ДВНЗ «Донецький національний технічний університет»

Факультет комп'ютерно-інформаційних технологій та автоматизації
(повне найменування інституту, назва факультету)

Кафедра прикладної математики та інформатики
(повна назва кафедри)

«До захисту допущено»
В.о. завідувача кафедри ПМІ

_____ Наталія МАСЛОВА
(підпис) (Ім'я та ПРІЗВИЩЕ)

"___"_____20__ р.

Випускна кваліфікаційна робота

бакалавра
(освітній ступінь)

на тему: «Відтворення звукових ефектів в пакетах візуального програмування» зі спецчастиною: «Розробка аудіоплейеру для розширення можливостей контенту в Unity-3D»

Виконав: студент 4 курсу, групи ІПЗ-19
(шифр групи)

спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)

освітньої програми 121 Інженерія програмного забезпечення

Головко Микита Андрійович _____
(прізвище та ініціали) (підпис)

Керівник в.о. зав. каф. ПМІ, к.т.н., доц. Наталя МАСЛОВА _____
(посада, науковий ступінь, вчене звання, Ім'я та ПРІЗВИЩЕ) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, Ім'я та ПРІЗВИЩЕ) (підпис)

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

Студент _____
(підпис)

Луцьк – 2023р.

ДВНЗ «Донецький національний технічний університет»
Факультет комп'ютерно-інформаційних технологій та автоматизації
Кафедра прикладної математики та інформатики
Освітній ступень бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(шифр і назва)
Освітня програма 121 Інженерія програмного забезпечення
(шифр і назва)

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Наталія МАСЛОВА
(підпис) (Ім'я та ПРІЗВИЩЕ)
«____»_____2023 року

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Головку Микиті Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи: «Відтворення звукових ефектів в пакетах візуального програмування» зі спецчастиною: «Розробка аудіоплейеру для розширення можливостей контенту в Unity-3D»
керівник роботи Наталія МАСЛОВА, к.т.н., доц., в.о. зав. каф. ПМІ,
(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання, посада)
затверджені наказом від "5" травня 2023 року №175
2. Строк подання студентом проєкту (роботи) 08.06.2023р.
3. Вихідні дані до проєкту (роботи) результати преддипломної практики
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібнорозробити)
   1) аналіз предметної області і постановка завдання;
   2) опис застосованих інструментів;
   3) опис та обґрунтування роботи аудіоплейеру;
   4) створення 3D моделей;
   5) розробка інтерфейсу;
   6) розробка ігрового додатку;
   7) тестування роботи аудіоплейеру в ігровому додатку.
5. Перелік графічного матеріалу (з точним зазначенням обов'язковихкреслень)
Робота містить 60 сторінок основного тексту, 57 рисунків, кількості яких достатньо для відображення сутності та основних положень роботи

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Назарова І.А. доцент каф. ПМІ		8.06.2023

7. Дата видачі завдання 5 травня 2023 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1.	Об'єктний аналіз предметної області	05.05.2023 – 08.05.2023	
2.	Створення матеріалів для ігрового додатку	09.05.2023 – 12.05.2023	
3.	Збір інформації та методичних відомостей, щодо посилань в Unity-3D	13.05.2023 – 15.05.2023	
4.	Розробка аудіоплейеру	16.05.2023 – 18.05.2023	
5.	Створення ігрового додатку	19.05.2023 – 24.05.2023	
6.	Тестування аудіоплейеру в ігровому додатку	25.05.2023 – 26.05.2023	
7.	Оформлення пояснювальної записки	27.05.2023 – 04.06.2023	
8.	Нормоконтроль пояснювальної записки, її перевірка антиплагіатною системою	05.06.2023 – 08.06.2023	
9.	Підготовка слайдів презентації	09.06.2023 – 21.06.2023	
10.	Захист роботи	22.06.2023	

Студент _____ Микита ГОЛОВКО
(підпис) (Ім'я та ПРІЗВИЩЕ)

Керівник роботи _____ Наталія МАСЛОВА
(підпис) (Ім'я та ПРІЗВИЩЕ)

А Н О Т А Ц І Я

Головко Микита Андрійович, Відтворення звукових ефектів в пакетах візуального програмування / Випускна кваліфікаційна робота на здобуття освітнього ступеня «бакалавр» за спеціальністю 121 «Інженерія програмного забезпечення». – ДВНЗ ДонНТУ, Луцьк, 2023.

Об’єктом дослідження - технології відтворення звукових ефектів в пакетах візуального програмування.

Предметом дослідження – методи створення та додавання аудіоплеєру до проєкту комп’ютерної гри, контенту якої розроблено в Unity-3D

Метою роботи є розширення можливостей ігрового контенту в Unity-3D з застосуванням скриптів та створення додаткових 3D-матеріалів.

Практичне значення полягає у створенні аудіоплеєра для ігор в Unity, який дозволяє користувачам відтворювати власні композиції. Цей функціонал розширює можливості гри, завдяки аудіоплеєру користувачі можуть використовувати власну музику під час гри, що додає індивідуальності та настрою до ігрового досвіду

Методи дослідження, що застосовані при розробці телеграм-боту: аналіз коду, тестування та відлагодження, профілювання та рефакторинг. Реалізація аудіоплеєру здійснюється в розробленому скрипті в середовищі розробки Unity.

Ключові слова: аудіоплеєр, Unity, C#, PlayMaker, MagicaVoxel, PixilArt, візуальне програмування, ігри.

ЗМІСТ

ВСТУП

З із зростанням популярності відеоігор з початку 2000-х років і збільшенням складності розробки, виникла потреба у спрощенні цього процесу. Завдяки появі ігрових двигунів та інструментів для роботи з 3D-об’єктами, розробники змогли полегшити створення складних ігрових світів з великою кількістю деталей. Використання інструментів, таких як програма SpeedTree, дозволило швидко і якісно створювати 3D-моделі дерев для ігрових проектів.

Найпопулярніші ігрові двигуни, зокрема Unreal Engine і Unity-3D, надають розробникам можливість вибирати між програмуванням на мові C++/C# та візуальним програмуванням.

Об’єктом даної кваліфікаційної роботи є технології відтворення звукових ефектів в пакетах візуального програмування.

Предметом дослідження виступають методи створення та додавання аудіоплеєру до проєкту комп’ютерної гри, контенту якої розроблено в Unity-3D.

Метою даної роботи є розширення можливостей ігрового контенту в Unity-3D з застосуванням скриптів та створенням додаткових 3D-матеріалів.

Для досягнення поставленої мети намічено рішення наступних задач:

1) аналіз методів візуального програмування;
2) пошук прикладів застосування посилань в Unity-3D;
3) розробка ігрового додатку;
4) розробка скрипту для відтворення композицій;
5) тестування розробленого скрипту.

Основним методом дослідження є експериментальний підхід, що передбачає проведення практичних експериментів, тестування різних методів та алгоритмів в реальних умовах.

Практичне значення результатів полягає в створенні аудіоплеєру, який дозволяє користувачам включати та насолоджуватися власними композиціями.

Кваліфікаційна робота складається з 4 розділів, включає в себе 56 рисунків, 20 джерел інформації, 65 загалом сторінок.

1 ОГЛЯД ПАКЕТІВ ВІЗУАЛЬНОГО ПРОГРАМУВАННЯ

1.1 Особливості створення контенту та механік, скінчені автомати

Ігровий контент та механіки є ключовими елементами будь-якої відеогри. Ігровий контент можна поділити на такі елементи, як візуальний стиль, звукові ефекти, історія, персонажі, рівні та інші компоненти, які роблять гру більш цікавою для гравців. Механіки визначають правила та основи гри, її цілі механізми, що керують діями гравців.

Для ефективного створення ігрових механік та контенту, можна використовувати математичні моделі, зокрема скінчені автомати. Скінчені автомати – це моделі, що використовуються для представлення різноманітних послідовностей дій або станів. Вони широко використовуються в галузі програмування ігор, де їх використовують для моделювання поведінки героїв, ворогів та інших об’єктів гри.

Під час створення гри потрібно враховувати і її аудиторію, наприклад, дитячі ігри, за правилом, мають бути більш кольоровими та яскравими, з простими та нескладними механіками, для більш зрілої аудиторії – більш реалістичними за візуальним стилем, складними та цікавими механіками.

Скінчені автомати – це математична модель, яка дозволяє представити поведінку об’єктів зі скінченною кількістю станів та переходів між ними. Обране доповнення з візуальним програмуванням для Unity-3D використовує метод скінчених автоматів, на рисунку 1.1 представлено середовище візуального програмування – PlayMaker для Unity 3D.

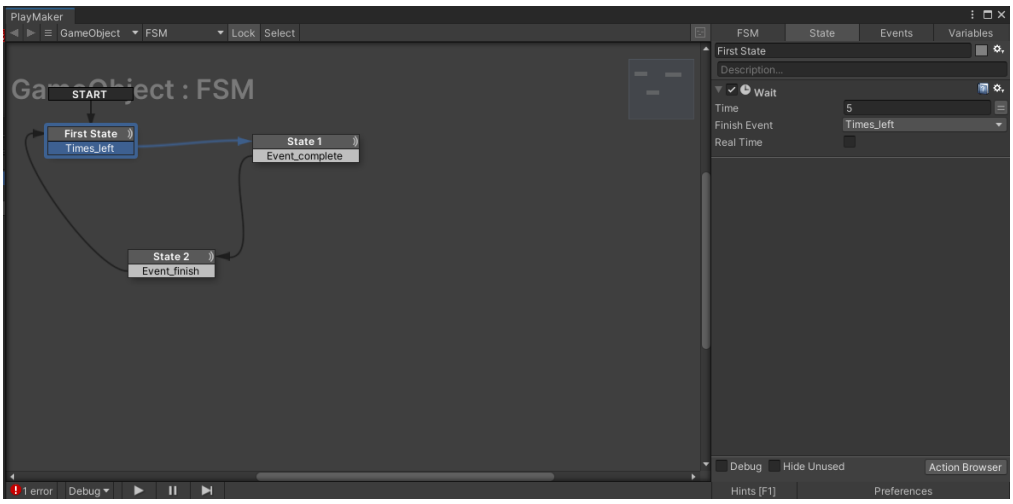

Рисунок 1.1 – Середовище візуального програмування

1.2 Огляд пакетів візуального програмування, їх переваги та недоліки

Одним із найпопулярнішим пакетом візуального програмування є PlayMaker. PlayMaker – популярний інструмент для автоматизації розробки ігрових механік та контенту в Unity, він дозволяє створювати складні механіки та поведінку гравців та інших об’єктів в грі, використовуючи графічний інтерфейс. До одного об’єкта можна додати декілька компонентів FSM, в кожному з них розробити декілька станів гравця, наприклад переміщення, стрільбу, падіння та зв’язати їх, щоб створити поведінку гравця в цілому.

Його перевагою є те, що він дозволяє швидко створювати ігровий контент та механіки, що забезпечує швидку та ефективну розробку гри. Однак можна виділити недолік, цей пакет може бути складним для новачків, які не знають, як використовувати скінчені автомати, також, в цьому пакеті відсутнє посилання на об’єкти. Оскільки цей пакет використовує скінченні автомати, то кожен створений FSM-скрипт буде оновлюватися кожен кадр, тобто при великій кількості таких скриптів може поганим чином сказатись на кількість кадрів прорисовки.

Окрім PlayMaker, існує пакет візуального програмування під назвою Bolt.

Bolt надає можливість створювати складку функціональність гри за допомогою вузлів та зв’язків між ними, інтерфейс цього пакету зображено на рисунку 1.2.

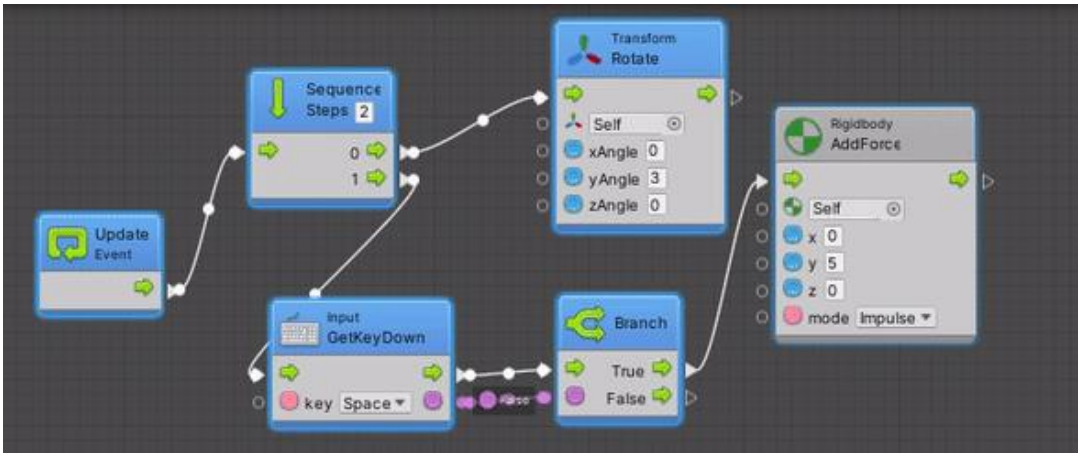

Рисунок 1.2. – Інтерфейс середовища візуального програмування Bolt

Зовнішній вигляд та функціонал цього пакету дуже схожий на середовище візуального програмування в Unreal Engine.

Його перевагою є те, що він дозволяє розробникам більш гнучно працювати зі складною функціональністю гри, забезпечуючи швидку та ефективну розробку. Недоліком Bolt є те, що він може бути дорогим для деяких розробників, а також те, що для роботи з цим пакетом, потрібно хоч на середньому рівні знати мову C# та клас MonoBehaviour.

У наступному пункті буде поверхнево розглянуто ігровий двигун Unreal Engine. В Unreal Engine є вибір способу програмування – C++ або візуальне програмування.

Пакет візуального програмування в Unreal Engine має назву Blueprints. Інтерфейс Blueprints зображено на рисунку 1.3.

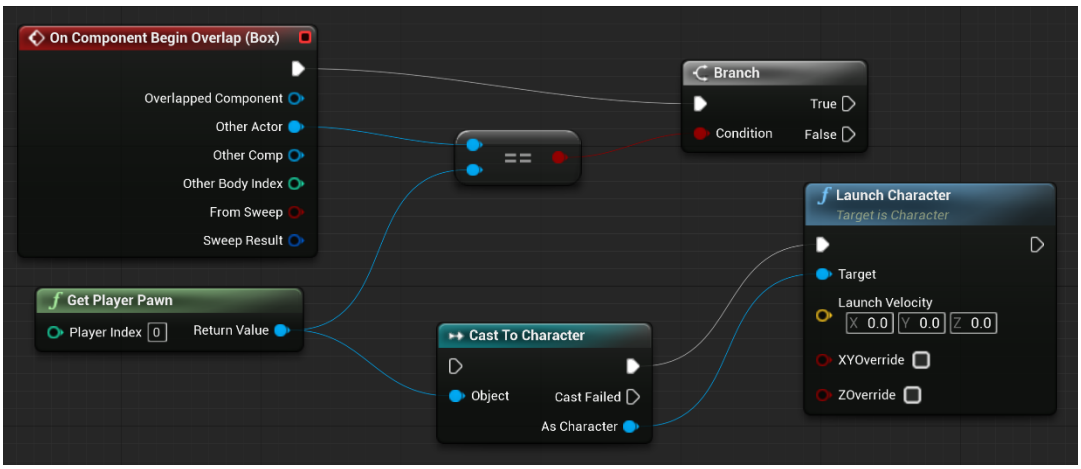

Рисунок 1.3. – Інтерфейс та середовище візуального програмування Blueprints

Blueprints дозволяє розробникам створювати складку функціональність гри за допомогою блоків, що з'єднанні зв'язками.

Перевагою Blueprints є те, що він є частиною Unreal Engine, що дозволяє розробникам з легкістю інтегрувати його в свою гру. Недоліком Blueprints є те, що він може бути складним для новачків, також, якщо для розробки гри на Unreal Engine було обрано Blueprint як спосіб програмування, то в цьому проєкті, можливості створити скрипт на мові C++ буде неможливим.

1.3 Опис та порівняння Unreal Engine та Unity

Unreal Engine та Unity є двома найбільш популярними ігровими двигунами. Обидва двигуни мають свої переваги та недоліки, але вони дозволяють розробникам створювати ігри на різних платформах.

Unreal Engine – є потужним та розширюваним інструментом для розробки гри. Він має більші можливості для реалістичної графіки, що робить його найбільш популярнім для створення AAA-ігор. Unreal Engine п'ятої версії має такі технології, як Lumen - система глобального освітлення в реальному часі, та Nanite – технологія масштабування геометрії. До недоліків цього двигуну, можна піднести не дуже доброзичливий до користувача інтерфейс, а також не зовсім зручні інструменти про створенні ігрового поля, наприклад, для нанесення текстури на поле, потрібно створити відокремлений шейдер для

поля, в який треба занести дані та зображення текстури, які будуть використані для деталізації ландшафту. Інтерфейс двигуну Unreal Engine зображено на рисунку 1.4.

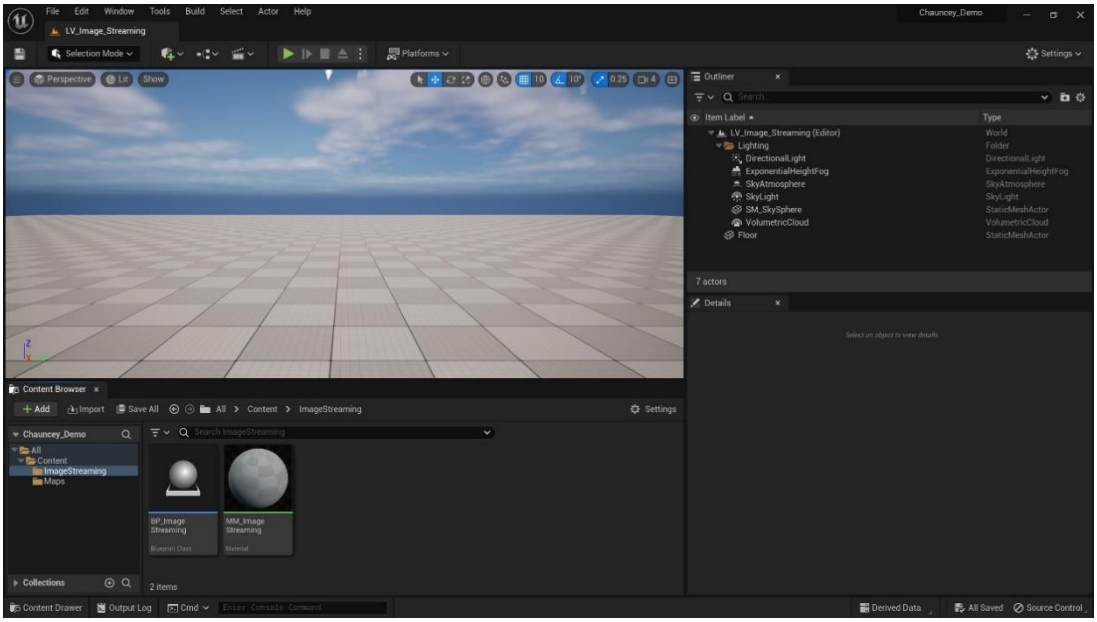

Рисунок 1.4 – Інтерфейс Unreal Engine

Unreal Engine має велику базу користувачів, але при цьому, він має невеликий вибір доповнень в маркеті, та скудний список відеоуроків. Маркет з доповненнями зображено на рисунку 1.5.

