНУВАННЯ РОБІТ

А.1 Деталізація мети проекту методом SMART

Сутність деталізації мети проекту за допомогою SMART-методу випливає з розшифрування термінів, які формують його назву: специфічність (Specific), вимірюваність (Measurable), узгодженість (Agreed Upon), реалістичність (Realistic), обмеженість в часі (Time-related).

Правильно деталізована за допомогою SMART-методу мета дозволяє сформувати загальне уявлення про проект, яке допомагає особам, що приймають рішення, а іншим зацікавленим сторонам дає змогу зрозуміти масштаби та особливості проекту. Результат деталізації методом SMART розміщено у табл. А.1.

Таблиця А.1 − Деталізація мети методом SMART

Specific (конкретна)	Створити мобільний ігровий додаток «Space Invasion»
Measurable (вимірювана)	Створити мобільний ігровий додаток «Space Invasion» за 82 дні.
Achievable (досяжна)	Реалізація роботи здійснюється шляхом використання ігрового рушію «Unity», програми для тривимірного моделювання «Blender» та Microsoft Visual Code для комфортнішої розробки програмного коду.
Relevant (реалістична)	Для успішної розробки мобільного ігрового додатку «Space Invasion» є всі необхідні засоби, розробники достатньо кваліфіковані для виконання поставлених задач.
Time-related (обмежена у часі)	Розробити ігровий мобільний додаток «Space Invasion» на основі сформованого календарного плану. Ціль обмежена часовими рамками. Роботи повинні бути завершені згідно з календарним планом.

А.2 Планування змісту структури робіт IT-проекту (WBS)

Для декомпозиції проекту на доступні для огляду і керовані частини використовується робоча структура проекту — WBS (Work Breakdown Structure). Вона допомагає визначити задачі та підзадачі, які потрібно зробити для виконання проекту. Залежно від масштабу проекту кількість рівнів декомпозиції може бути різною, аж до виокремлення робіт, готових для включення в сіткову модель. На основі цих рівнів майбутнього проекту будуємо таблицю WBS (рис. А.1).

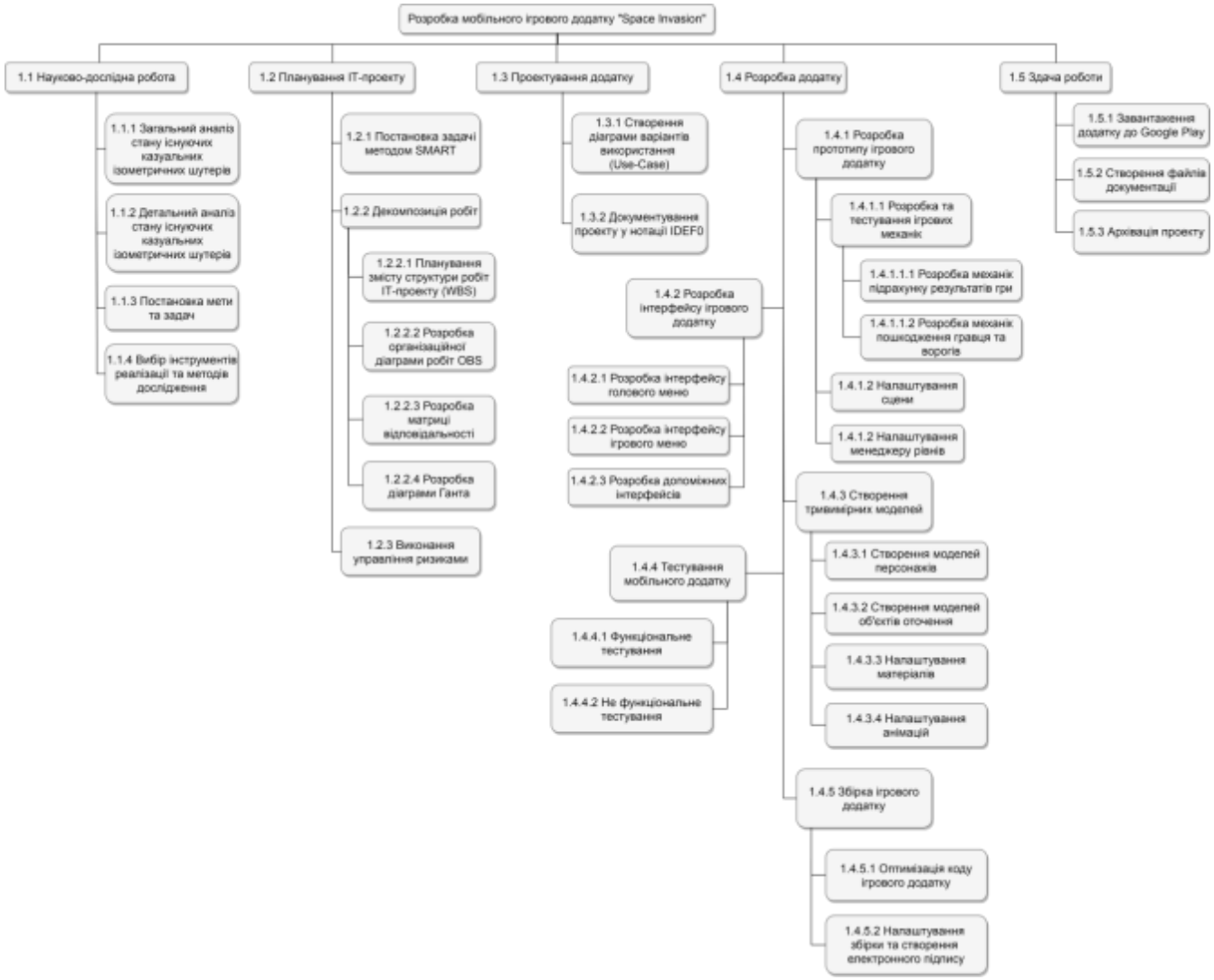

Рисунок А.1 - Діаграма декомпозиції робіт проекту (WBS)

А.3 Організаційна структура проекту (OBS)

Формування WBS логічно тягне за собою визначення відповідних виконавців цієї роботи, відповідно для цього потрібно створення організаційної структури проекту OBS (Organization Breakdown Structure). Вона визначає відносини між учасниками проекту, їх відповідальність і повноваження в процесі його реалізації.