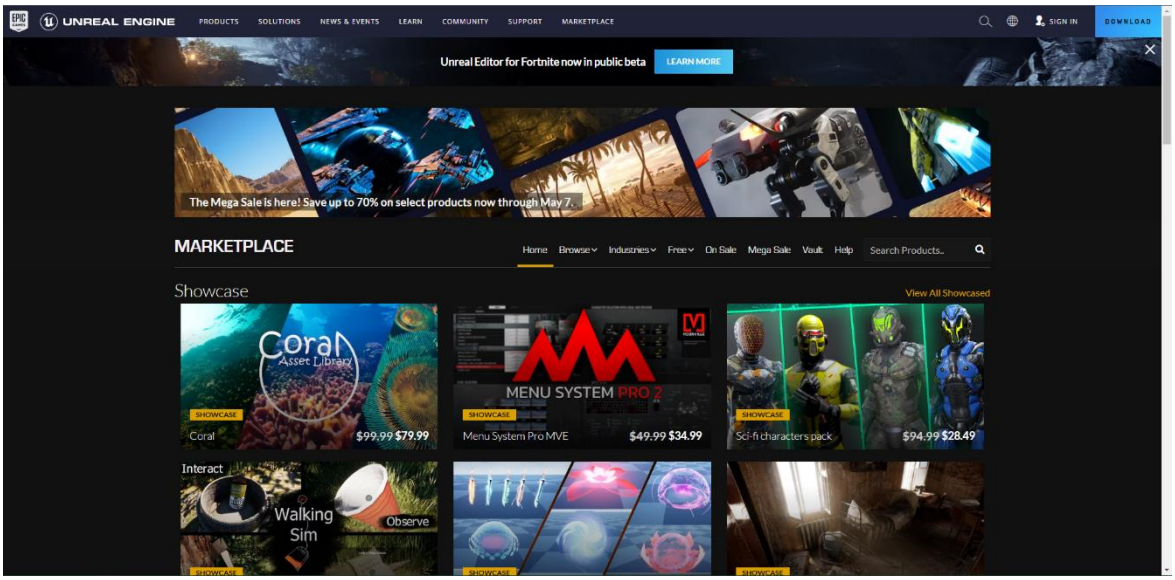

Рисунок 1.5 – Маркет з доповненнями для Unreal Engine

Unity являється більш легким та доступним для розробки гри. Редактор в Unity має простий інтерфейс, що робить його більш популярним для початківців та середніх розробників. Пакети візуального програмування в Unity – це доповнення, це дозволяє використовувати в одному проєкті візуальне програмування, та скрипти на мові C#. Unity більш підходить для створення простих та невеликих відеоігор. Існує пакет Unity HDRP, котрий додає до двигуну покращене освітлення і більший вибір пост-обробки картинки, але навіть з таким додатком, гра, яка була створена на Unreal Engine буде мати більш красиву картинку. Інтерфейс двигуну Unity зображено на рисунку 1.6.

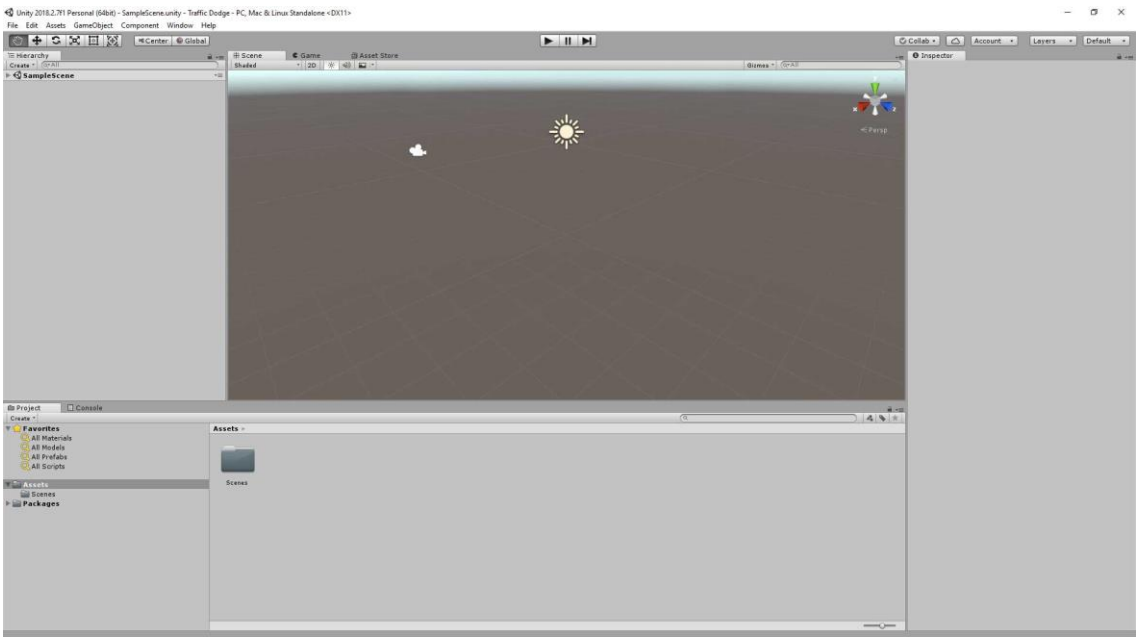

Рисунок 1.6 – Інтерфейс двигуну Unity

Для Unity також існує свій маркет доповнень, за кількістю доповнень Unity виграє у Unreal. Для новачків є дуже велика кількість відеоуроків. До недоліків двигуну Unity, можна віднести високу вартість для комерційних проєктів та низьку продуктивність при створені більш великих проєктів. Маркет доповнень зображено на рисунку 1.7.

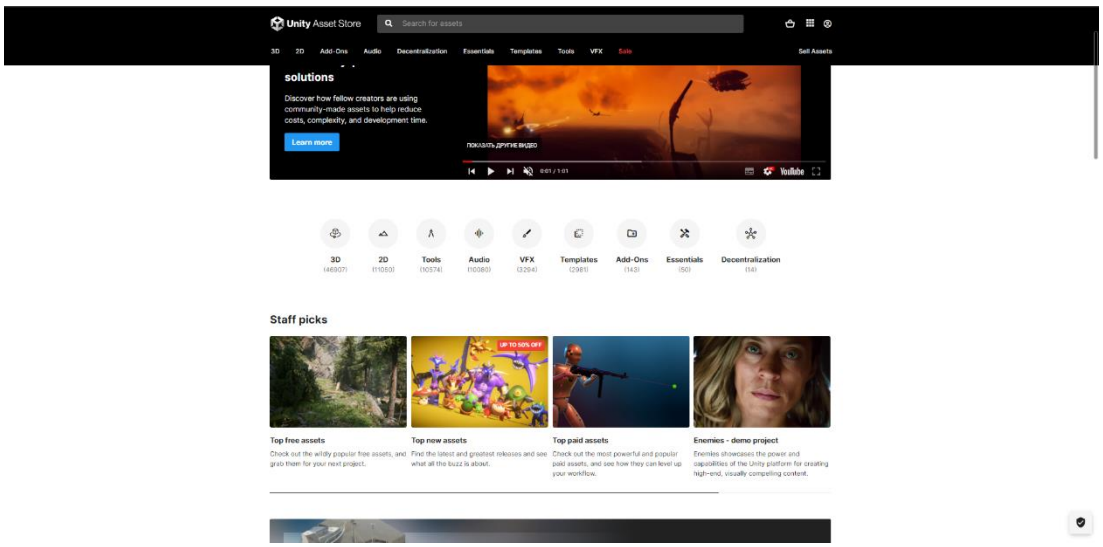

Рисунок 1.7 – Маркет доповнень для Unity

1.4 Відсутність функціоналу в PlayMaker

Незважаючи на те, що PlayMaker є потужним інструментом для візуального програмування, він має деякі недоліки у порівнянні з програмуванням на мові скриптів, написаних на C#.

Один з найбільших недоліків PlayMaker – обмежений функціонал у порівнянні з написанням скриптів на мові програмування C#. PlayMaker пропонує обмежений набір дій та станів, які можна використовувати для розробки ігор.

Окрім цього, в PlayMaker також відсутня підтримка складних структур даних, таких як масиви або словники. Також, PlayMaker не підтримує пакети зі сторонніх джерел, що може призвести до обмеження в використанні деяких інструментів та ресурсів.

Хоча PlayMaker дозволяє швидко створювати прототипи та прискорює процес розробки, він не дозволяє більш детально налаштовувати поведінку програми настільки, наскільки це дозволяє програма, написана на C#.

До й так невеликого набору дій та станів у PlayMaker, неможливо додати свої. Ця проблема лише вирішується написанням окремих скриптів на мові C#.

Незважаючи на це, PlayMaker можна використовувати для описання поведінки деяких об’єктів, які не повинні мати гнучкий функціонал.

1.5 Висновки за розділом

У даному розділі було розглянуто особливості створення ігрового контенту та механік з використанням скінченних автоматів. Було описано візуальне програмування PlayMaker, Bolt та Blueprints їх переваги та недоліки. Також було описано і порівняно два найпопулярніші ігрові двигуни – Unreal Engine та Unity.

Не дивлячись на обмеження Playmaker, він залишається корисним інструментом для розробки ігор, зокрема, для швидкої розробки прототипів та внесення змін у готовій проєкт, однак для більшої гнучкості та функціональності, розробники можуть використовувати програми на мові скриптів C#.

2 ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Огляд MagicaVoxel

MagicaVoxel – це безкоштовний графічний редактор для створення воксельних моделей. Цей редактор дозволяє швидко та легко створити 3D-модель з кубів. MagicaVoxel має інтуїтивний інтерфейс та включає в себе багато різноманітних функцій.

Основні можливості MagicaVoxel:

- можливість створення складних 3D -моделей з блоків;
- детальне налаштування освітлення, текстур та інших параметрів для перегляду об’єкта;
- можливість імпорту та експорту моделей у різних форматах;
- функція анімації, яка дозволяє створювати прості рухи для моделі;
- інтеграція з Unity та іншими двигунами гри;
- простий інтерфейс, який дозволяє швидко створювати моделі без додаткового навчання.

MagicaVoxel відрізняється від інших програм для редагування воксельної графіки тим, що вона проста та інтуїтивно зрозуміла у використанні. Її інтерфейс мінімалістичний, що дозволяє користувачам швидко навчитися роботі з нею. Крім того, MagicaVoxel підтримує різні платформи, такі як Windows, Linux та MacOS, що робить її універсальним рішенням для різних користувачів.

Не дивлячись на простоту воксельної графіки, яка складається лише з фігур квадрату, можна добитись дуже красивого вигляду сцен, об’єктів, приклади створених сцен за допомогою цього інструменту зображено на рисунках 2.1, 2.2.

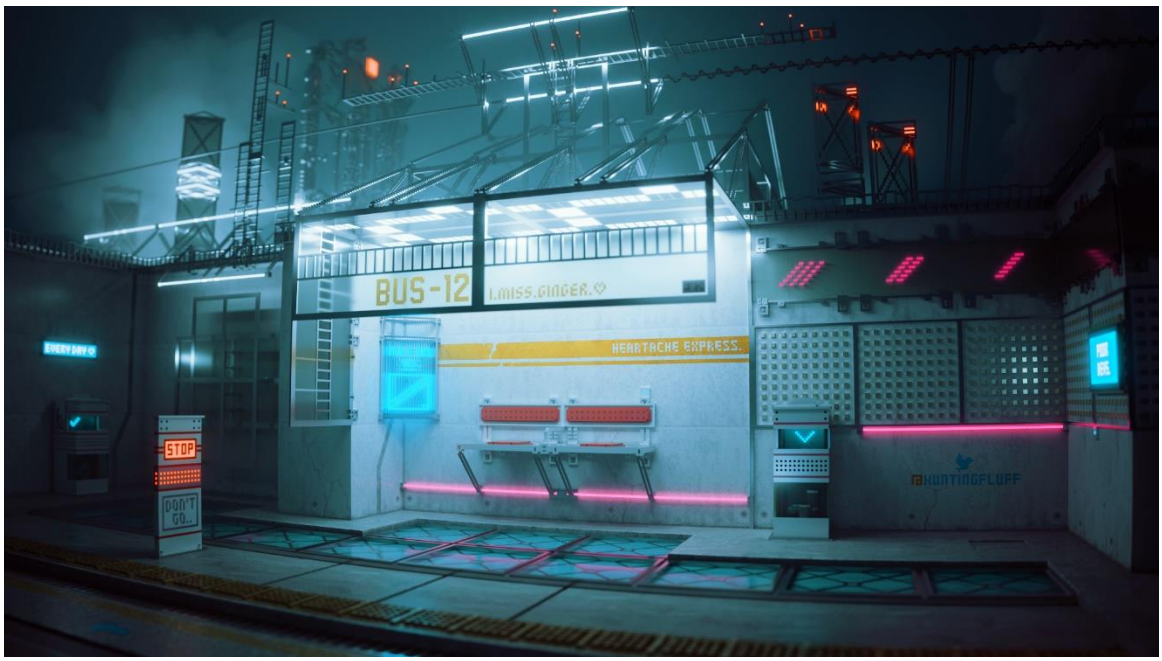

Рисунок 2.1 – перший приклад сцени створеної в програмі MagicaVoxel

Рисунок 2.2 – другий приклад створеної сцени в програмі MagicaVoxel

Окрім будування моделей, до MagicaVoxel можливо додати шейдери від користувачів, за допомогою котрих, з’являється можливість генерувати деякі об’єкти, від простого дерева до цілого міста.

2.2 Створення 3D-моделей за допомогою Voxel

Для розробки ігор у 3D-просторі необхідно створювати різноманітні об’єкти, які будуть використовуватися для створення оточення гри та

взаємодії з гравцем. Воксельні моделі у порівняні з моделей, які створенні з полігонів – дуже прості у конструюванні.

Основна різниця між моделями, створеними з полігонів і вокселей, полягає в їх структурі і способі створення. Моделі з полігонів складаються з поверхонь, створених з тисячі трикутників, в той час, як воксельні моделі складаються з тривимірних пікселів, що забезпечує більш грубу деталізацію та створення більш складних форм.

Одна з важливих відмінностей полягає в тому, як моделі оброблюються і редагуються. Для полігонів зазвичай використовують програми для моделювання, такі як Blender, який буде використано в наступному розділі, Maya або 3ds Max, ці програми дозволяють детально редагувати окремі полігони і їх текстури, створювати складні форми та обробляти освітлення.

З іншого боку, для створення воксельних моделей, використовують такі програми, як MagicaVoxel, VoxelShop або Qubicle. Вони дозволяють відобразити всі воксельні моделі як тривимірну сітку в якій можна додавати і видаляти вокселі, встановлювати колір, а також застосовувати різноманітні ефекти, такі як туманність або градієнти.

Для створення воксельної моделі краще використовувати інструмент – MagicaVoxel, інтерфейс даної програми буде зображено на рисунку 2.3.

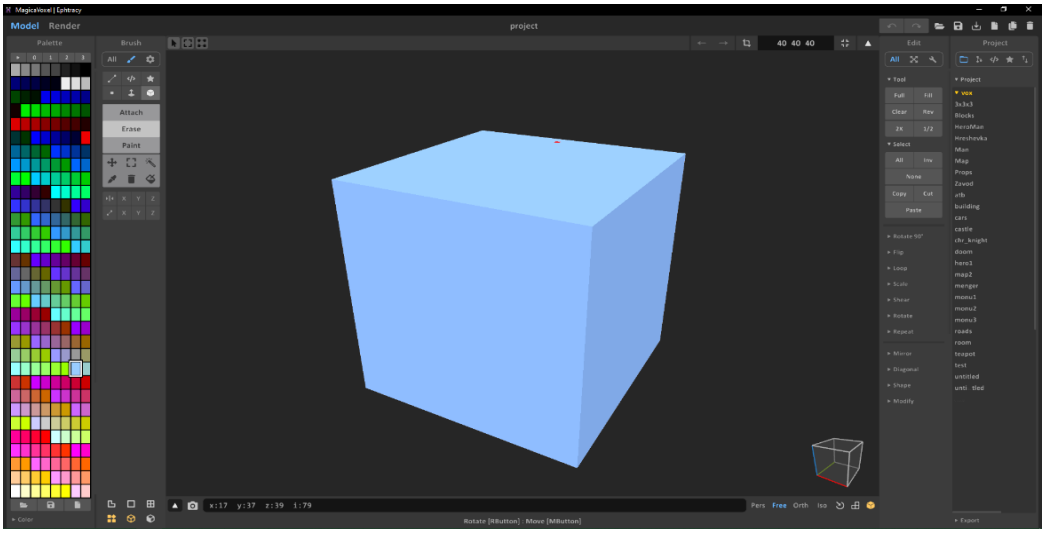

Рисунок 2.3 – інтерфейс MagicaVoxel

До переваг MagicaVoxel можна віднести:

- простий та інтуїтивно зрозумілий інтерфейс;
- можливість експорту об’єкта у формати .OBJ, .FBX, .VOX та інші;
- широкий вибір інструментів та можливість розширення можливостей шляхом додавання користувацьких шейдерів.

До недоліків MagicaVoxel можна віднести:

- обмеження на розмір моделей та їх складність;
- відсутність створення складних анімацій;
- генерація неправильної сітки полігонів.

Отже, хоча MagicaVoxel є потужним інструментом для створення візуально привабливого ігрового контенту, його використання може бути обмеженим у випадках створення моделей більшої складності та реалістичності.

2.3 Оптимізація Voxel 3D-моделі

Головною проблемою MagicaVoxel – це отримана сітка полігонів при експорті файлу в формати .OBJ та .FBX.

Для вирішення цієї проблеми потрібно використати програму VoxelShop. VoxelShop – це інший редактор для воксельної графіки, але на відмінну від MagicaVoxel, VoxelShop має більш непривабливий інтерфейс, і не дуже зручне керування камерою. Але незважаючи на це, VoxelShop правильно створює сітку полігонів на об’єктах.

Для оптимізації об’єкту із MagicaVoxel потрібно експортувати об’єкт у формат VOX, зразок тестового об’єкту зображено на рисунку 2.4.

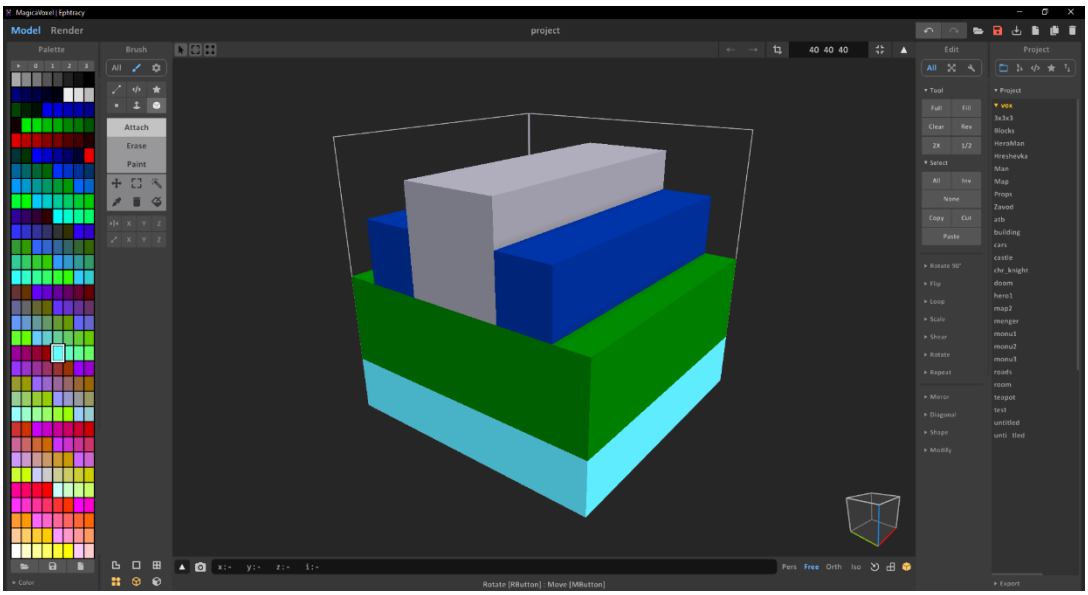

Рисунок 2.4 – зразок створеного об’єкту в MagicaVoxel

Експортуємо файл у формат .VOX (для тестування також було проведено експорт в формат .OBJ), та імпортуємо в програму VoxelShop і нічого не редагуючи одразу відкриваємо експорт, параметри експорту ті інтерфейс VoxelShop зображено на рисунку 2.5.

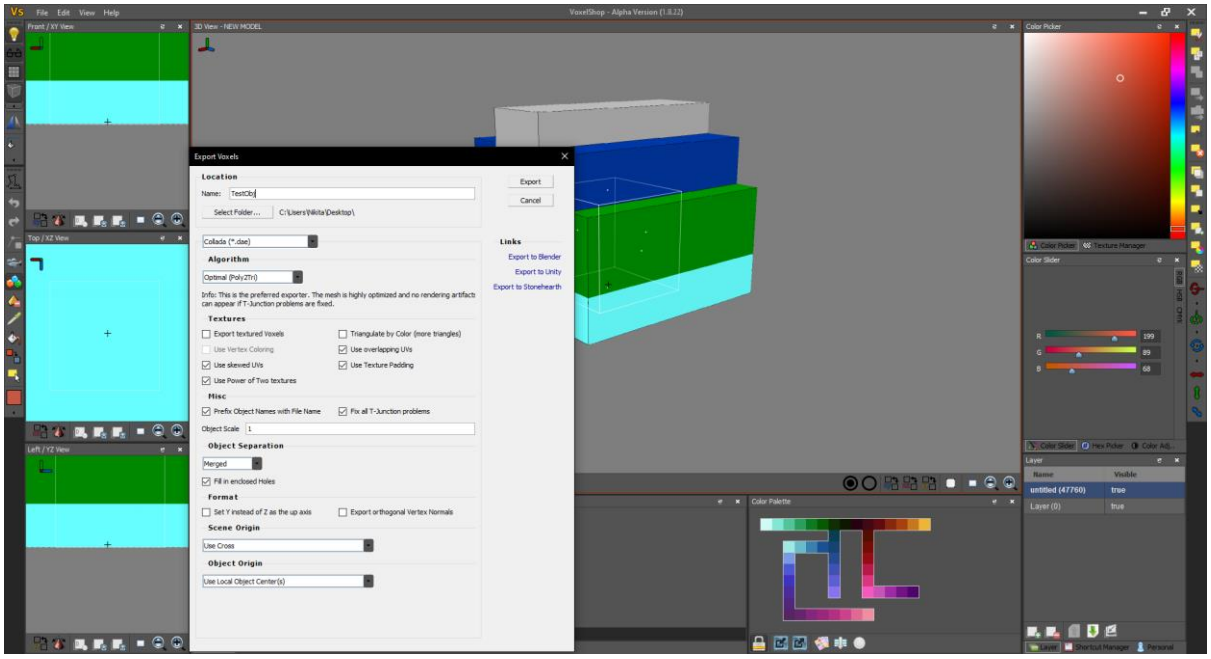

Рисунок 2.5 – інтерфейс VoxelShop та параметри експорту

Експортований файл має формат .DAE, для використання моделі в ігровому двигуні, потрібно переформувати цей формат до .OBJ або .FBX, для цього потрібно додати цей файл до програми Blender.

Blender – це безкоштовний, відкритий до спільноти програмний продукт, який призначений для моделювання, анімації, рендерингу моделей. Він має великий функціонал для моделювання, текстурування, анімації, фізики, скульптування та багато іншого.

В програмі Blender необхідно імпортувати створений об’єкт, та для показу, було додано експортований об’єкт в форматі .OBJ із MagicaVoxel, два об’єкти та інтерфейс програми Blender зображено на рисунку 2.6.

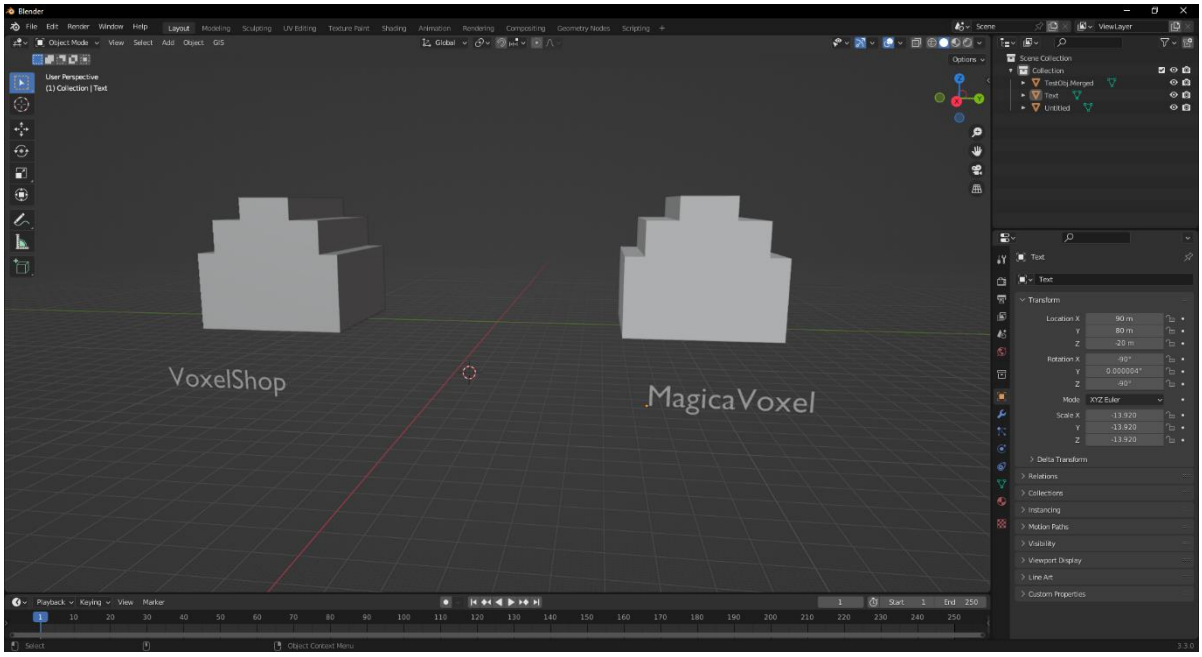

Рисунок 2.6. – експортовані об’єкти та інтерфейс програми Blender

Відкривши перегляд полігонів, можна помітити, що на об’єкті, котрий був експортований в програмі VoxelShop має меншу кількість полігонів, різницю зображено на рисунку 2.7.

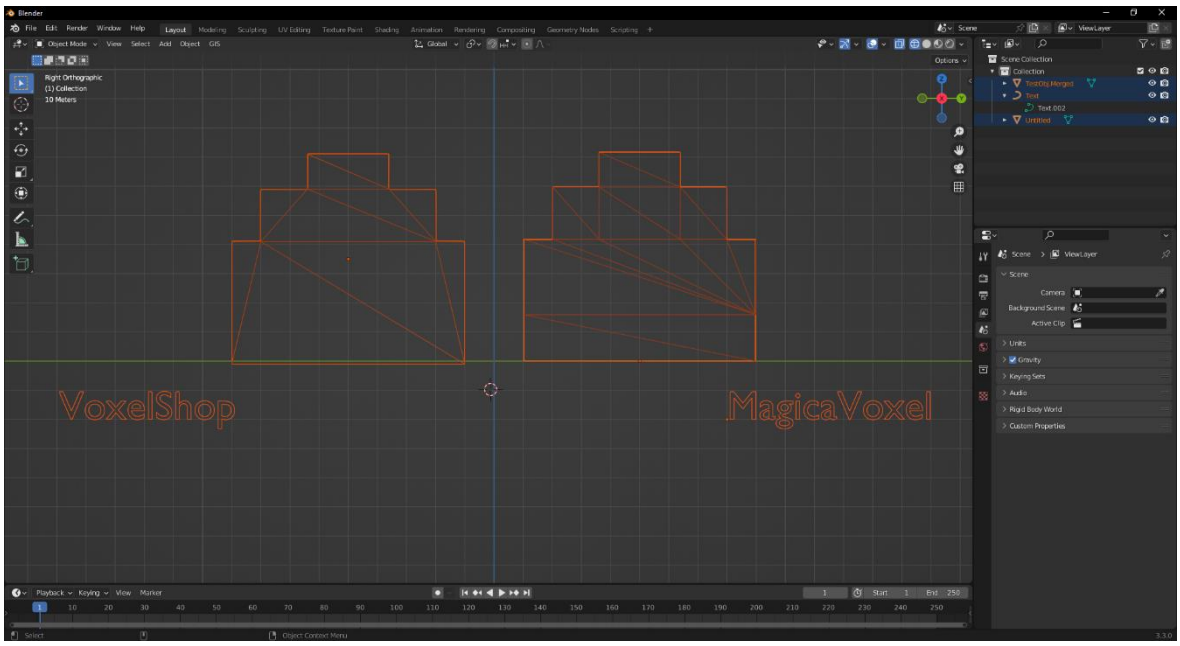

Рисунок 2.7 – різниця між оптимізованим та звичайним об’єктом

2.4 Можливості скриптової мови C#

У розробці відеоігор важливою складовою є програмування. Однією із найпоширеніших мов програмування для розробки ігор є мова C#.

Ця мова має декілька важливих переваг, такі як:

- мова C# відноситься до групи мов програмування зі строгою типізацією;
- велика кількість бібліотек та фреймворків, що спрощують розробку ігор;
- C# являється мовою з високим рівнем абстракції, що дозволяє розробникам швидко і ефективно реалізовувати складні функції.

Однак, як і в будь-якій мові програмування, є й деякі недоліки, наприклад, для розробки ігор на C# потрібно мати досить високий рівень знань програмування. Крім того, на деяких платформах C# може виконуватись повільніше, ніж мови програмування з низькорівневим доступом до апаратного забезпечення.

Мова C# є основною мовою програмування для розробки ігор в Unity. Вона дозволяє створювати складні функції та взаємодію об’єктів в грі. Для використання C# в Unity потрібно мати середні знання програмування.

Можна виділити такі можливості мови програмування C# у розробці ігор на Unity:

- створення складних логічних розрахунків для керування поведінкою героїв, ворогів та інших об’єктів в грі;
- робота зі звуком та графікою, тобто створення ефектів, анімації та рухів об’єкта;
- взаємодія зі сторонніми системами, створення мультиплеєрних ігор, робота з базами даних та API,
- використання бібліотек та фреймворків для розробки ігор на Unity.

Unity має вбудовану інтеграцію з C#, що дозволяє швидко створювати скрипти та додавати їх до об’єктів в грів. Окрім цього, Unity має клас MonoBehaviour. MonoBehaviour – це клас в Unity, який надає можливості для управління поведінкою об’єктів в грі. Цей клас дозволяє реалізувати взаємодію з користувачем, зміну позицій об’єктів, роботу зі звуками, анімаціями та багато іншого. Для створення таких скриптів, необхідно наслідувати свій скрипт від MonoBehaviour.

2.5 Створення ігрового плейеру

Велика кількість відеоігор з відкритим світом, як правило, мають своє радіо. Музика – один із головних компонентів відеогри, вона може задавати атмосферу локації, або позначити якусь подію. Але на прикладі простої гри у відкритому світі, можна створити радіо, заповнення котрого будуть займатись гравці.

Пакет візуального програмування PlayMaker, має лише одну функцію посилання, і це посилання на текстуру для об’єкта, тому для створення плейеру, потрібно використати мову програмування C#.

Для реалізації плейеру, скрипт повинен мати шлях до каталогу, в який користувачі завантажують медіа файли формату .mp3, скрипт повинен проаналізувати цей каталог, та усі файли формату .mp3 завантажити до

масиву. Керування буде реалізовано з використанням кнопок, котрі будуть додані в Unity через редактор UI.

Для створення аудіо композиції та ефектів в іграх, можна скористатись різноманітними програмами для створення музики, такими як FL Studio, Ableton, Logic, та інші. При аналізі усіх програм, для створення аудіо контенту, було обрано FL Studio.

FL Studio – це цифрова аудіо робоча станція з інтуїтивним інтерфейсом та потужними функціями для створення музики та обробки звуків. Програма містить широкий набір інструментів та ефектів, що дозволяють створювати різноманітні звукові композиції, включаючи електронну музику, рок, індастріал та інші. Інтерфейс програми зображено на рисунку 2.8.

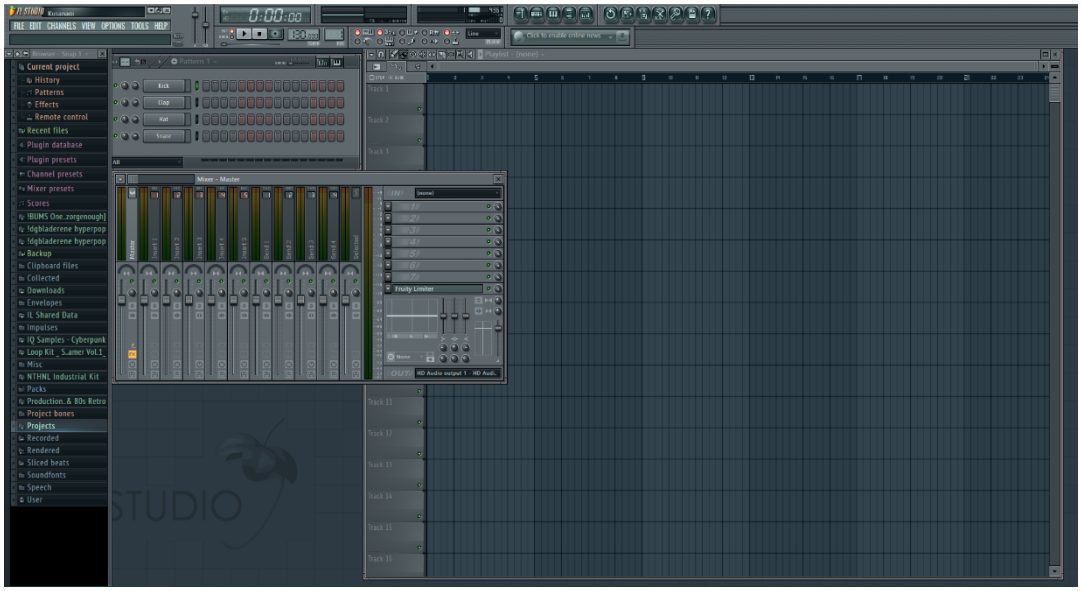

Рисунок 2.8 – інтерфейс програми FL Studio

Інтерфейс для плейеру, можна намалювати в онлайн редакторі Pixilart. Pixilart – онлайн редактор піксельного мистецтва, який дозволяє користувачам створювати графіку у стилі піксельного малюнку.

Готовий малюнок експортуємо в формат .PNG, і далі його можна використати в UI компонентах Unity.

2.6 Висновки за розділом

Було розглянуто інструмент для створення 3D моделей MagicaVoxel, проведено оптимізацію 3D моделі за допомогою таких програм як VoxelShop та Blender. Досліджено можливості скриптової мови C# та її використання в Unity. Теоретично досліджено можливість створення ігрового плейера.

Розділ містить огляд інструментів для створення 3D моделі, 2D малюнку та створення аудіоплеєру, що є основною метою представленої роботи.

3 РОЗРОБКА ПРОТОТИПУ ДЛЯ ВІДТВОРЕННЯ ЗВУКОВИХ ЕФЕКТІВ

3.1 Опис прототипу

Створений прототип гри не повинен мати великої кількості характеристик для персонажів, або багатьох механік. Суть гри проста, у користувача є персонаж, котрим потрібно переміщуватись між завданнями, котрі відображались як бій. Зображення ігрового процесу першого прототипу зображено на рисунках 3.1 та 3.2.

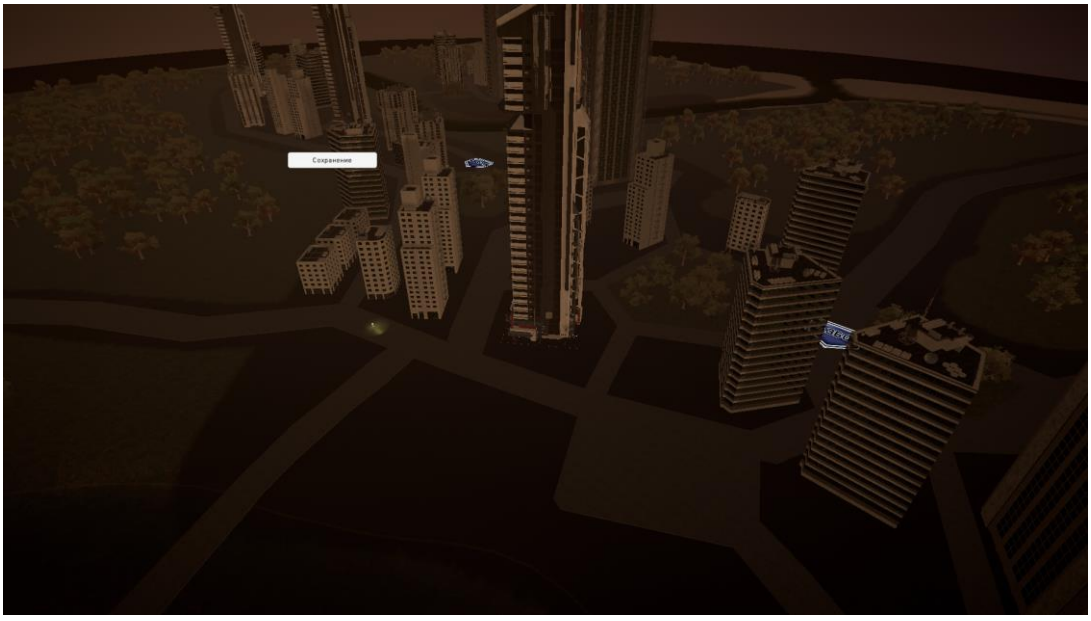

Рисунок 3.1 – Переміщення в прототипі ігрового додатку

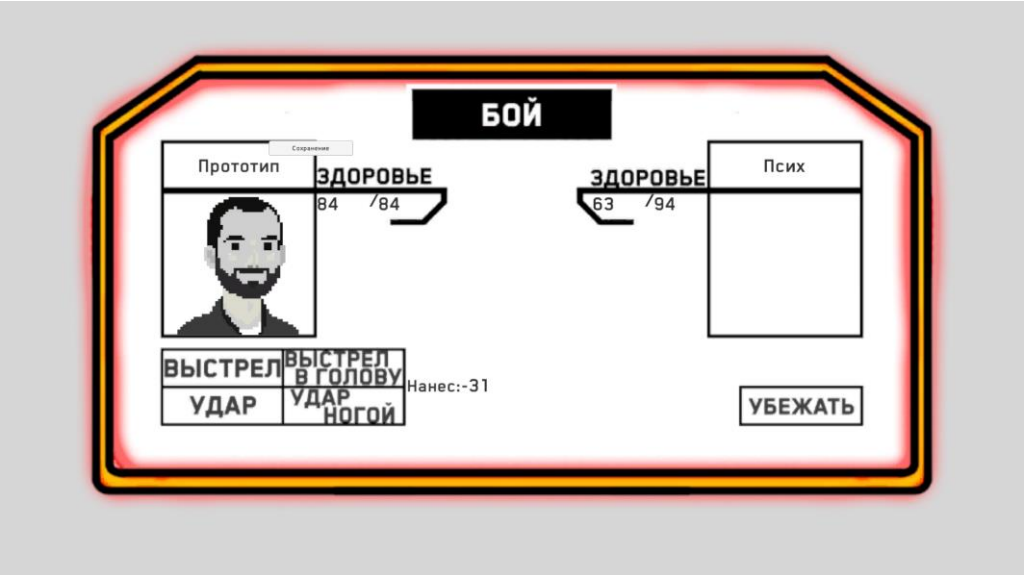

Рисунок 3.2 – зображення бою в прототипі ігрового додатку

Карта гри була заповнена за допомогою інструменту створення більш детальних об’єктів, та будівель з маркету доповнень Unity.