Визначимо виконавців робіт та інших зацікавлених сторін проекту:

1. Виконавець (Пархоменко С. В.).

2. Науковий керівник (Федотова Н.А.).

Після визначення зацікавлених сторін можна переходити то розробки структури. OBS-структура організації робіт зображена на рис. А.2.

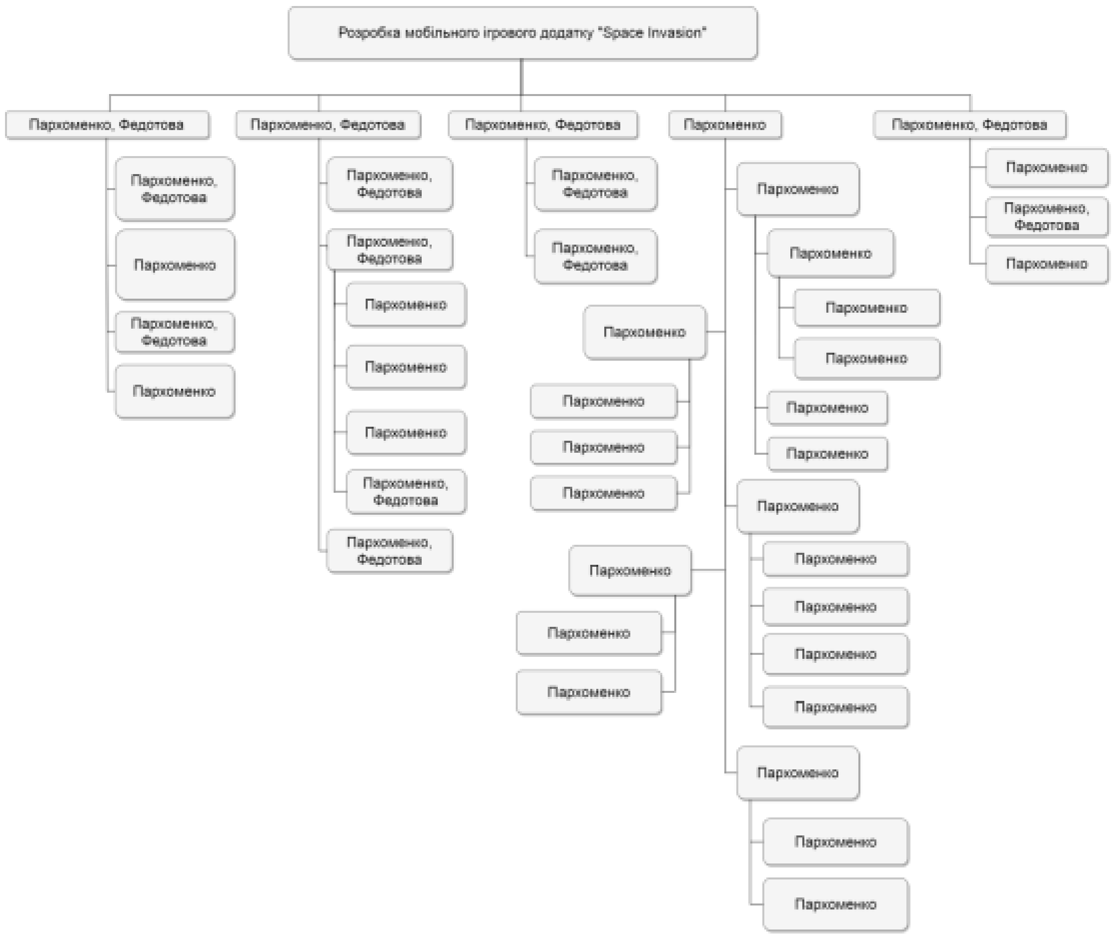

Рисунок А.2 – Діаграма організації робіт проекту (OBS)

Оскільки найпоширенішою формою проектних структур є матрична організаційна структура, то інструментом, який дозволяє вирішувати можливі проблеми і суперечки, виступає матриця відповідальності, що пов’язує WBS і OBS. Вона закріплює за кожною елементарною роботою з ієрархії певного виконавця, забезпечуючи опис і узгодження структури відповідальності за виконання пакетів робіт. Матрицю відповідальності проекту наведено у таблиці А.2.

Таблиця А.2 – Матриця відповідальності

WBS\OBS	**Пархоменко С. В.**	**Федотова Н.А.**
1 Розробка мобільного ігрового додатку "Space Invasion"	x	x
1.1 Науково-дослідна робота	x	x
1.1.1 Загальний аналіз стану існуючих казуальних ізометричних шутерів	x	x
1.1.2 Детальний аналіз стану існуючих казуальних ізометричних шутерів	x	
1.1.3 Постановка мети та задач	x	x
1.1.4 Вибір інструментів реалізації та методів дослідження	x	
1.2 Планування IT-проекту	x	x
1.2.1 Постановка задачі методом SMART	x	x
1.2.2 Декомпозиція робіт	x	x
1.2.2.1 Планування змісту структури робіт IT-проекту (WBS)	x	
1.2.2.2 Розробка організаційної діаграми робіт OBS	x	
1.2.2.3 Розробка матриці відповідальності	x	

Продовження таблиці А.2 – Матриця відповідальності

1.2.2.4 Розробка діаграми Ганта	x	x
1.2.3 Виконання управління ризиками	x	x
1.3 Проектування додатку	x	x
1.3.1 Створення діаграми варіантів використання (Use-Case)	x	x
1.3.2 Документування проекту у нотації IDEF0	x	x
1.4 Розробка додатку	x	
1.4.1 Розробка прототипу ігрового додатку	x	
1.4.1.1 Розробка та тестування ігрових механік	x	
1.4.1.1.1 Розробка механік підрахунку результатів гри	x	
1.4.1.1.2 Розробка механік пошкодження гравця та ворогів	x	
1.4.2 Розробка інтерфейсу ігрового додатку	x	
1.4.2.1 Розробка інтерфейсу голового меню	x	
1.4.2.2 Розробка інтерфейсу ігрового меню	x	
1.4.2.3 Розробка допоміжних інтерфейсів	x	
1.4.3 Створення тривимірних моделей	x	
1.4.3.1 Створення моделей персонажів	x	
1.4.3.2 Створення моделей об'єктів оточення	x	
1.4.3.3 Налаштування матеріалів	x	
1.4.3.4 Налаштування анімацій	x	

Продовження таблиці А.2 – Матриця відповідальності

1.4.4 Тестування мобільного додатку	x	
1.4.4.1 Функціональне тестування	x	
1.4.4.2 Не функціональне тестування	x	
1.4.5 Збірка ігрового додатку	x	
1.4.5.1 Оптимізація коду ігрового додатку	x	
1.4.5.2 Налаштування збірки та створення електронного підпису	x	
1.5 Здача роботи	x	x
1.5.1 Завантаження додатку до Google Play	x	
1.5.2 Створення файлів документації	x	x
1.5.3 Архівація проекту	x	

А.4 Побудова календарного графіка виконання ІТ-проекту

Календарний графік робіт будують щоб володіти реальним уявленням щодо тривалості виконання окремих робіт з урахуванням обмеження у використанні ресурсів, а також тривалості проекту у цілому з урахуванням вихідних та святкових днів.

Діаграма Ганта є реальним розподілом робіт проекту за календарними датами, тобто своєрідним розкладом виконання робіт . Зазвичай побудова діаграми Ганта відбувається у декілька кроків. На горизонталі фіксують календар у тих одиницях часу, які обрані для проекту (години, дні). Ліворуч на вертикалі розташовують найменування всіх робіт. На робочому полі, що утворилось, поставляють у вигляді прямокутників роботи, довжина яких по горизонталі відповідає їхній тривалості. Між роботами лініями вказують логічні зв’язки.

За допомогою спеціальних програм побудовано діаграму Ганта (рис. А.3).

Рисунок А.3 – Календарний план та діаграма Ганта

А.5 Управління ризиками

Під процесами управління ризиками проекту розуміється планування управління ризиками, ідентифікація і їх аналіз, вироблення методів реагування на ризики, контроль, моніторинг і управління ними у ході реалізації проекту. За допомогою процесів управління ризиками проекту намагаються підвищити ймовірність виникнення і впливу сприятливих ризиків на проект і знижують ймовірність виникнення і впливу несприятливих ризиків на проект в момент його

виконання. Побудуємо матрицю декомпозиції ризиків для нашого проекту (рис. А.5).

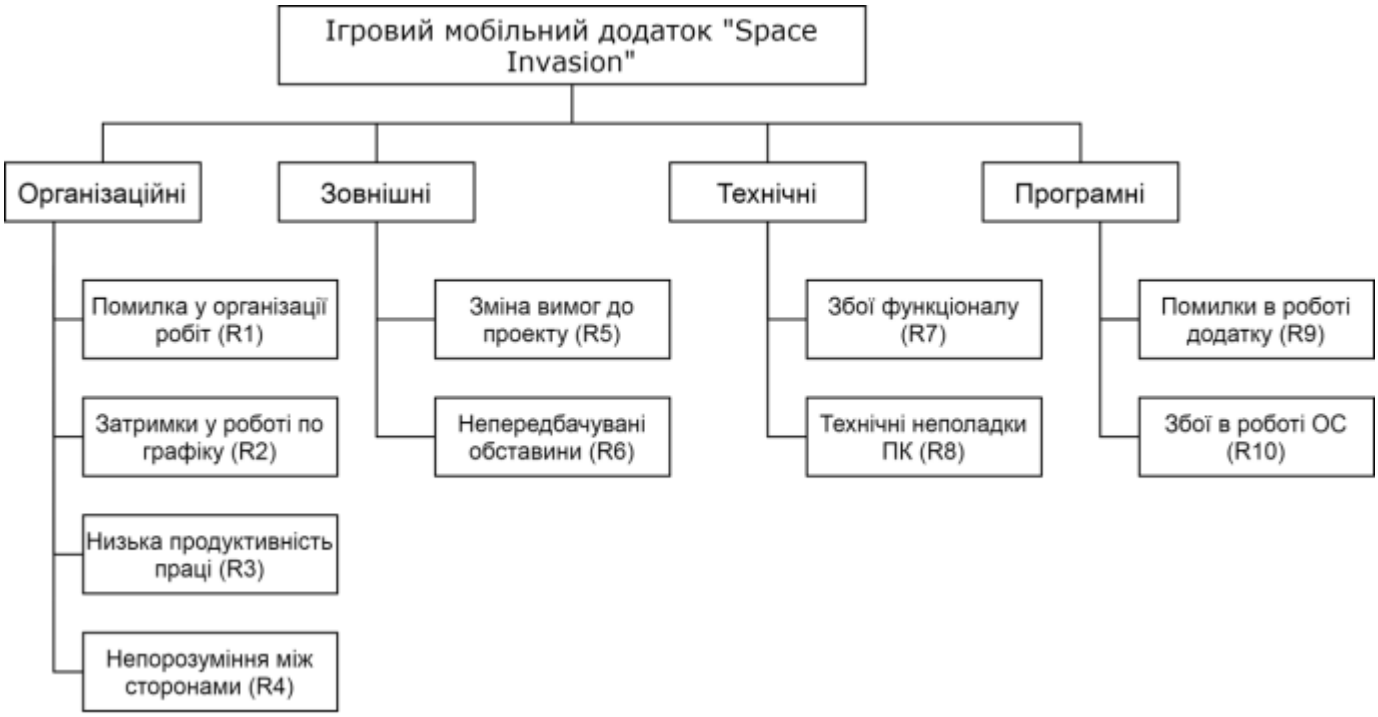

Рисунок А.5 – Матриця декомпозиції ризиків RBS

Для більш точного дослідження впливу ризиків на проект охарактеризуємо їх за показниками ймовірності виникнення (табл. А.3) та ступенями втрат (табл. А.4).