З всього цього прийшла ідея створення гри, де бої пояснювалися існуючим віртуальним всесвітом. Характеристики персонажу довелось зменшити за причиною нестачі досвіду у створенні балансу в вибраному жанрі відеоігор.

Жанр розробленого прототипу являється покрокова стратегія з простими механіками бою.

3.2 Ігрові моделі

Одним із ключових елементів гри, є моделі різних об’єктів, персонажів, транспорту, тощо. Для створення таких об’єктів було обрано воксельний редактор MagicaVoxel.

Воксельні моделі є альтернативою полігональним моделям і мають ряд переваг. Одна з основних переваг воксельних моделей – простота створення. Вони вимагають меншої кількості точок, порівняно з полігональними моделями, що полегшує їхнє створення та редагування.

Одним із важливих елементів ігрового світу є дороги. Для полегшення роботи з встановленням дороги в Unity, було обрано максимальний розмір для

кожного елементу, а також форма самої дороги. Створенні елементи дороги зображено на рисунку 3.3.

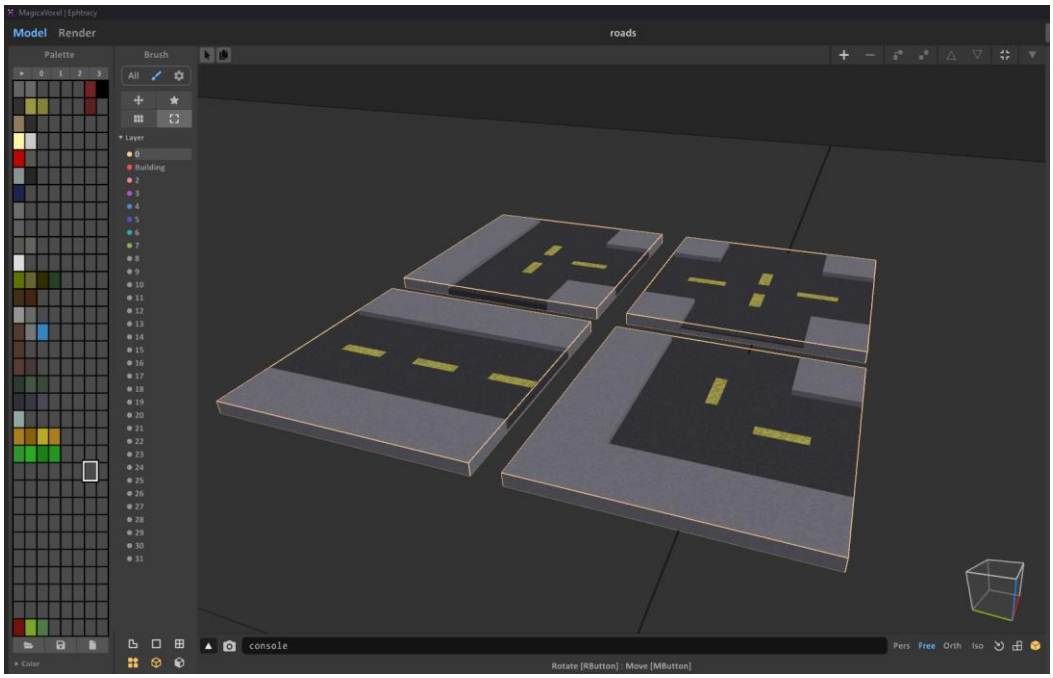

Рисунок 3.3 – різні елементи дороги для заповнення ігрового світу

Для заповнення пустих ділянок ігрового світу, було створено два об’єкти, які мають розміри дороги, що спрощує додавання їх до ігрового світу. Перший об’єкт – ділянка сірого кольору, другий об’єкт – ділянка з імітацією трави. Ці плоскі об’єкти зображено на рисунку 3.4.

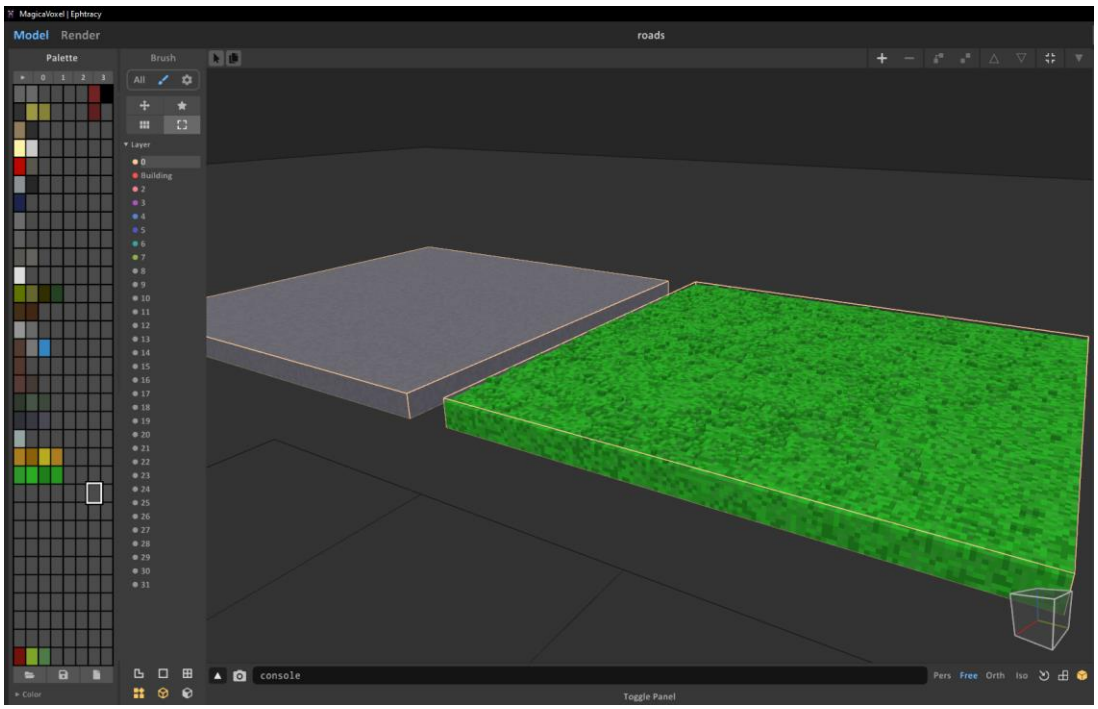

Рисунок 3.4 – плоскі елементи для заповнення ігрового світу

До важливих об’ктів ігрового світу можна додати будівлі. За допомогою воксельного редактору було створено три версії однієї будівлі, а також один доповнений об’єкт. Будівлі для заповнення ігрового світу зображено на рисунку 3.5.

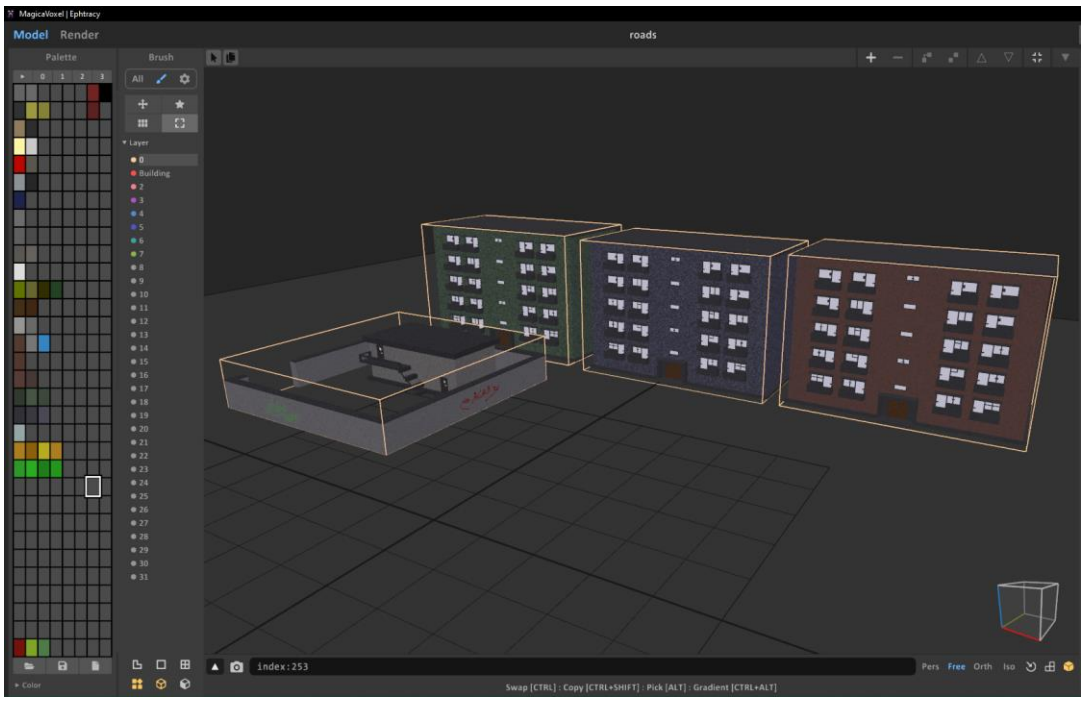

Рисунок 3.5 – створені будівлі для заповнення ігрового світу

Для заповнення світу також було створено декілька маленьких об’єктів, завдяки цьому, світ гри становиться менш пустим. Різноманітні об’єкти для ігрового світу зображено на рисунку 3.6.

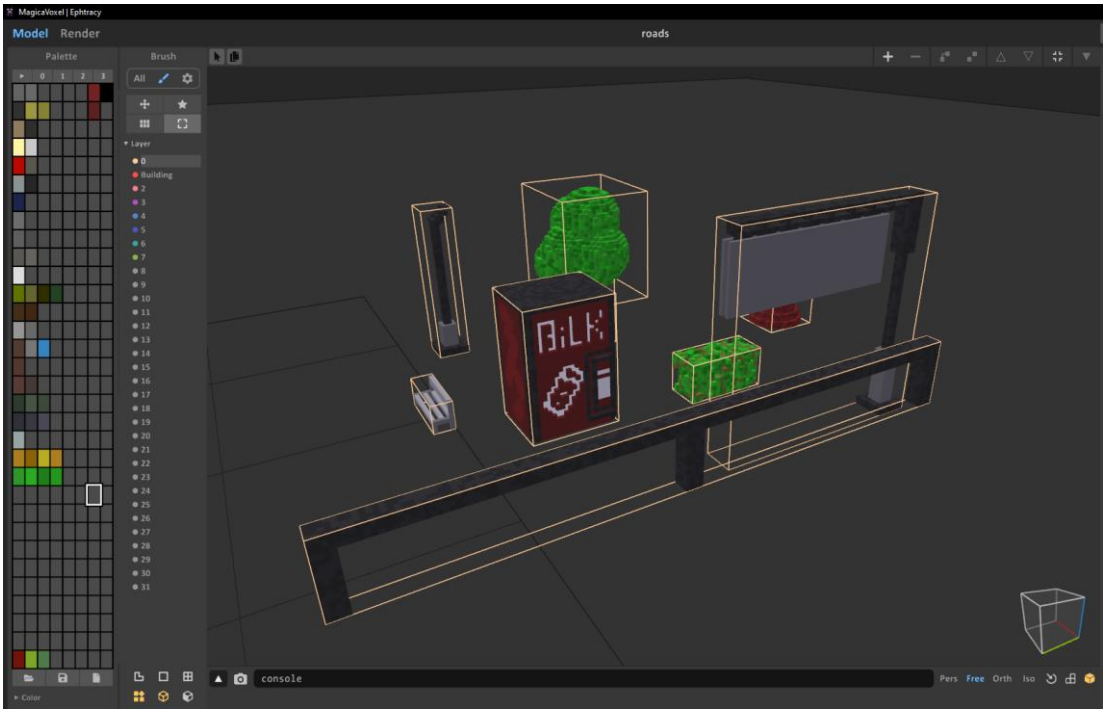

Рисунок 3.6 – невеликі об’єкти для заповнення ігрового світу

Для надання нашій грі більшої живості, було враховано наявність рухомих об’єктів, таких як люди та транспорт. Додавання цих елементів створить враження динамічного і активного ігрового світу.

Завдяки воксельному стилю гри, моделі людей можуть бути простими, тож було створено три моделі, перша модель – модель користувача, для другої та третьої моделі було використано різні кольори, щоб додати різноманіття до персонажів, котрі будуть переміщуватись по ігровому світу. Дані моделі зображено на рисунку 3.7.

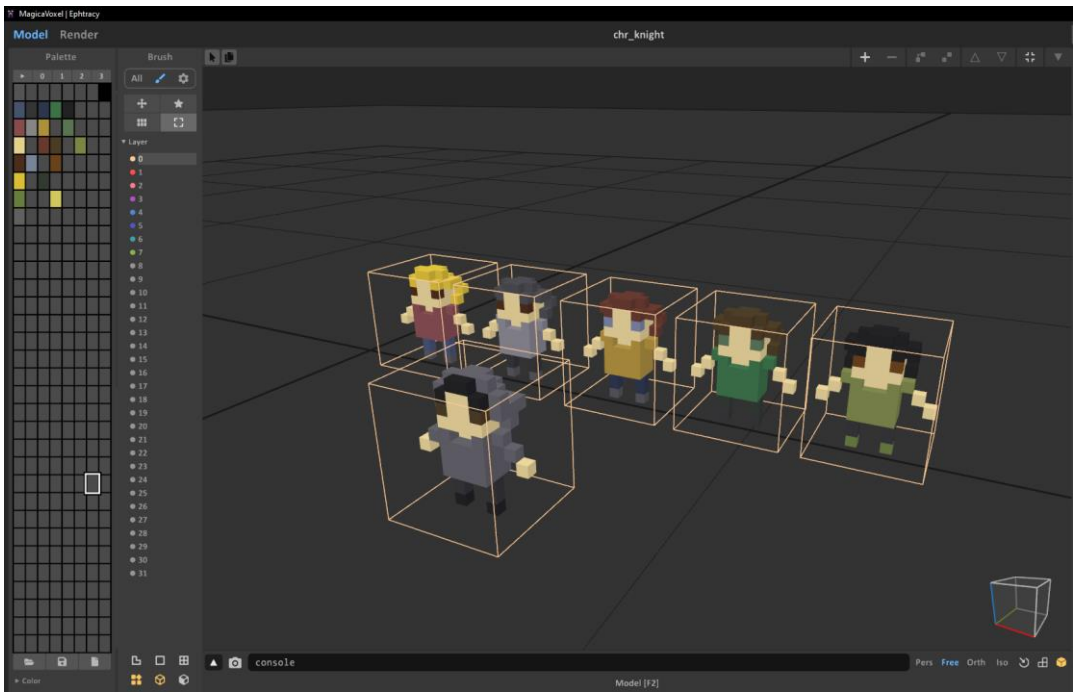

Рисунок 3.7 – моделі людей для ігрового додатку

Окрім цього, була створені об’єкти автомобілів різних кольорів однієї моделі. Ці моделі зображено на рисунку 3.8.

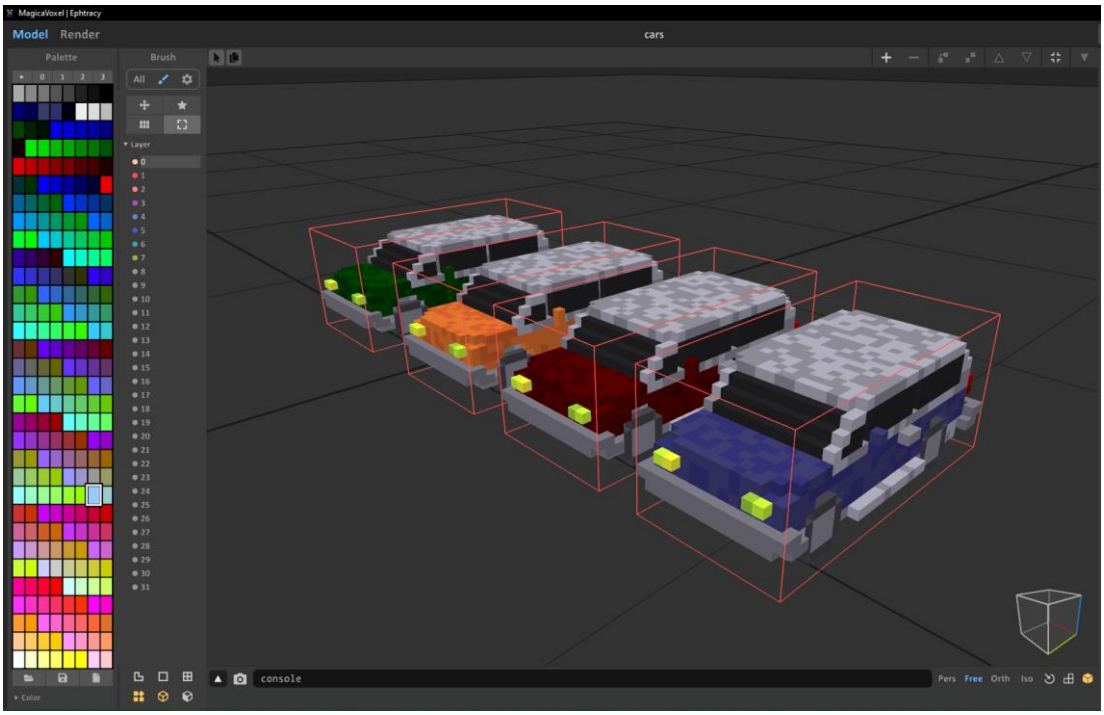

Рисунок 3.8 – моделі автомобілів для ігрового додатку

Для додавання цих моделей до ігрового двигуну, спочатку необхідно їх оптимізувати для зменшення навантаження до фізичних ресурсів комп’ютера. В самому Unity необхідно до кожного об’єкта додати коллайдер, щоб персонаж гравця, та інші рухомі об’єкти гри не випадали з ігрового світу, а також не проходили через стіни. Щоб додати коллайдер, потрібно додати сам об’єкт до простору сцени, далі у меню об’єкта додати компонент “Mesh collider”, та у цьому компоненті увімкнути “Convex”, для додавання сітки. Результат зображено на рисунку 3.9.