Таблиця А.3 – Ймовірність виникнення ризиків проекту

Ступінь ймовірності		R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
1	Слабо ймовірно				■						
2	Мало ймовірно	■						■	■		■
3	Ймовірно		■	■		■					
4	Доволі ймовірно						■				
5	Майже ймовірно									■	

Таблиця А.4 – Ступінь втрат ризиків проекту

Ступінь втрат		R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
1	Мінімальний										
2	Низький										
3	Середній										
4	Високий										
5	Максимальна										

На основі показників ймовірності виникнення та ступеню втрат побудуємо матрицю «Ймовірність-втрати». За основу візьмемо значення впливу ризику R, помноживши значення ймовірності виникнення (Pq) на ступінь витрати (Iq). Відповідно за добутком визначимо ступінь впливу:

– ті, що можуть ігноруватися - $1 \leq R \leq 4$;

– незначні - $5 \leq R \leq 8$;

– помірні - $9 \leq R \leq 11$;

– суттєві - $12 \leq R \leq 17$;

– критичні - $20 \leq R \leq 25$;

Значення ступеню впливу та його рівень наведені у табл. А.5.

Таблиця А.5 – Класифікація ризиків за ступенем впливу

Ризик проекту	Ступінь впливу	Класифікація рівня
Помилка у організації робіт (R1)	6 (незначний)	Виправданий
Затримки у роботі по графіку (R2)	12 (суттєвий)	Недопустимий
Низька продуктивність праці (R3)	9 (помірний)	Виправданий
Непорозуміння між сторонами (R4)	1 (може ігноруватися)	Прийнятний

Продовження таблиці А.5 – Класифікація ризиків за ступенем впливу

Ризик проекту	Ступінь впливу	Класифікація рівня
Зміна вимог до проекту (R5)	6 (незначний)	Виправданий
Непередбачувані обставини (R6)	16 (суттєвий)	Недопустимий
Збої функціоналу (R7)	6 (незначний)	Виправданий
Технічні неполадки ПК (R8)	4 (той, що може ігноруватися)	Прийнятний
Помилки в роботі додатку (R9)	25 (критичний)	Недопустимий
Збої в роботі ОС (R10)	4 (той, що може ігноруватися)	Прийнятний

Потім на основі даної таблиці побудуємо саму матрицю «Ймовірність-втрати» (табл. А.6), віднісши ризики до кольорових областей в залежності від прийнятності ризику:

– прийнятні ризики (добуток 1≤R≤4, зелений колір);

– виправданні ризики (добуток 5≤R≤11, жовтий колір);

– недопустимі ризики (добуток 12≤ R≤25, червоний колір).

Таблиця А.6 – Матриця «Ймовірність-втрати»

Максимальна(5)					R9
Висока(4)			R2	R6	
Середня (3)		R1, R7	R3		
Низька(2)		R8, R10	R5		
Мінімальна(1)	R4				
	Слабо ймовірно (1)	Мало ймовірно (2)	Ймовірно (3)	Доволі ймовірно (4)	Майже ймовірно (5)

ДОДАТОК Б ЛІСТИНГ СКРИПТІВ

Код скрипту EnemySpawner.cs:

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

public class EnemySpawner : MonoBehaviour
{
    Player player;
    public float Period;
    public GameObject Enemy;
    public bool isSpawn = true;
    public float spawnDistanse = 15;

    float currentDistance;
    float TimeUntilNextSpawn;


    // Start is called before the first frame update
    void Start()
    {
        player = FindObjectOfType<Player>();
        TimeUntilNextSpawn = Random.Range(0, Period);
    }

    // Update is called once per frame
    void Update()
    {
        TimeUntilNextSpawn -= Time.deltaTime;

        if (TimeUntilNextSpawn <= 0.0f && isSpawn) {
            TimeUntilNextSpawn = Period;
            Instantiate(Enemy, transform.position, transform.rotation);
            GetComponentInChildren<ParticleSystem>().Play();
        }
    }

    void FixedUpdate()
    {
        currentDistance = Mathf.Abs(Vector3.Distance(transform.position,
player.transform.position)); // Дистанція до гравця
        if (currentDistance <= spawnDistanse)
        {
            isSpawn = true;
        }
        else
            isSpawn = false;
    }
}
```

Код скрипту ExitBtnScript.cs:

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

public class ExitBtnScript : MonoBehaviour
```

```
{
    public void changeScene(){

        Time.timeScale = 1;
        SceneManager.LoadScene(0);
    }
}
```

Код скрипту HealthBar.cs:

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections;

public class HealthBar : MonoBehaviour
{
	Player player;
	public float maxValue;
	public Color color = Color.red;
	public int width = 4;
	public Slider slider;

	private static float current;

	void Start()
	{
		player = FindObjectOfType<Player>();
		slider.fillRect.GetComponent<Image>().color = color;
		slider.maxValue = player.maxHealth;
		maxValue = player.maxHealth;
		slider.minValue = 0;
		current = player.playerHealth;

		//UpdateUI();
	}

	public static float currentValue
	{
		get { return current; }
	}

	void Update()
	{
		if (current < 0) current = 0;
		if (current > maxValue) current = maxValue;
		slider.value = current;
	}

	void UpdateUI()
	{
		RectTransform rect = slider.GetComponent<RectTransform>();

		int rectDeltaX = Screen.width / width;
		float rectPosX = 0;

		rectPosX = rect.position.x + (rectDeltaX - rect.sizeDelta.x) / 2;
		slider.direction = Slider.Direction.LeftToRight;

		rect.sizeDelta = new Vector2(rectDeltaX, rect.sizeDelta.y);
		rect.position = new Vector3(rectPosX, rect.position.y, rect.position.z);
	}

	public static void AdjustCurrentValue(float adjust)
	{
		current += adjust;
```

```
        }
}
```

Код скрипту HealthBox.cs:

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

public class HealthBox : MonoBehaviour
{
    BoxCollider boxCollider;
    Player player;
    public AudioClip pickupSound;
    public float valueHealth;

    // Start is called before the first frame update
    void Start()
    {
        player = FindObjectOfType<Player>();
        boxCollider = GetComponent<BoxCollider>();
    }

    void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.tag == "Player")
        {
            GetComponent<AudioSource>().PlayOneShot(pickupSound, 0.5f);
            player.UpdateHealth(valueHealth);
            Debug.Log("HealthBox: pickup - OK");

            Destroy(boxCollider);
            Destroy(gameObject);
        }
    }
}
```

Код скрипту HitPlayer.cs:

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

public class HitPlayer : MonoBehaviour
{
    public float damage = 0;

    Player player;

    void Start()
    {
        player = FindObjectOfType<Player>();
    }

    void OnTriggerEnter(Collider other)
    {
        if(other.gameObject.tag == "Player"){
            Debug.Log("Damage Player: " + damage + " hp");
            player.UpdateHealth(damage);
        }
    }

}
```

Код скрипту InitializeOnLoad.cs:

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

public class InitializeOnLoad : MonoBehaviour
{

    public GameSoundsSetting gameSoundValue;
    public GameSoundtrackSetting gameSoundtrackValue;

    // Start is called before the first frame update
    void Start()
    {
        gameSoundValue.SetStartVolume();
        gameSoundtrackValue.SetStartVolume();
    }
}
```

Код скрипту Joystic_controller.cs:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.EventSystems;

public class Joystic_controller : MonoBehaviour, IPointerDownHandler, IDragHandler,
IPointerUpHandler
{
    Image joystickBg;
    Image joystick;
    Vector2 inputVector;
    float horizontalMove;
    float verticalMove;

    void Start()
    {
        joystickBg = GetComponent<Image>();
        joystick = transform.GetChild(0).GetComponent<Image>();
    }
    public void OnPointerDown(PointerEventData eventData)
    {
        OnDrag(eventData);
    }
    public void OnDrag(PointerEventData eventData)
    {
        Vector2 pos;
        if (
                RectTransformUtility.ScreenPointToLocalPointInRectangle
                (
                    joystickBg.rectTransform,
                    eventData.position,
                    eventData.pressEventCamera,
                    out pos
                )
            )
        {
            pos.x = (pos.x / joystickBg.rectTransform.sizeDelta.x);
            pos.y = (pos.y / joystickBg.rectTransform.sizeDelta.y);
        }
        var inputVector = new Vector2(pos.x * 2 - 1, pos.y * 2 - 1);
        inputVector = (inputVector.magnitude > 1.0f) ? inputVector.normalized : inputVector;
        //Debug.Log(inputVector);
```

```
        horizontalMove = (float)System.Math.Round(inputVector.x , 1);
        verticalMove = (float)System.Math.Round(inputVector.y , 1) * (-1);

        joystick.rectTransform.anchoredPosition = new Vector2(inputVector.x *
(joystickBg.rectTransform.sizeDelta.x / 2), inputVector.y * (joystickBg.rectTransform.sizeDelta.y
/ 2));
    }
    public void OnPointerUp(PointerEventData eventData)
    {
        //inputVector = Vector2.zero;
        horizontalMove = 0;
        verticalMove = 0;
        joystick.rectTransform.anchoredPosition = Vector2.zero;
    }


    public float Horizontal()
    {
        if(horizontalMove != 0)
            return horizontalMove;
        else
            return Input.GetAxisRaw("Horizontal");
    }
    public float Vertical()
    {
        if (verticalMove != 0)
            return verticalMove;
        else
            return Input.GetAxisRaw("Vertical");
    }

}
```

Код скрипту Joystick_Rotate.cs:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.EventSystems;

public class Joystick_Rotate : MonoBehaviour, IPointerDownHandler, IDragHandler,
IPointerUpHandler
{
    Image joystickBg;
    Image joystick;
    Vector2 inputVector;
    float horizontalMove;
    float verticalMove;

    void Start()
    {
        joystickBg = GetComponent<Image>();
        joystick = transform.GetChild(0).GetComponent<Image>();
    }
    public void OnPointerDown(PointerEventData eventData)
    {
        OnDrag(eventData);
    }
    public void OnDrag(PointerEventData eventData)
    {
        Vector2 pos;
        if (RectTransformUtility.ScreenPointToLocalPointInRectangle(joystickBg.rectTransform,
eventData.position, eventData.pressEventCamera, out pos))
        {
```

```
            pos.x = (pos.x / joystickBg.rectTransform.sizeDelta.x);
            pos.y = (pos.y / joystickBg.rectTransform.sizeDelta.y);
        }
        var inputVector = new Vector2(pos.x * 2 - 1, pos.y * 2 - 1);
        inputVector = (inputVector.magnitude > 1.0f) ? inputVector.normalized : inputVector;
        horizontalMove = (float)System.Math.Round(inputVector.x, 1);
        verticalMove = (float)System.Math.Round(inputVector.y, 1) * (-1);

        joystick.rectTransform.anchoredPosition = new Vector2(inputVector.x *
(joystickBg.rectTransform.sizeDelta.x / 2), inputVector.y * (joystickBg.rectTransform.sizeDelta.y
/ 2));
    }
    public void OnPointerUp(PointerEventData eventData)
    {
        //inputVector = Vector2.zero;
        horizontalMove = 0;
        verticalMove = 0;
        joystick.rectTransform.anchoredPosition = Vector2.zero;
    }


    public float Horizontal()
    {
        return horizontalMove;
    }
    public float Vertical()
    {
        return verticalMove;
    }
}
```