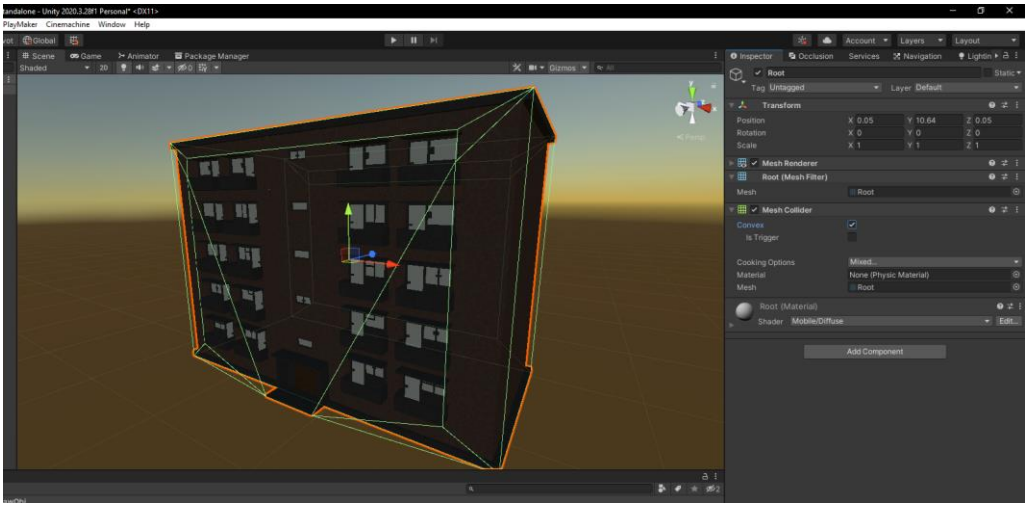

Рисунок 3.9 – додавання сітки коллайдер до об’єкту

Це треба повторити до кожного об’єкта, і далі перенести їх до каталогу, щоб створити “Prefab” цього об’єкту для подальшого використання при створенні ігрового світу.

3.3 Інтерфейс гри

Інтерфейс гри є одним з найважливіших аспектів, який впливає на взаємодію гравця з ігровим світом. Він включає в себе елементи, що допомагають користувачу керувати персонажем, виконувати завдання, тощо.

Основною метою інтерфейсу гри є забезпечення зручності та доступності для гравця. Це досягається шляхом використання різних

елементів, таких як меню, інформаційні панелі, кнопки, та інші візуальні елементи.

Для створення інтерфейсу, було обрано онлайн інструмент Pixilart. Інтерфейс у піксельному стилі буде привабливо дивитись, при воксельній реалізації світу гри. Було намальовано такі елементи, як аватар персонажів, плейер, меню гри, також для додавання різних панелей з інформацією герою, та бою, була намальована портативна приставка.

Для створення інтерфейсу в Unity, необхідно обрати у меню “GameObject”, “UI”, такий об’єкт, як “Canvas”, що зображено на рисунку 3.10, це буде слугувати полотном для всього інтерфейсу.

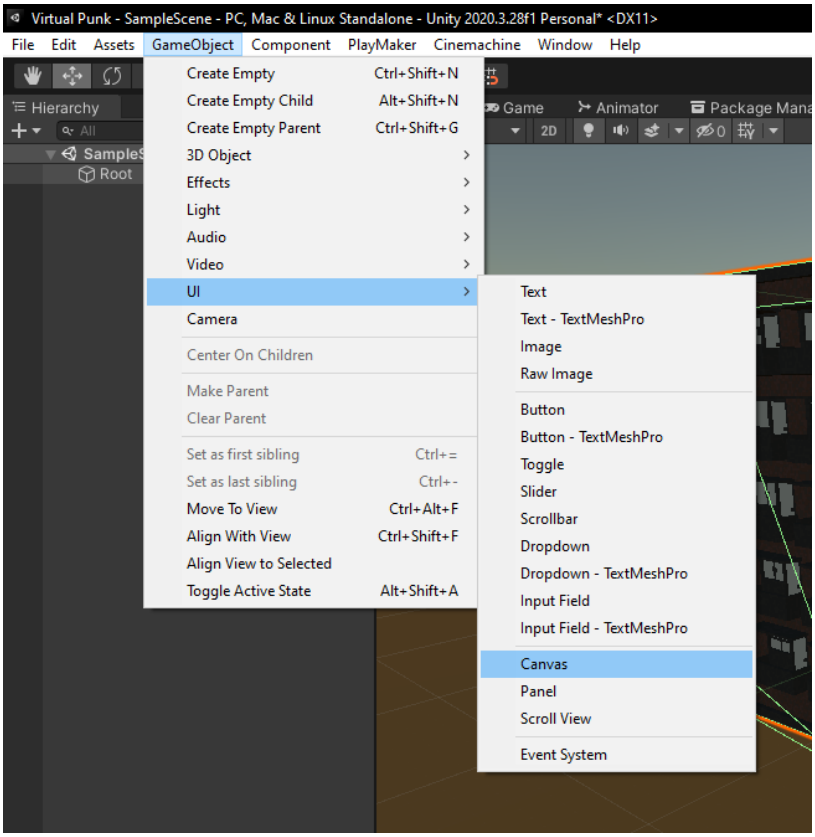

Рисунок 3.10 – додавання полотна для інтерфейсу

Далі додаємо до цього полотна наші створенні елементи інтерфейсу, а також кнопки керування. Результат зображено на рисунку 3.11.

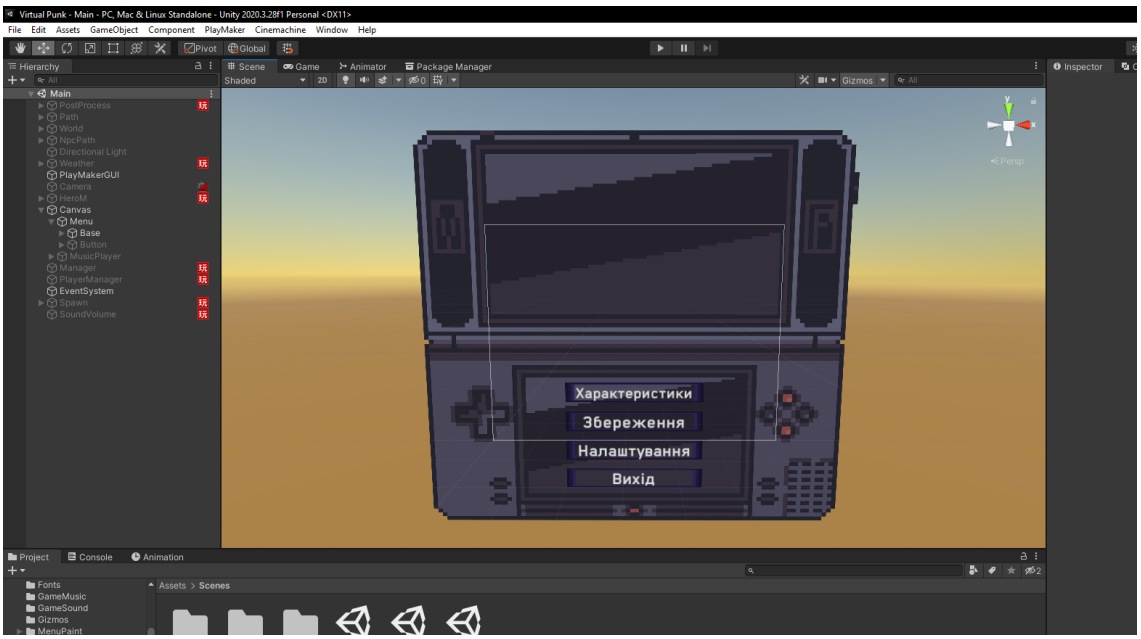

Рисунок 3.11 – ігрове меню

До нього додаємо елементи інтерфейсу, до яких будемо отримувати доступ за допомогою встановлених кнопок. Вся частина скриптів була зроблена за допомогою візуального програмування Playmaker, для додавання скрипту Playmaker для подальшого його редагування, потрібно додати до об’єкту компонент “Playmaker FSM”. Для переміщення по меню за допомогою кнопок, було створено скрипт, який зображено на рисунку 3.12.

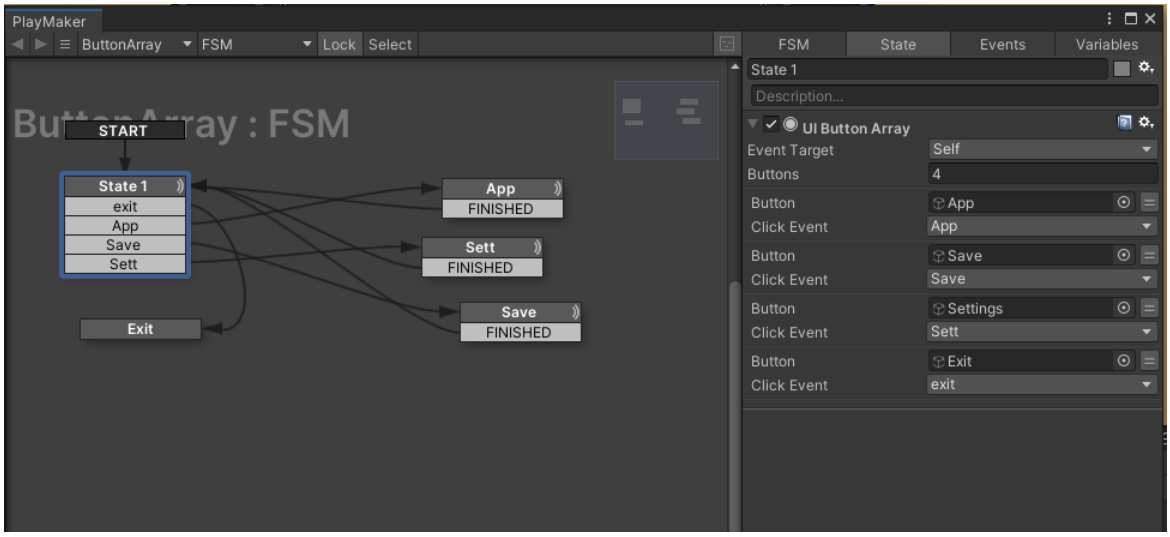

Рисунок 3.12 – скрипт для переміщення по ігровому меню

До першого стану додано елемент для додатку масиву з кнопок, при нажаті на які, відкривалося наступне меню, або виконувалась дія. Принцип дії зображено на рисунку 3.13.

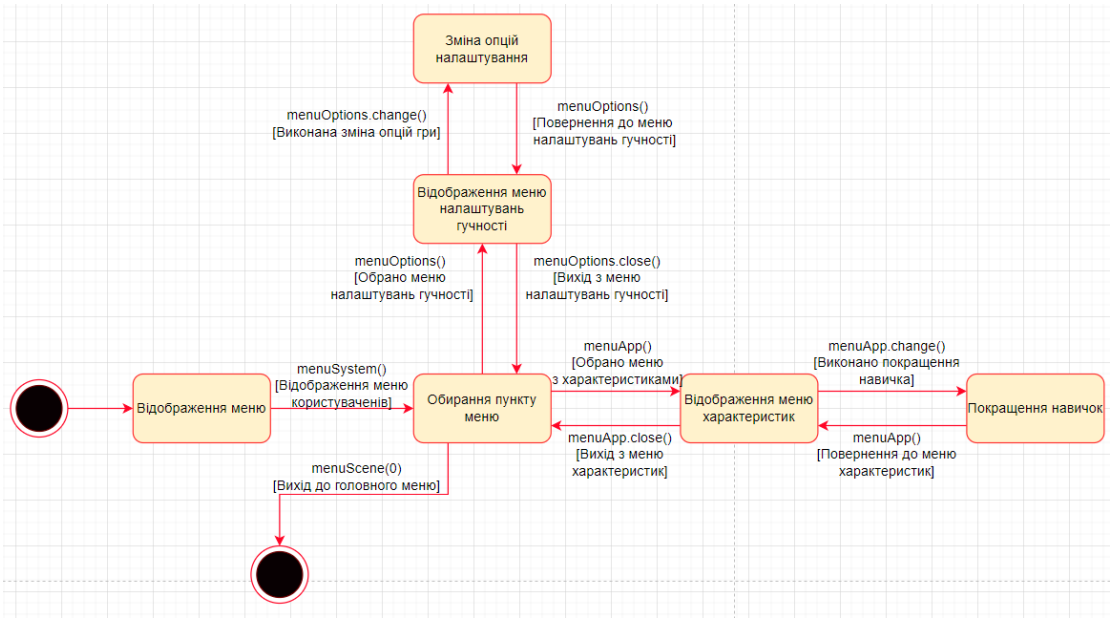

Рисунок 3.13 – діаграма дій в меню

На діаграмі можна побачити усі дії, які викликаються шляхом натискання на кнопки у цьому меню.

3.4 Логіка ігрового меню

Головне меню гри грає не останню роль при створенні ігрового продукту, це перше місце, де гравець зустрічається з грою. Інтерфейс було намальовано в онлайн додатку, в Unity потрібно до нього додати кнопки для керування. Створене меню зображено на рисунку 3.14.

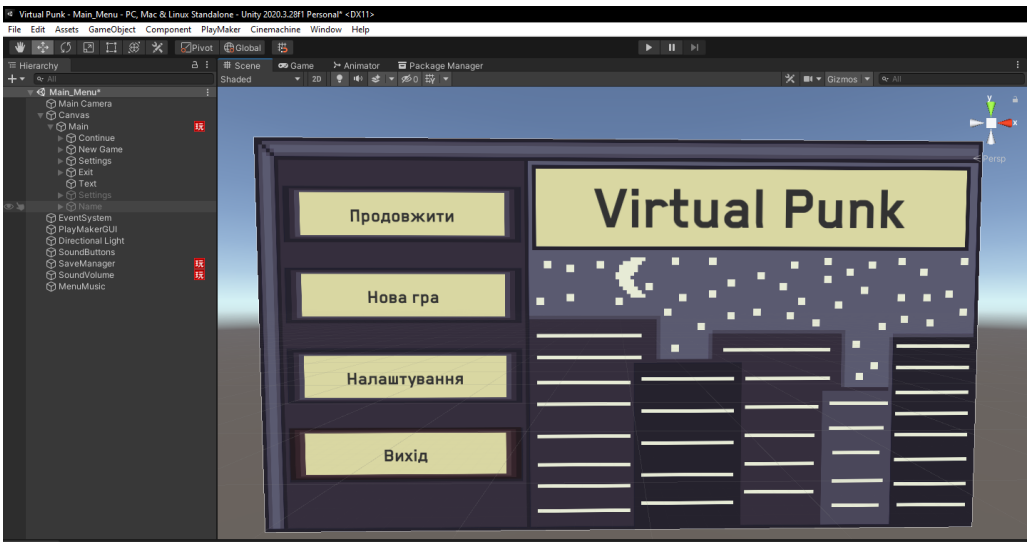

Рисунок 3.14 – головне меню гри

При переході до налаштування відкривається панель з налаштуванням графіки, яке в грі реалізоване за допомогою ефектів пост-обробки. Меню налаштувань зображено на рисунку 3.15.

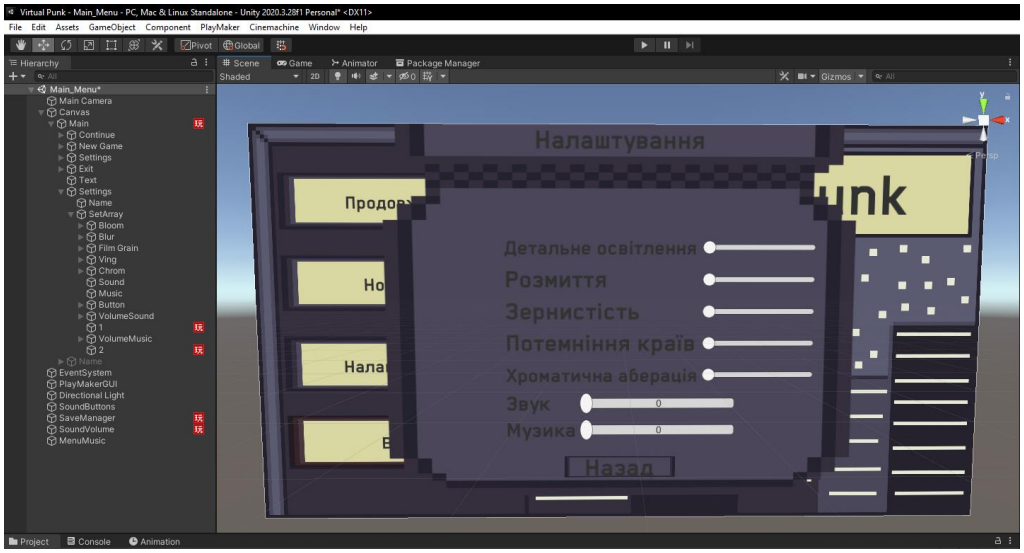

Рисунок 3.15 – панель налаштування в меню гри

При переході в нову гру, користувачу потрібно написати ім’я для свого персонажа, панель для заповнення імені зображено на рисунку 3.16.

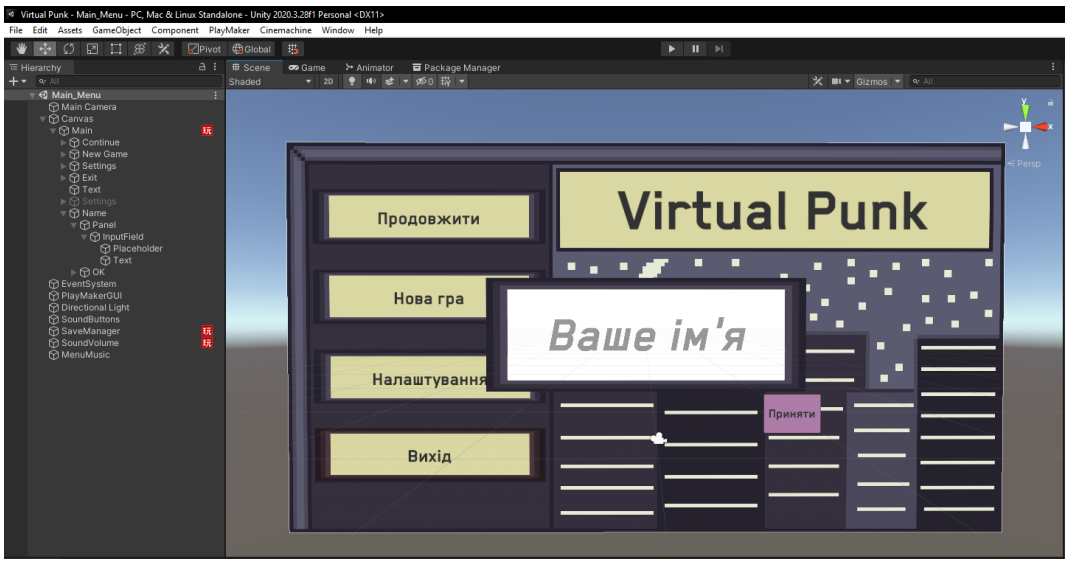

Рисунок 3.16 – панель з заповненням імені персонажа

До меню додано скрипт Playmaker, котрий реалізовує керування в меню, схему зображено на рисунку 3.17.