Код скрипту LoadResultOfGame.cs:

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


public class LoadResultOfGame : MonoBehaviour
{

    public Text L_DesertOfCrystals_bestScore;
    public Text L_DesertOfCrystals_bestTime;

    public Text L_SpaceFactory_bestScore;
    public Text L_SpaceFactory_bestTime;


    float bestTime;
    string bestScore;


    public void LoadBestResultOfGames(){

        if (PlayerPrefs.HasKey("DesertOfCrystals_bestScore")) {
            L_DesertOfCrystals_bestTime.text =
PlayerPrefs.GetFloat("DesertOfCrystals_bestTime").ToString();
            L_DesertOfCrystals_bestScore.text =
PlayerPrefs.GetString("DesertOfCrystals_bestScore");
        }
        else{
            L_DesertOfCrystals_bestTime.text = "0";
            L_DesertOfCrystals_bestScore.text = "0";
        }
```

```
        if (PlayerPrefs.HasKey("SpaceFactory_bestScore")) {
            L_SpaceFactory_bestTime.text =
PlayerPrefs.GetFloat("SpaceFactory_bestTime").ToString();
            L_SpaceFactory_bestScore.text = PlayerPrefs.GetString("SpaceFactory_bestScore");
        }
        else{
            L_SpaceFactory_bestTime.text = "0";
            L_SpaceFactory_bestScore.text = "0";
        }

    }
}
```

Код скрипту MenuControls.cs:

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

public class MenuControls : MonoBehaviour
{
    public void PlayPressed(string sceneName)
    {
        SceneManager.LoadScene(sceneName);
    }
    public void ExitPressed()
    {
        Application.Quit();
        Debug.Log("Exit pressed!");
    }
}
```

Код скрипту MovementAnimator.cs:

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

public class MovementAnimator : MonoBehaviour
{
    NavMeshAgent navMeshAgent;
    Animator animator;

    // Start is called before the first frame update
    void Start()
    {
        navMeshAgent = GetComponent<NavMeshAgent>();
        animator = GetComponentInChildren<Animator>();
    }

    // Update is called once per frame
    void Update()
    {
        animator.SetFloat("speed", navMeshAgent.velocity.magnitude);
    }
}
```

Код скрипту Player.cs:

```
using System.Collections;
```

```
using System.Collections.Generic;
using TMPro;
using UnityEngine;
using UnityEngine.AI;

[RequireComponent(typeof(Rigidbody))]

public class Player : MonoBehaviour
{

    NavMeshAgent navMeshAgent;
    Shot shot;
    public AudioClip shotSound;
    public Transform gunBarrel;
    public Joystic_controller joystickMove;
    public Joystick_Rotate joystickRotate;
    public UiPauseBtn gamePause;
    public GameObject cursor;

    public float moveSpeed;
    public float playerHealth = 100;
    public float playerDamage = 5;
    public float rotateSpeed = 1;
    public float shotPeriod;
    public float maxHealth = 100;

    float TimeUntilNextShot;
    bool isShot;

    private bool isAlive = true;
    private bool isPause = false;


    void Start()
    {
        if(playerHealth > maxHealth){
            playerHealth = maxHealth;
        }
        shot = FindObjectOfType<Shot>();
        navMeshAgent = GetComponent<NavMeshAgent>();
        navMeshAgent.updateRotation = true;
    }

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

        if(Time.timeScale > 0) {
            isPause = false;
        }
        else{
            isPause = true;
        }
        if(isPause == true)
            return;

        if(isAlive){

            Vector3 movement = new Vector3(joystickMove.Vertical(), 0.0f,
joystickMove.Horizontal());

            if(joystickMove.Vertical() == 0 || joystickMove.Horizontal() == 0){
                Vector3 dir = Vector3.zero;

                if (Input.GetKey(KeyCode.A))
```

```
                dir.z = -1.0f;
            if (Input.GetKey(KeyCode.D))
                dir.z = 1.0f;
            if (Input.GetKey(KeyCode.W))
                dir.x = -1.0f;
            if (Input.GetKey(KeyCode.S))
                dir.x = 1.0f;

            movement = new Vector3(dir.x, 0.0f, dir.z);
        }

        navMeshAgent.velocity = movement * moveSpeed * Time.deltaTime;

        // The step size is equal to speed times frame time.
        float singleStep = rotateSpeed * Time.deltaTime;
        var targetDirection = new Vector3(joystickRotate.Vertical(), 0,
joystickRotate.Horizontal());
        Vector3 newDirection = Vector3.RotateTowards(transform.forward, targetDirection,
singleStep, 0.0f);

        transform.rotation = Quaternion.LookRotation(newDirection);

        if (joystickRotate.Vertical() != 0 && joystickRotate.Horizontal() != 0)
        {
            TimeUntilNextShot -= Time.deltaTime;
            if (TimeUntilNextShot <= 0.0f || Input.GetMouseButtonDown(0))
            {
                TimeUntilNextShot = shotPeriod;

                var from = gunBarrel.position;
                var target = cursor.transform.position;
                var to = new Vector3(target.x, from.y, target.z);

                var direction = (to - from).normalized;

                RaycastHit hit;
                if (Physics.Raycast(from, to - from, out hit, 100))
                {
                    to = new Vector3(hit.point.x, from.y, hit.point.z);
                    if (hit.transform != null)
                    {
                        Debug.Log("Player: success shot");
                        var enemy = hit.transform.GetComponent<Zombie>();
                        if (enemy != null)
                            enemy.Kill(playerDamage);

                    }
                }
                else
                    to = from + direction * 100;

                shot.Show(from, to);
                GetComponent<AudioSource>().PlayOneShot(shotSound, 0.5f);
            }
        }
    }
    else{
        gamePause.OnPointerClick();
        Debug.Log("Player: GameOver, hero die");
    }
}


public void UpdateHealth(float devHealth){
    if(devHealth > 0){
```

```
            if(playerHealth + devHealth > maxHealth){
                playerHealth = maxHealth;
            }
            else{
                playerHealth += devHealth;
            }

        }
        else{
            playerHealth += devHealth;
        }
        HealthBar.AdjustCurrentValue(devHealth);
        if(playerHealth <= 0){
            isAlive = false;
        }

    }
    public float getHealh()
    {
        return playerHealth;
    }
    public float getMaxHealh()
    {
        return maxHealth;
    }

    private void playerMovement()
    {
        Vector3 movement = new Vector3(joystickMove.Vertical(), 0.0f, joystickMove.Horizontal());
        transform.Translate(movement * moveSpeed * Time.deltaTime);
        //navMeshAgent.velocity(movement * moveSpeed * Time.deltaTime);
    }

}
```

Код скрипту PlayerCamera.cs:

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


public class PlayerCamera : MonoBehaviour
{
    Player player;
    Vector3 offset;

    // Start is called before the first frame update
    void Start()
    {
        player = FindObjectOfType<Player>();
        offset = transform.position - player.transform.position;
    }

    // Update is called once per frame
    void LateUpdate()
    {
        transform.position = player.transform.position + offset;
    }
}
```

Код скрипту Shot.cs:

```
using System.Collections;
```

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

public class Shot : MonoBehaviour
{
    LineRenderer lineRenderer;
    bool visible;

    // Start is called before the first frame update
    void Start()
    {
        lineRenderer = GetComponent<LineRenderer>();
    }

    // Update is called once per frame
    void FixedUpdate()
    {
        if (visible)
            visible = false;
        else
            gameObject.SetActive(false);
    }

    public void Show(Vector3 from, Vector3 to)
    {
        lineRenderer.SetPositions(new Vector3[]{ from, to });
        visible = true;
        gameObject.SetActive(true);
    }
}
```

Код скрипту UiGameScore.cs:

```
using System.Collections;
using System.Collections.Generic;
using System.Globalization;
using UnityEngine;
using UnityEngine.UI;

public class UiGameScore : MonoBehaviour
{

    public float gameScore = 0;
    public Text gameScoreText;

    // Update is called once per frame
    public void UpdateScore(float enemyScore)
    {
        gameScore += enemyScore;
        gameScoreText.text = gameScore.ToString();
    }

    public float getGameScore(){
        return gameScore;
    }
}
```

Код скрипту UiGameScore.cs:

```
using System.Collections;
using System.Collections.Generic;
using System.Reflection;
using UnityEngine;
using UnityEngine.UI;
```

```
using UnityEngine.EventSystems;
using UnityEngine.SceneManagement;
using System.Threading;

public class UiPauseBtn : MonoBehaviour
{
    public bool isPause = false;
    public Image image;
    public GameObject gameUi;
    public GameObject pauseUi;
    public GameObject btnResumeGame;
    Player player;


    public Text timerText;
    public Text gameScoreText;

    void Start(){
        if (Time.timeScale == 0)
        {
            isPause = false;
            OnPointerClick();
        }
        else
        {
            isPause = true;
            OnPointerClick();
        }
        player = FindObjectOfType<Player>();
    }

    public void OnPointerClick()
    {
        var GameTimeObject = GameObject.Find("TimerCounter"); //шукаємо обьект на інтерфейсі
        var GameTimeScript = GameTimeObject.GetComponent<UiTimer>();

        var GameScoreObject = GameObject.Find("UiScore"); //шукаємо обьект на інтерфейсі
        if (!isPause)
        {
            Time.timeScale = 0;
            isPause = true;
            gameUi.SetActive(false);
            pauseUi.SetActive(true);
            if(player.getHealh() <= 0)
            {
                btnResumeGame.SetActive(false);
            }
            //отримуємо час гри
            timerText.text = GameTimeScript.getTimerValueStr();

            //Отримуємо значення балів
            var GameScoreScript = GameScoreObject.GetComponent<UiGameScore>();
            gameScoreText.text = GameScoreScript.getGameScore().ToString();

            string curSceneBestScore = SceneManager.GetActiveScene().name + "_bestScore";
            string curSceneBestTime = SceneManager.GetActiveScene().name + "_bestTime";
            Debug.Log("texts");
            if(PlayerPrefs.HasKey(curSceneBestScore)){
                if(float.Parse(PlayerPrefs.GetString(curSceneBestScore)) <=
GameScoreScript.getGameScore()){
                    if(float.Parse(PlayerPrefs.GetString(curSceneBestScore)) ==
GameScoreScript.getGameScore()){
                        if(PlayerPrefs.GetFloat(curSceneBestTime) <
GameTimeScript.getTimerValue()){
```