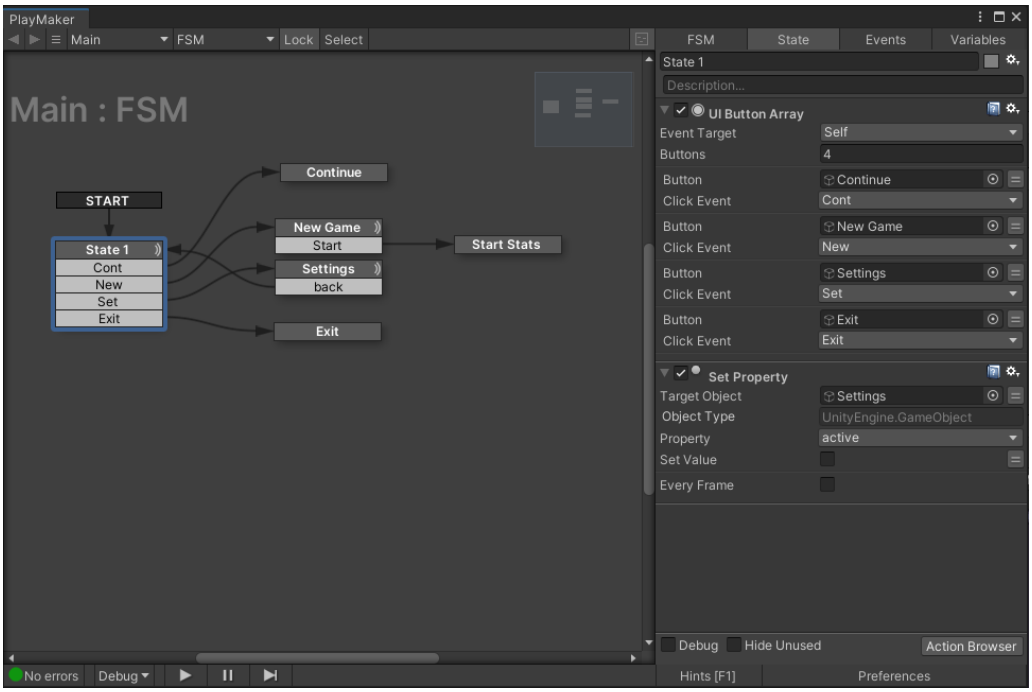

Рисунок 3.17 – логіка ігрового меню

Логіка дуже проста, при натисканні кнопки “Продовжити”, гра загружає збереженні данні, та користувач може продовжити гру за свого персонажа з тими ж характеристиками, з якими він виходив з гри, “Нова гра” – при

натисканні видаляє усі збереженні данні, відкриває меню для запису імені, котре потім буде відображатись в грі, та додає стартові характеристики для персонажу, при натисканні на кнопку “Налаштування” відкривається панель з налаштуванням, де можна увімкнути або вимкнути ефекти пост-обробки, або змінити гучність гри, кнопка “Вихід” відповідає за вихід з ігрового додатку. Діаграма логіки ігрового меню зображено на рисунку 3.18.

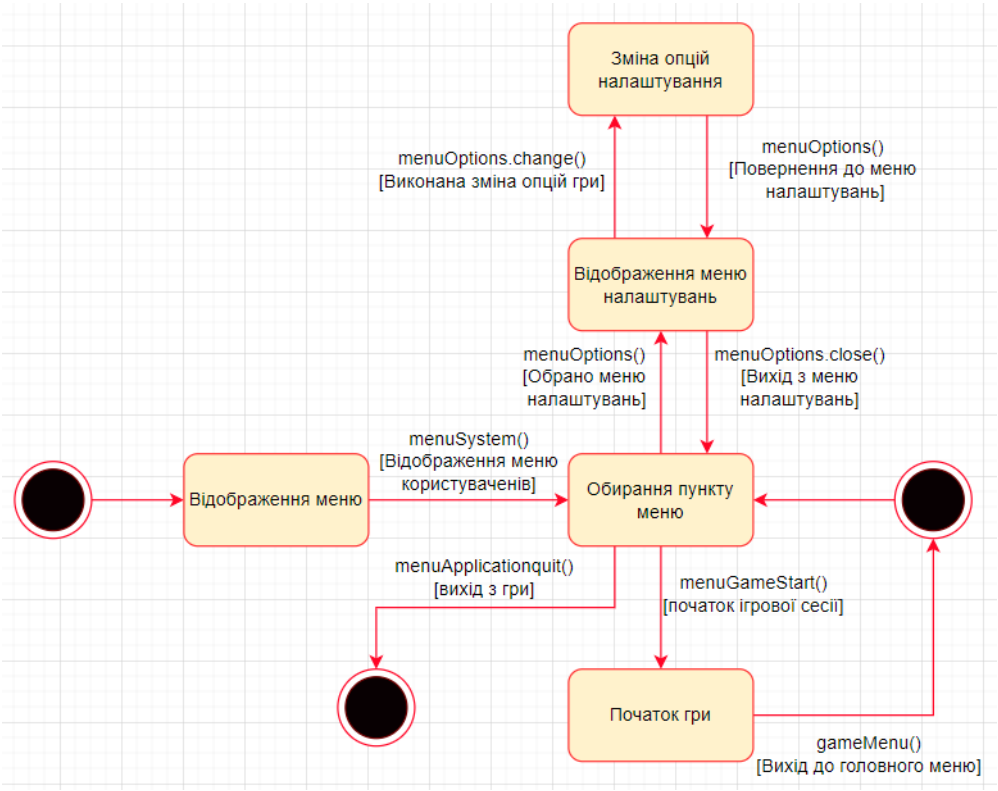

Рисунок 3.18 – діаграма логіки ігрового меню

На діаграмі можна побачити усі переходи, та дії, які викликаються при натисканні кнопок.

3.5 Ігровий процес, простий штучний інтелект

Ігровий процес – це сукупність правил, взаємодій та подій, які відбуваються в грі. Цей процес визначає механіки гри, завдання гравця, взаємодія з оточуючим світом, тощо.

Оскільки розроблюваний ігровий продукт перебуває у жанрі покрокової стратегії, було вирішено розробити завдання-події, які з’являються у відкритому світі.

Для простої бойовки з елементами рольової гри, було добавлено характеристики героя. Кожна характеристика надає параметри герою, а саме кількість здоров’я, сила пошкодження з різних видів атаки, тощо.

До розробленого продукту було додано такі характеристики, як сила, витривалість, спритність, інтелект, удача. Кожна характеристика надає різні параметри, наприклад витривалість надає герою кількість здоров’я у розмірі x*4, тобто, якщо витривалість герою дорівнює 5, то кількість здоров’я буде дорівнювати 20.

Герой після бою отримує досвід, досвід надається герою за формулою генерації випадкового числа від 10 до 150, та до цього додається доповнений досвід, який вираховується за формулою 10*рівень герою.

Характеристики герою, та їх підвищення, а також показ рівню, кількості досвіду можна подивитись в меню приставки у відповідному меню, яке зображено на рисунку 3.19.

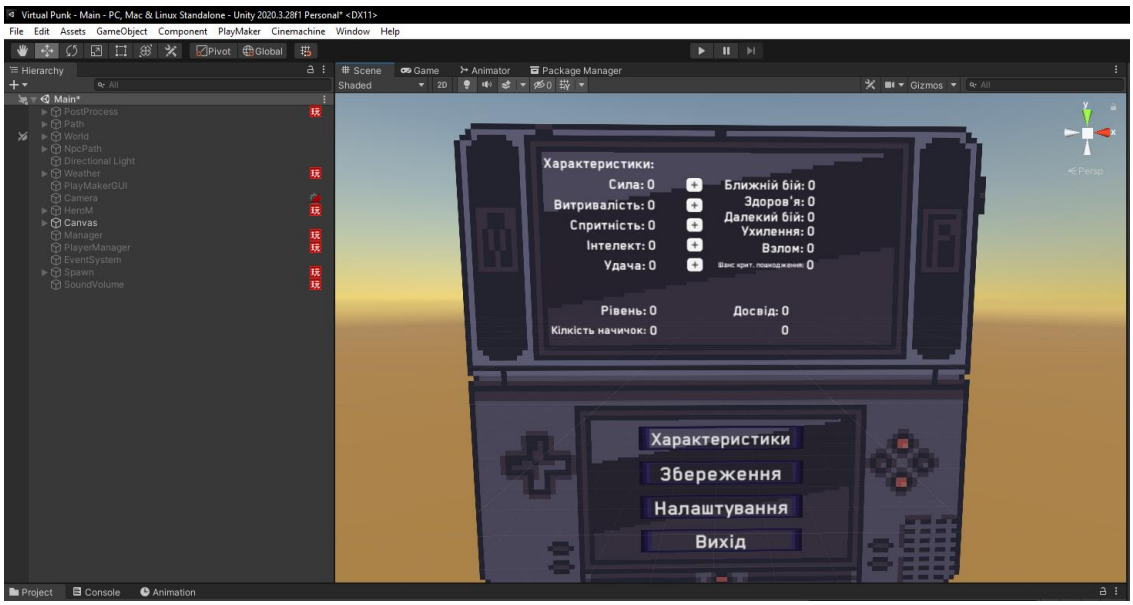

Рисунок 3.19 – Меню з характеристиками

Бій в грі повністю відтворено за допомогою візуального програмування Playmaker, у гравця є вибір із трьох атак, пошкодження з кожної залежить від того, які характеристики підвищував користувач, окрім атак, також є можливість уникнути бою, якщо є підозра програшу.

Для балансу ворогів, було створено систему автоматичного підвищення сили ворога, тобто, противник буде генерувати свій рівень здоров'я, відштовхуючись від кількості здоров'я герою. Для встановлення кількості пошкодження, проводиться перевірка, який у героя навик більш удосконалений, тобто, якщо у героя далекий бій дорівнює 10, а ближній бій 5, то противник, буде генерувати свій рівень нанесення пошкоджень опираючись на далекий бій герою. Окрім цього, для супротивників було намальовано свої іконки профілю, а також перед початком бою, для супротивника генерується випадкове ім'я, яке обирається зі списку.

Схему бою, яке відтворене в FSM скрипті зображено на рисунку 3.20.

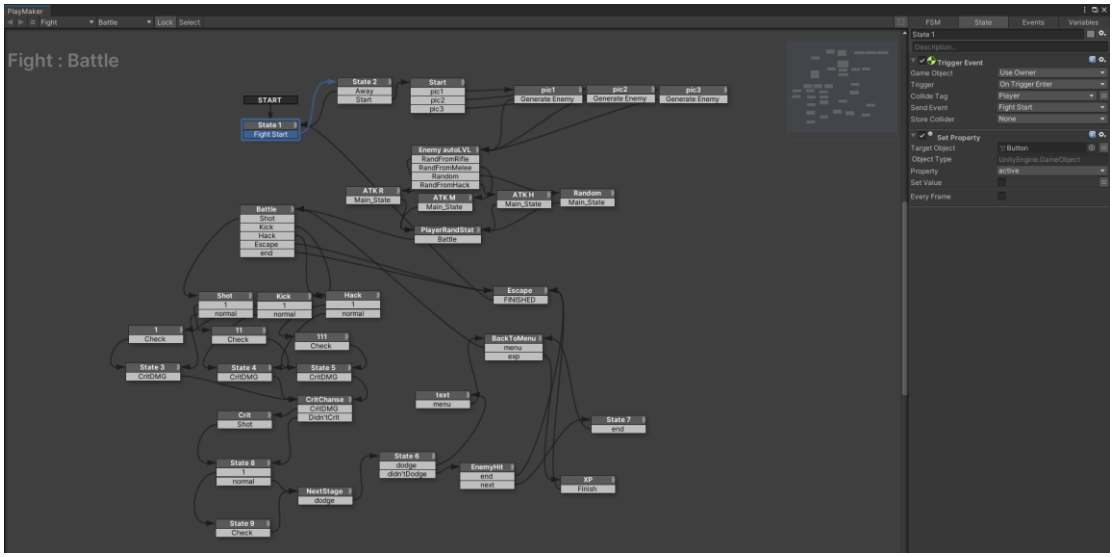

Рисунок 3.20 – схема проведення бою в FSM вигляді

Для пояснення, коли гравець підходить до зони проведення бою, потрібно натиснути клавішу дії, після чого генерується супротивник за рахунок характеристик герою, а також йому надається ім'я, та іконка профілю. Гравцю під час бою дається вибір дії, тобто яку з атак використовувати, після

чого вираховується саме пошкодження. Після нанесення пошкодження, йде хід ворога, атака якого базується на силі навичок героя.

Для відображення бою було намальовано невеликий інтерфейс до нього, який зображено на рисунку 3.21.

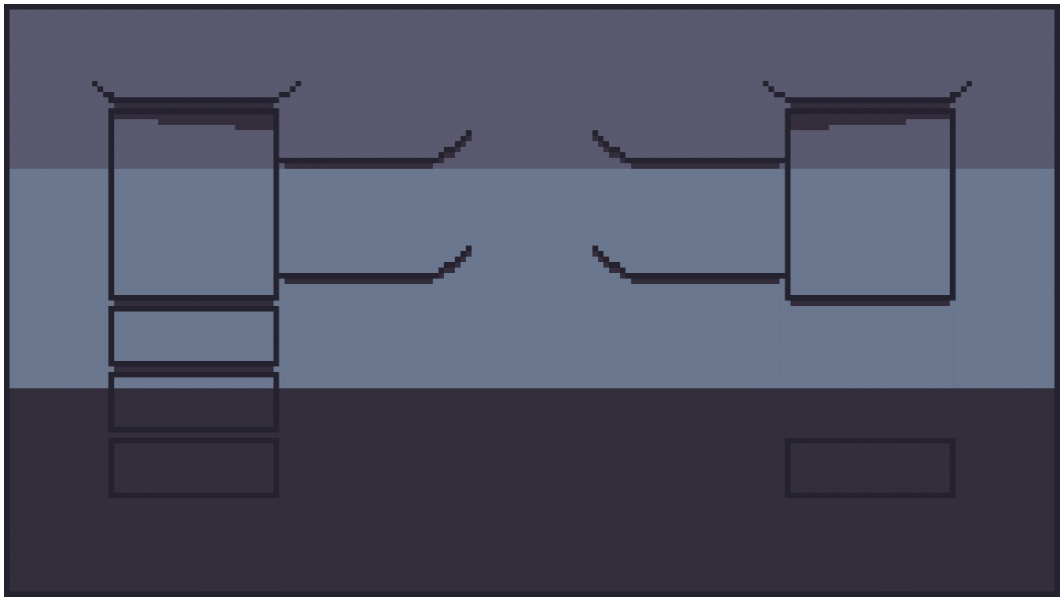

Рисунок 3.21 – інтерфейс бою

Усі інші компоненти до нього додаються у самому просторі Unity, повний інтерфейс бою зображено на рисунку 3.22.

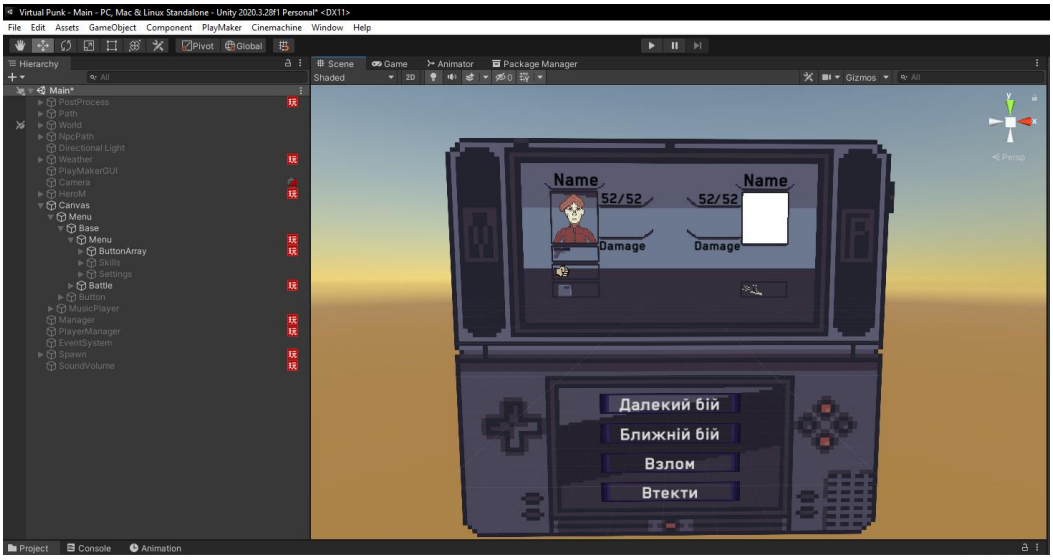

Рисунок 3.22 – заповнений інтерфейс бою

Для керування використовуються кнопки, які розташовані на нижньому екрані портативної приставки.

Подія, яка активує бій, можна спостерігати в ігровому всесвіті, ця подія має вигляд, який зображено на рисунку 3.23.

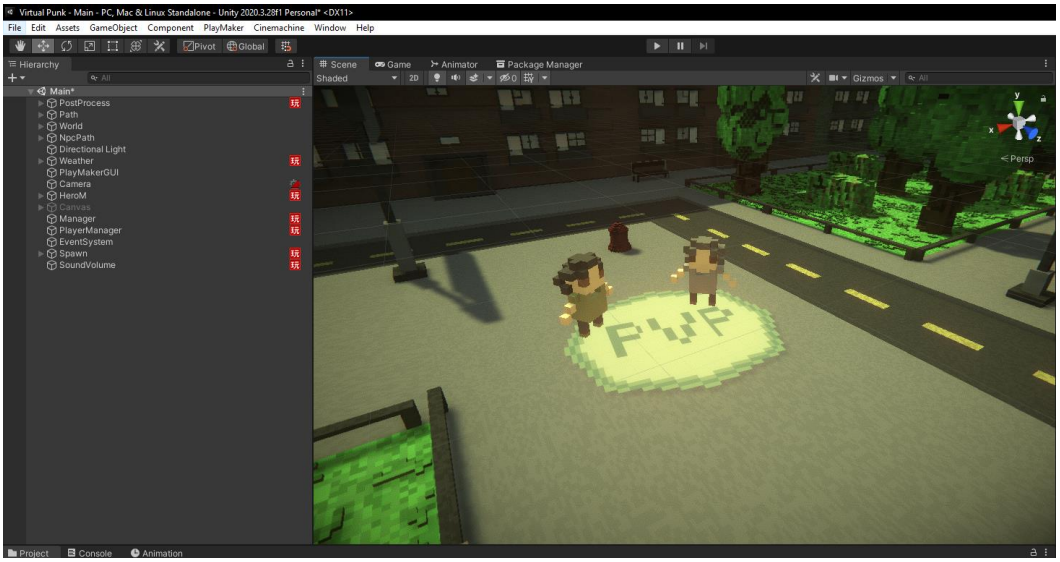

Рисунок 3.23 – позначення події для входу до бою

Окрім цього, до ігрового світу було додано просту систему трафіку, а також пішоходів. Для цього було створено два скрипти на мові C#, тому що у просторі візуального програмування Playmaker має дуже просту працездатність з масивами, котра не підходить до додавання трафіку, та руху пішоходів.

Перший скрипт створює шлях, за допомогою масивів об’єктів, які потрібно розташувати у ігровому всесвіті, окрім переміщення об’єкту по цьому шляху, скрипт має випадкове створення об’єкту, яке потрібно занести до масиву. Другий скрипт потрібно додати до пішоходів та транспорту, він надає змогу об’єкту переміщуватись за шляхом, який створено у першому скрипті. Заповнений ігровий світ з трафіком та пішоходами зображено на рисунку 3.24.

Рисунок 3.24 – ігровий світ з системою трафіку, пішоходами, та подіями

Рух по цьому світу виконується за допомогою клавіш WASD, а для обзору використовується комп’ютерна миша.

Діаграма усього ігрового процесу зображена на рисунку 3.25.

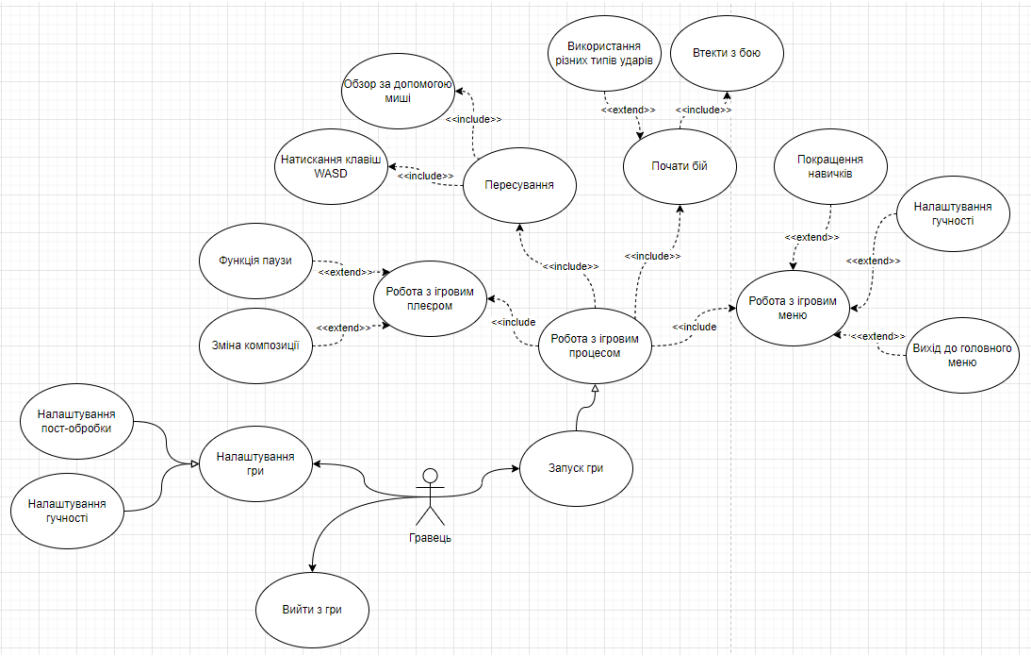

Рисунок 3.25 – діаграма ігрового процесу

На діаграмі можна побачити можливості гравця, а саме робота з меню, в ігровому процесі зміна налаштувань гучності, покращення навичок, початку бою, роботу з плеєром.

3.6 Створення та додавання плеєру до проєкту

Одним із важливих елементів гри – є музика. Наприклад в грі “Doom”, музика зроблена на декількох станів, тобто одна композиція розділена на спокійний стан, на стан в бою, та стан після бою, самі композиції настільки гарні, що цю гру називають доповненням до самого музичного альбому. Є подібний варіант використання музики, який було реалізовано в грі “Devil May Cry 5”, в цій грі, чим більше гравець вибиває комбінацію ударів, тим більше сама композиція буде доповнюватись, наприклад при вході в бій грає лише мелодія композиції, при більшому комбо додаються барабани, гітара, басова частина, та при максимальному комбо – додається вокал, з цим пов’язана доволі кумедна ситуація, коли ігрові журналісти скаржилися на відсутність музики, була лише мелодія, і лише вона. Тобто вони занизили оцінки тому що, дуже погано грали.

У варіанті плеєру для прототипу, була обрана така модель, яка використовується у деяких ігор у відкритому світі, наприклад як в грі “Grand Theft Auto V”. Щоб не писати композиції для гри, користувач сам додає до гри свої композиції.

Аудіоплеєр може бути важливим елементом гри з кількох причин. Перш за все, він допомагає створювати належний настрій та поглиблює емоційну залученість гравця до гри. Варто відзначити, що звукове супроводження є невід’ємною частиною ігрового досвіду, яке додає реалізм та іммерсивність. Також причина полягає у розширенні можливостей геймплею, тобто аудіоплеєр дозволяє гравцям включати власну музику зовнішньої бібліотеки, це розширює можливості гравців і дає їм можливість грати під супровід улюбленої музики, яка відповідає їхньому смаку.

Враховуючи всі ці фактори, можна зрозуміти, що аудіоплеєр може бути не тільки додатковою функцією гри, але й важливим елементом, який сприяє більш іммерсивному та персоналізованому ігровому процесу.

Розроблений аудіоплеєр має кілька переваг, наприклад, він має простий та інтуїтивно зрозумілий інтерфейс для переключення та включення композицій. Користувачі можуть використовувати власну музику до гри, створити свій плейлист.

Скрипт аудіоплеєру легко модифікувати, додаючи до нього нові можливості та функціонал, наприклад можна додати випадковий вибір композиції, або додати до використання інші каталоги, щоб створити повноцінне ігрове радіо, де користувач може додати до кожного свій плейлист.

Даний скрипт являється унікальним серед музичних додатків в маркеті Unity, його можна використати як продукт для продажу або ліцензування іншим розробникам ігор.

В самому Unity, до самого плеєру треба додати кнопки для керування, та додати поле для відображення назви композиції. Музичний плеєр в Unity зображено на рисунку 3.26.

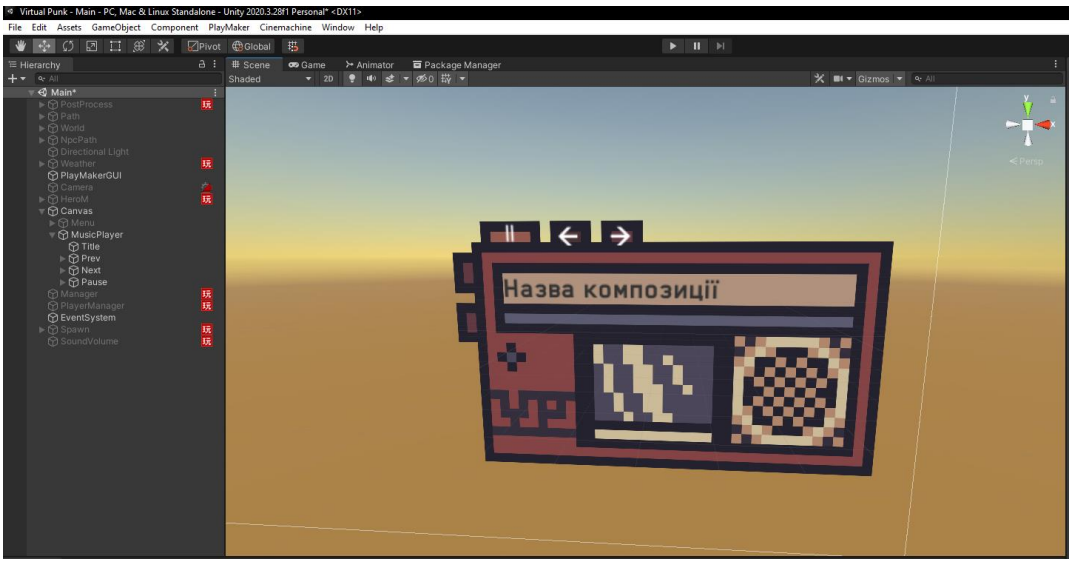

Рисунок 3.26 – музичний плеєр в Unity

Оскільки Playmaker не надає можливостей до використання посилань на файл, було написано скрипт на C#, лістинг коду якого можна знайти у додатку. Скрипт працює наступник образом, у каталозі гри, є каталог “Music”, куди користувач завантажує свої композиції, скрипт бере усі медіа файли формату .mp3, та додає до масиву, переключення між піснями виконується за допомогою кнопок, які додані в самому Unity, також в налаштуваннях гри, буде можливість змінити гучність музики.

Якщо подивитись на маркет доповнень Unity, то там буде відсутнє доповнення з подібною працею плеєру, можна знайти лише плеєр, який використовую URL посилання, але такий спосіб може відштовхнути від ігрового процесу, так як потрібно постійно копіювати посилання на композицію для її відтворення, з таким функціоналом легше буде увімкнути музику на самому комп’ютері.

Тестування самого плеєру буде зображено у четвертому розділі.

3.7 Додаткові аудіо елементи гри

Додаткові аудіо елементи в грі можуть бути використані для поліпшення іммерсивності та взаємодії гравця з грою.

Для створення аудіо ефекту, була використана програма Fl Studio. В цій програмі додаємо до інструментів плагін “Sytrus”, що зображено на рисунку 3.27.

Рисунок 3.27 – плагін для Fl Studio

У цьому плагіні шукаємо потрібний нам звук, котрий будемо оброблювати, наприклад, це буде інструмент під назвою “Тао”. Після додавання потрібно обрати нову доріжку для нього, щоб розпочати обробку. Для обробки вибираємо еквалайзер, та реверберацію для пониження високих частот звуку, та збільшення низький, окрім цього, додаємо невеликий ефект відголосу, настройки обробки зображено на рисунку 3.28.

Рисунок 3.28 – налаштування обробки звуку

Після цього, додаємо до доріжки отриманий звук, змінивши при цьому його тривалість, та експортуємо у формат .mp3. В Unity додаємо пустий об’єкт, підпишемо його “Sound”, до цього об’єкта додаємо в компонент “AudioSource”, в який завантажуємо отриманий звук, далі до кожної кнопки зробимо на нього посилання, та увімкнемо функцію “PlayOneShot”. Кнопка з такою функцією зображена на рисунку 3.29.

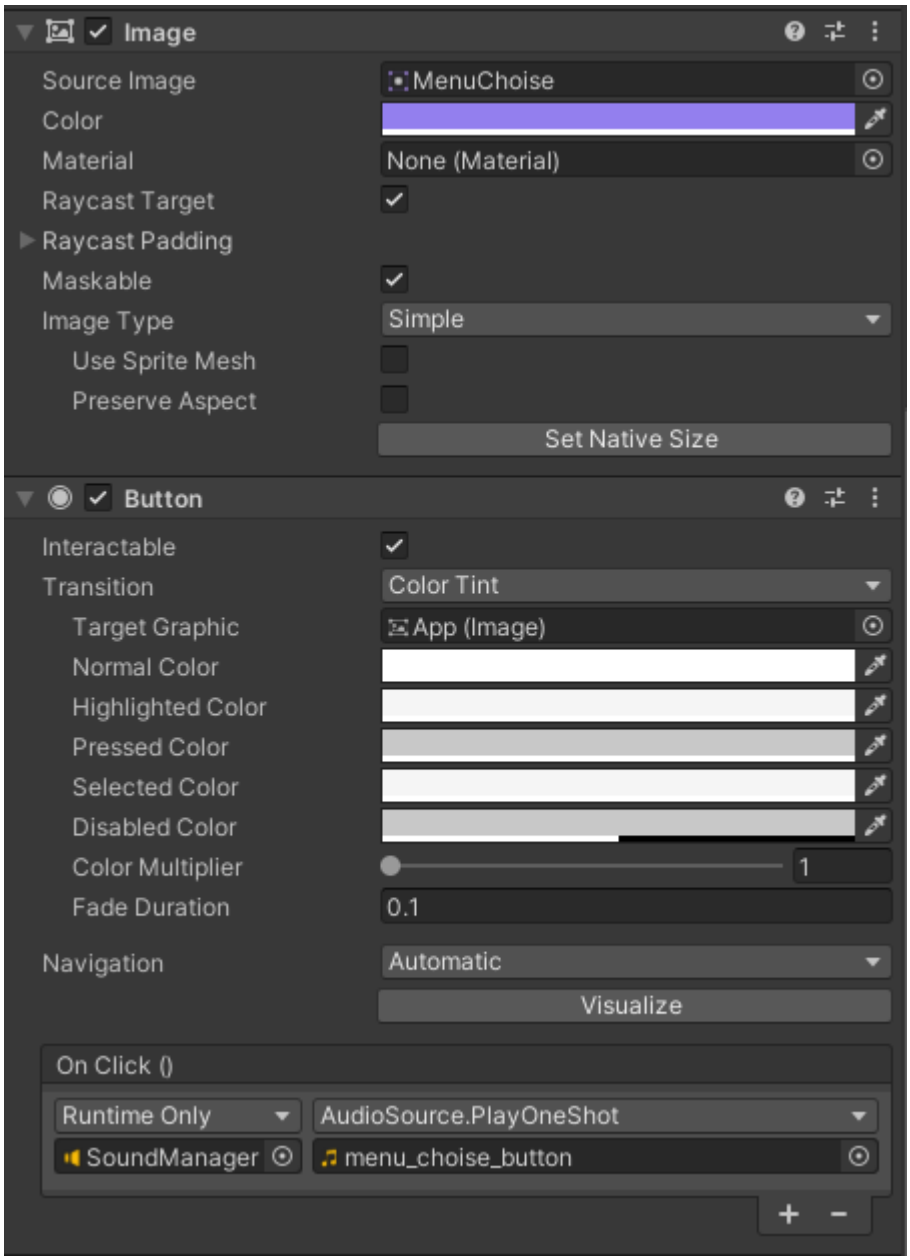

Рисунок 3.29 – кнопка з посиланням на звук

Після цього, якщо натиснути на кнопку, буде програватись звук.

3.8 Зберігання ігрових даних

В наш час неможливо представити гру без функції збереження даних, яка дозволяє гравцям зберегти свій прогрес і повертатись до гри у будь-який зручний момент без втрати усього прогресу.

Збереження даних в грі дозволяє гравцям продовжити гру з того самого місця, де вони зупинилися. Особливо це корисні для ігор з великою кількість активностей.

Функція збереження у ігровому додатку реалізована за допомогою візуального програмування PlayMaker, та його функціями “PlayerPrefs”, за допомогою чого, можна зчитати, або записати змінні.

Розроблений ігровий продукт має такі змінні, як характеристики герою, також було додано змінні на налаштування графіки та гучності звуку й музики. Для збереження необхідно спочатку записати усі данні, в ігровому додатку це реалізовано за допомогою FSM скрипту, який визивається за допомогою кнопки “Збереження” у ігровому меню. Скрипт зображено на рисунку 3.30.

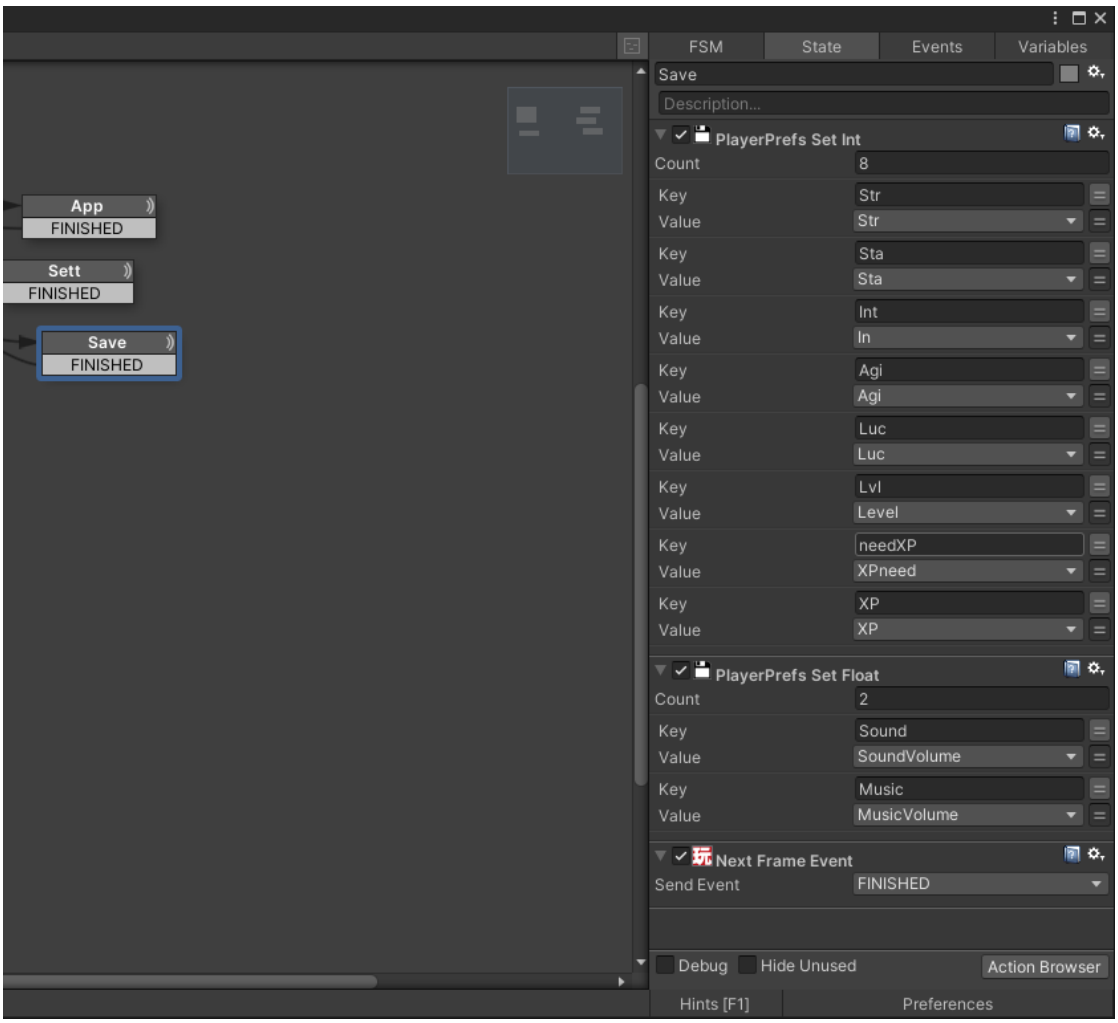

Рисунок 3.30 – встановлення до PlayerPrefs змінних

Щоб завантажити збереженні змінні, у головному меню, при натисканні кнопки “Продовжити”, у цьому варіанті, скрипт зчитує данні, які було записано до цього. Зчитування даних у FSM скрипті зображено на рисунку 3.31.