```
                            PlayerPrefs.SetFloat(curSceneBestTime,
GameTimeScript.getTimerValue());
                            PlayerPrefs.SetString(curSceneBestScore, gameScoreText.text);
                            PlayerPrefs.Save();
                            Debug.Log("UiPauseBtn: Progress saved (Score: " + gameScoreText.text
+ " / Time: " + GameTimeScript.getTimerValue() + ")");
                        }
                    }
                    if(float.Parse(PlayerPrefs.GetString(curSceneBestScore)) <
GameScoreScript.getGameScore()){
                        PlayerPrefs.SetFloat(curSceneBestTime,  GameTimeScript.getTimerValue());
                        PlayerPrefs.SetString(curSceneBestScore, gameScoreText.text);
                        PlayerPrefs.Save();
                        Debug.Log("UiPauseBtn: Progress saved (Score: " + gameScoreText.text + "
/ Time: " + GameTimeScript.getTimerValue() + ")");
                    }
                }
            }
            else{
                PlayerPrefs.SetFloat(curSceneBestTime,  GameTimeScript.getTimerValue());
                PlayerPrefs.SetString(curSceneBestScore, gameScoreText.text);
                PlayerPrefs.Save();
                Debug.Log("UiPauseBtn: Progress saved (Score: " + gameScoreText.text + " / Time:
" + GameTimeScript.getTimerValue() + ")");
            }
        }
        else
        {
            Time.timeScale = 1;
            isPause = false;
            gameUi.SetActive(true);
            pauseUi.SetActive(false);
        }
    }
    public void setPause(bool pause)
    {
        isPause = pause;
    }
}
```

Код скрипту UiTimer.cs:

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Globalization;
using UnityEngine;
using UnityEngine.UI;

public class UiTimer : MonoBehaviour
{
    public float startTime = 0;
    public float timerValue = 0;
    public Text timerText;
    private float timerValueStr = 0;

    private bool isStart = true;

    // Update is called once per frame
    void Update()
    {
        if (isStart)
        {
            timerValue += 1 * Time.deltaTime;
```

```
            timerValueStr = (float)Math.Round(timerValue, 2);
            timerText.text = timerValueStr.ToString();
        }
    }

    public float getTimerValue(){
        return (float)Math.Round(timerValue, 2);
    }

    public string getTimerValueStr(){
        return timerValueStr.ToString();
    }
}
```

Код скрипту Zombie.cs:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;
using UnityEngine.Animations;

public class Zombie : MonoBehaviour
{
    NavMeshAgent navMeshAgent;
    Player player;
    CapsuleCollider capsuleCollider;
    Animator animator;
    MovementAnimator movementAnimator;

    public float attackDistance = 2;
    public float attackSpeed = 1;
    public float enemyScore = 0;
    public float enemyHealth = 20;
    public float dropChance = 0;
    public GameObject weapon;
    public GameObject dropItems;

    public AudioClip takeDamageSound;
    public AudioClip attackTargetSound;
    public AudioClip dieSound;

    private float currentDistance;
    private BoxCollider weaponBox;
    bool attackSoundPlay = true;
    bool dead;
    bool attack;
    float TimeUntilNextAttack;


    // Start is called before the first frame update
    void Start()
    {
        navMeshAgent = GetComponent<NavMeshAgent>();
        navMeshAgent.updateRotation = false;
        player = FindObjectOfType<Player>();
        capsuleCollider = GetComponent<CapsuleCollider>();
        animator = GetComponentInChildren<Animator>();
        movementAnimator = GetComponent<MovementAnimator>();
        weaponBox = weapon.GetComponent<BoxCollider>();

    }

    // Update is called once per frame
```

```
    void Update()
    {
        if (dead)
            return;

        currentDistance = Mathf.Abs(Vector3.Distance(navMeshAgent.transform.position ,
player.transform.position)); // Дистанція до гравця

        if(currentDistance <= attackDistance){

            navMeshAgent.SetDestination(navMeshAgent.transform.position);

            Vector3 forward = player.transform.position - transform.position;
            transform.rotation = Quaternion.LookRotation(new Vector3(forward.x, 0, forward.z));

            TimeUntilNextAttack -= Time.deltaTime;
            if (TimeUntilNextAttack <= 0.0f)
            {
                TimeUntilNextAttack = attackSpeed;
                animator.SetTrigger("attack");
            }

            if (animator.GetCurrentAnimatorStateInfo(0).IsName("attack")){ //Якщо зараз анімація
атаки,
                if (attackSoundPlay) {
                    GetComponent<AudioSource>().PlayOneShot(attackTargetSound, 0.3F);
                    attackSoundPlay = false;
                }
                weaponBox.enabled = true; //то вмикаємо колізію зброї
            }
            else{
                weaponBox.enabled = false; // та вимикаємо колізію
                attackSoundPlay = true;
            }
        }
        else{

            navMeshAgent.SetDestination(player.transform.position);
            transform.rotation = Quaternion.LookRotation(navMeshAgent.velocity.normalized);
        }

    }

    public void Kill(float damage)
    {
        if (!dead) {
            enemyHealth -= damage; // наносимо урон противнику
            GetComponent<AudioSource>().PlayOneShot(takeDamageSound, 0.3F);

            if (enemyHealth <= 0){ //якщо health менше 0, тоді
                var GameScoreObject = GameObject.Find("UiScore"); //шукаємо обьект на інтерфейсі
для додавання рахунку
                var GameScoreScript = GameScoreObject.GetComponent<UiGameScore>();
                GameScoreScript.UpdateScore(enemyScore);

                dead = true;
                GetComponent<AudioSource>().PlayOneShot(dieSound, 0.3F);
                dropItem();
                Destroy(capsuleCollider);
                Destroy(movementAnimator);
                Destroy(navMeshAgent);
                animator.SetTrigger("died");
                GetComponentInChildren<ParticleSystem>().Play();
                Destroy(gameObject, 3);
            }
```

```
            else{
                GetComponentInChildren<ParticleSystem>().Play();
            }
        }
    }
    public void dropItem(){

        if(Random.Range( 0.0f, 1.0f ) <= dropChance){
             Instantiate(dropItems, transform.position, transform.rotation);
        }
    }
}
```