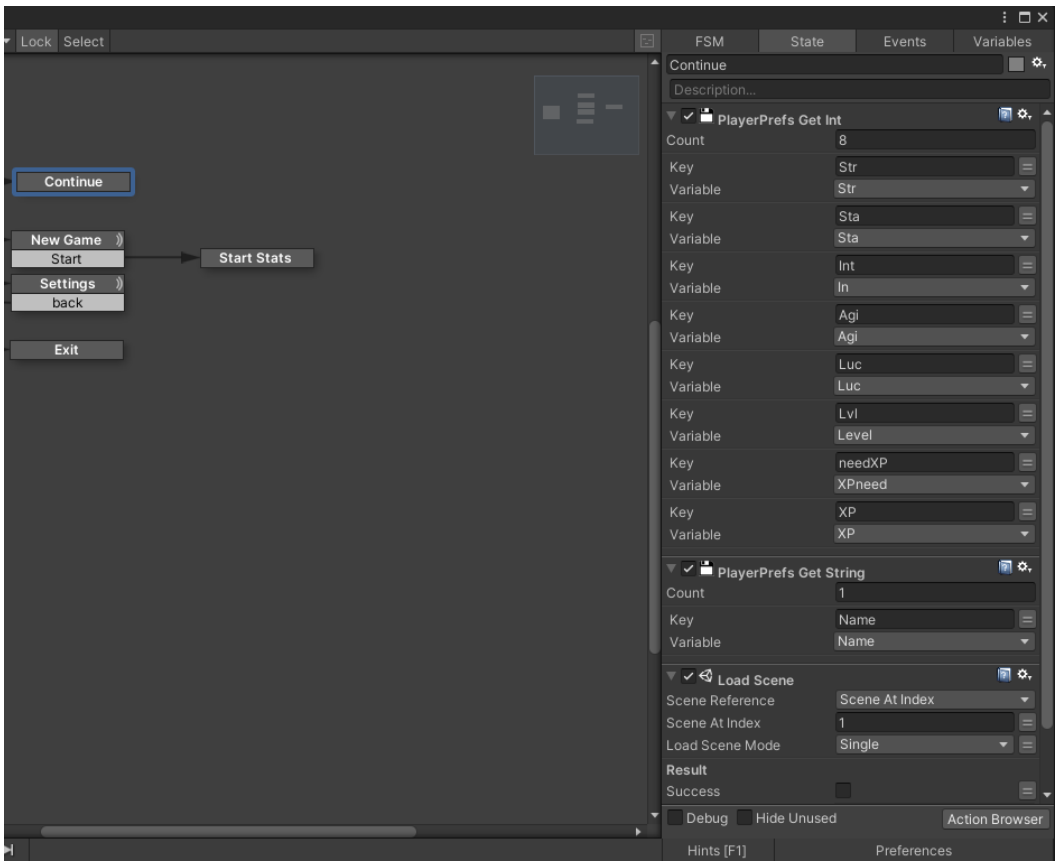

Рисунок 3.31 – зчитування збережених змінних

Таким чином, розроблений ігровий додаток має таку важливу функцію, як збереження даних.

3.9 Висновки за розділом

З метою демонстрації працездатності розробленого плейеру створено й описано прототип гри, яка включає створені 3D об’єкти за допомогою програми MagicaVoxel, були створені скрипти для системи трафіку, та руху пішоходів. Була розроблена логіка проведення бою, надано характеристики для герою, також було зроблено ігрове меню для гри, налаштування графіки та гучність звуку та музику. Було розроблено аудіоплеєр, котрий вмикав композиції користувача.

4 ТЕСТУВАННЯ ПРОГРАМНОЇ РОЗРОБКИ

4.1 Робота з меню

При вході до гри, нас зустрічає головне меню гри, вигляд даного меню зображено на рисунку 4.1.

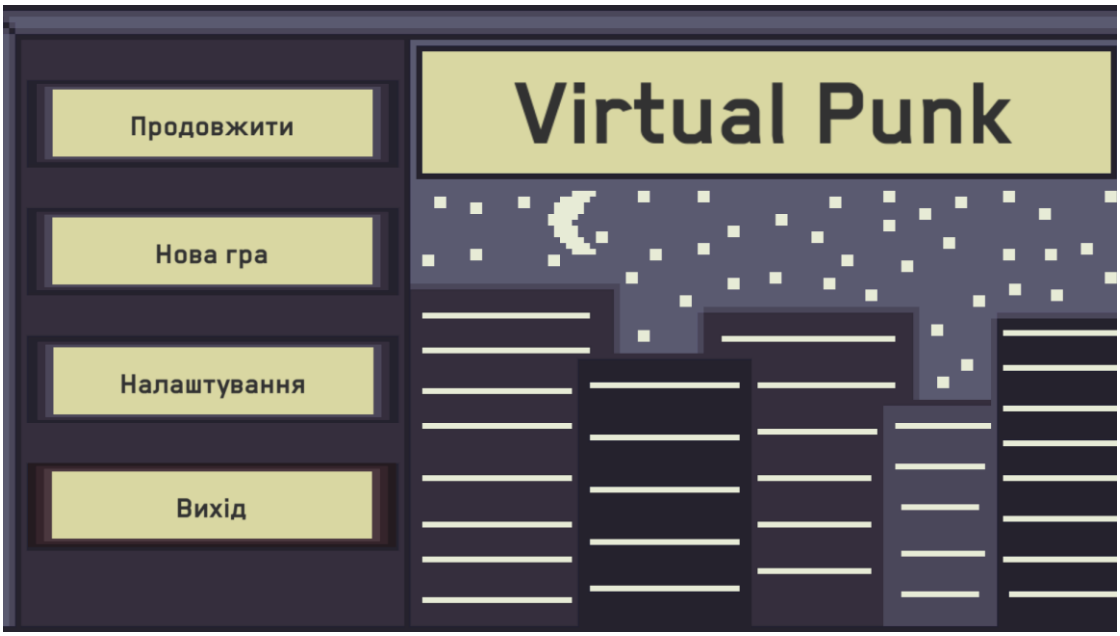

Рисунок 4.1 – ігрове головне меню

Натискаємо на кнопку з налаштуваннями, після цього відкривається меню налаштувань, в якому є можливість увімкнути або вимкнути ефекти пост-обробки, або змінити гучність, меню з налаштуваннями зображено на рисунку 4.2.

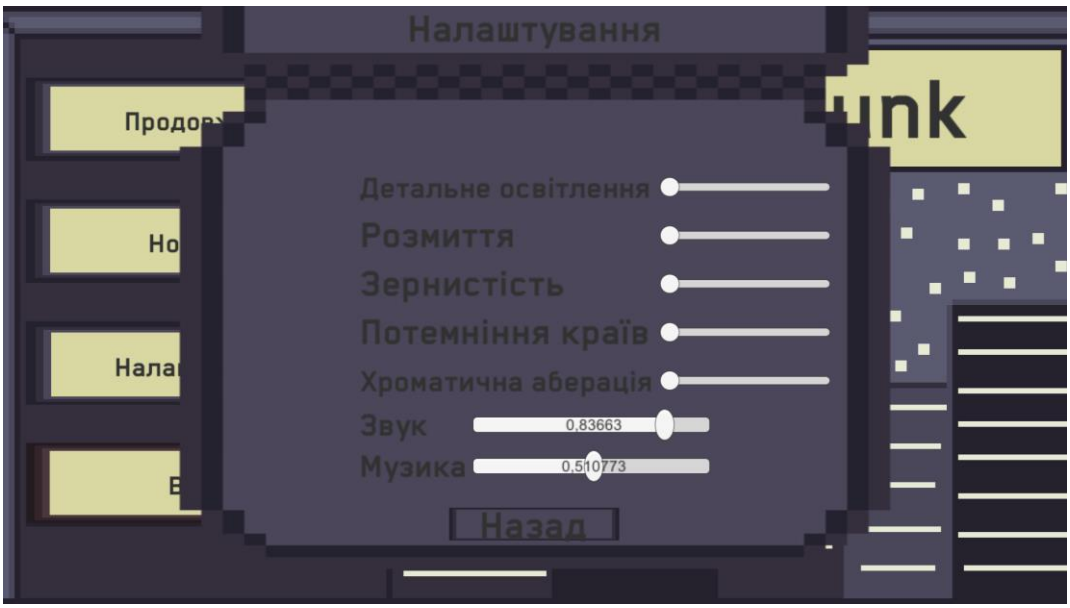

Рисунок 4.2 – панель налаштувань

Нажимаємо на кнопку назад, та переходимо до нової гри, де нас зустрічає панель з вводом імені, дана панель зображена на рисунку 4.3.

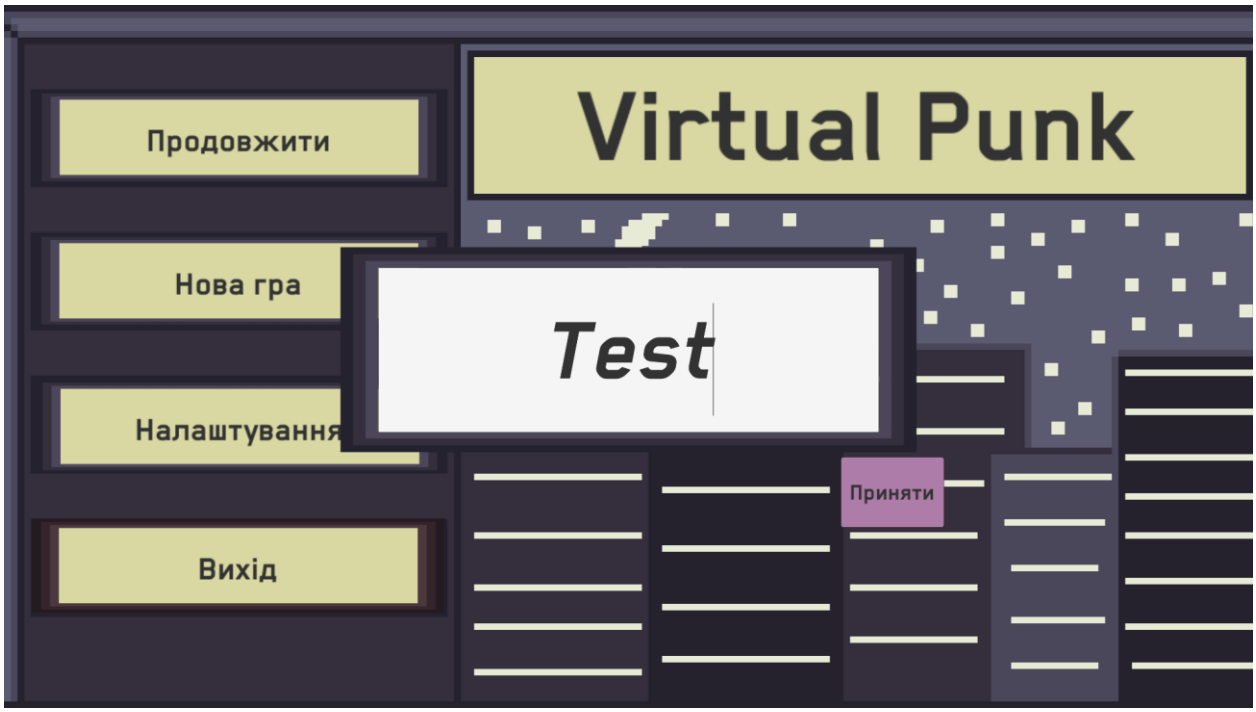

Рисунок 4.3 – панель вводу імені персонажа

Окрім цього було перевірено кнопку “Вихід”, при натисканні на неї проводиться вихід з додатку. При натисканні на кнопку “Продовжити”, гра починається за вже існуючого персонажа.

4.2 Перевірка роботи звука

Для перевірки роботи звуку було обрано програму OBS – це безкоштовна та відкрита програма для запису та трансляції відео та аудіо контенту, за допомогою її інтерфейсу з’являється можливість подивитись на рівень звуку.

У головному меню є своя композиція, яка також реагує на налаштування гучності, тож перша перевірка буде зроблена на тестування гучності саме у головному меню, результат буде зображено на рисунках 4.4 та 4.5.

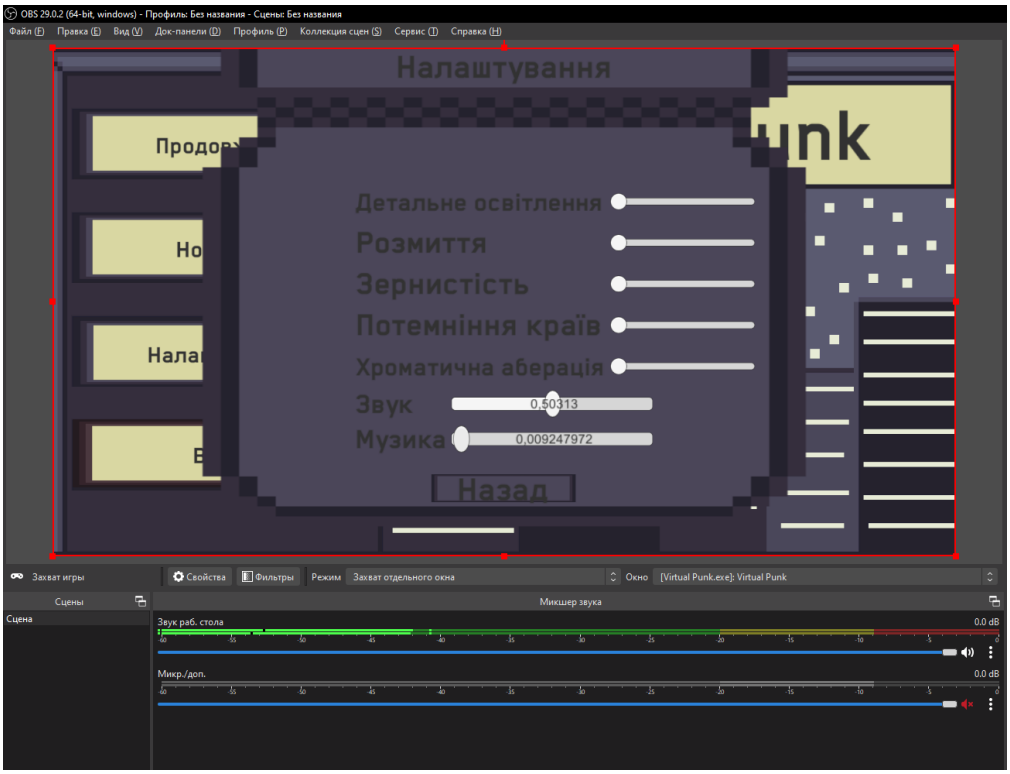

Рисунок 4.4 – перевірка гучності музики у головному меню

При налаштуванні гучності майже мінімально можливого рівня, можна спостерігати, шо гучність приблизно на рівні -42 Дб.

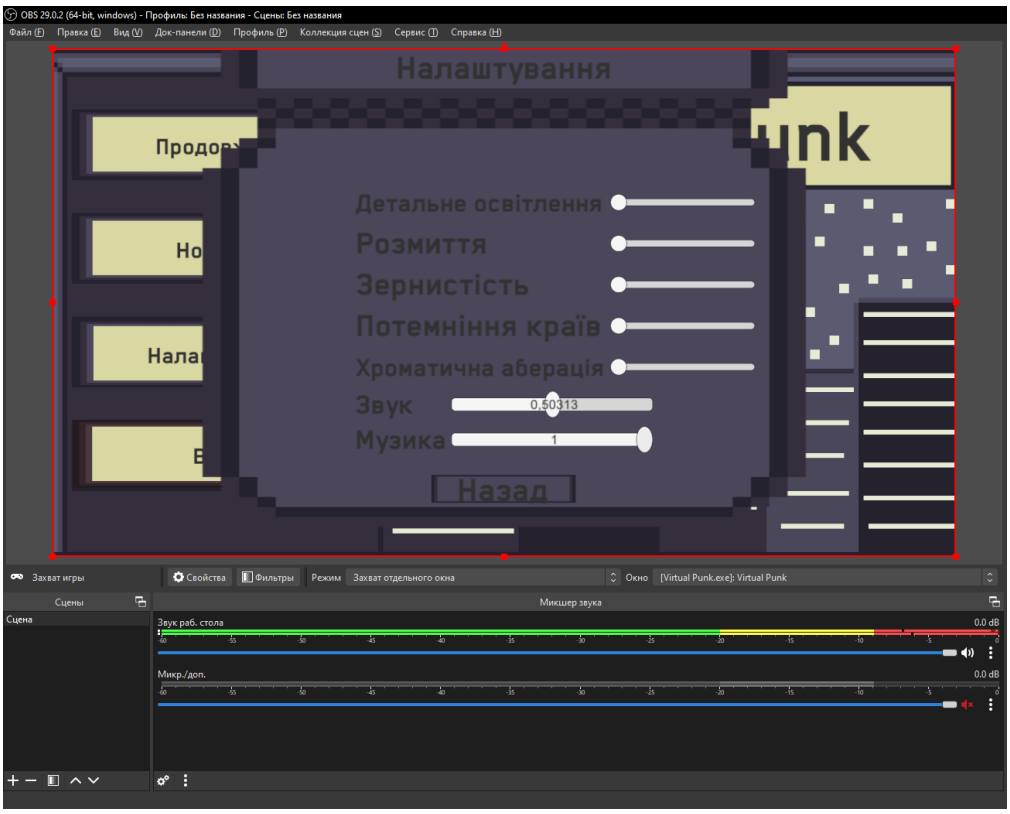

Рисунок 4.5 – друга перевірка гучності музики у головному меню

Можна спостерігати, що при максимальному налаштуванні гучності музики, наша гучність перевищує рівень 0 Дб.

Також проведемо перевірку роботи плейеру, та вплив налаштувань гучності до нього, спочатку потрібно завантажити свою музику до каталогу, який знаходиться в кореневому каталозі гри, каталог з завантаженою музикою можна спостерігати на рисунку 4.6.

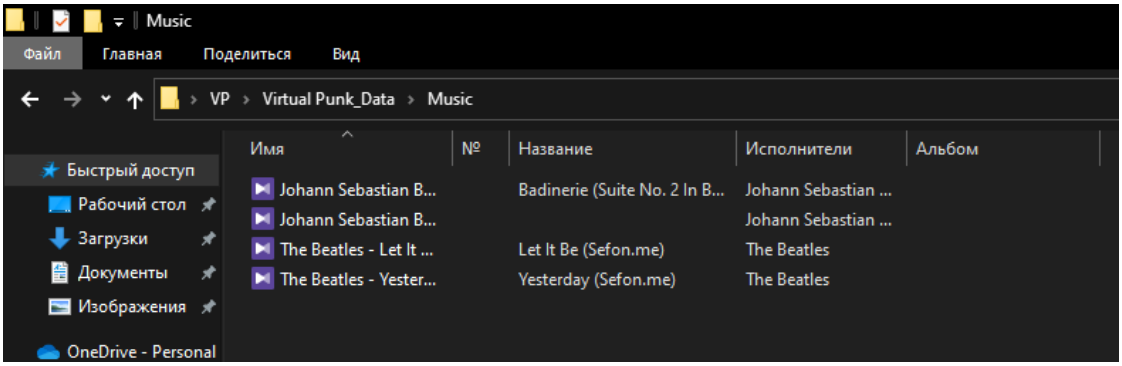

Рисунок 4.6 – каталог з музикою

Далі в самій грі, нам потрібно натиснути клавішу відтворення музики. Перевірка з вікном OBS зображено на рисунку 4.7.

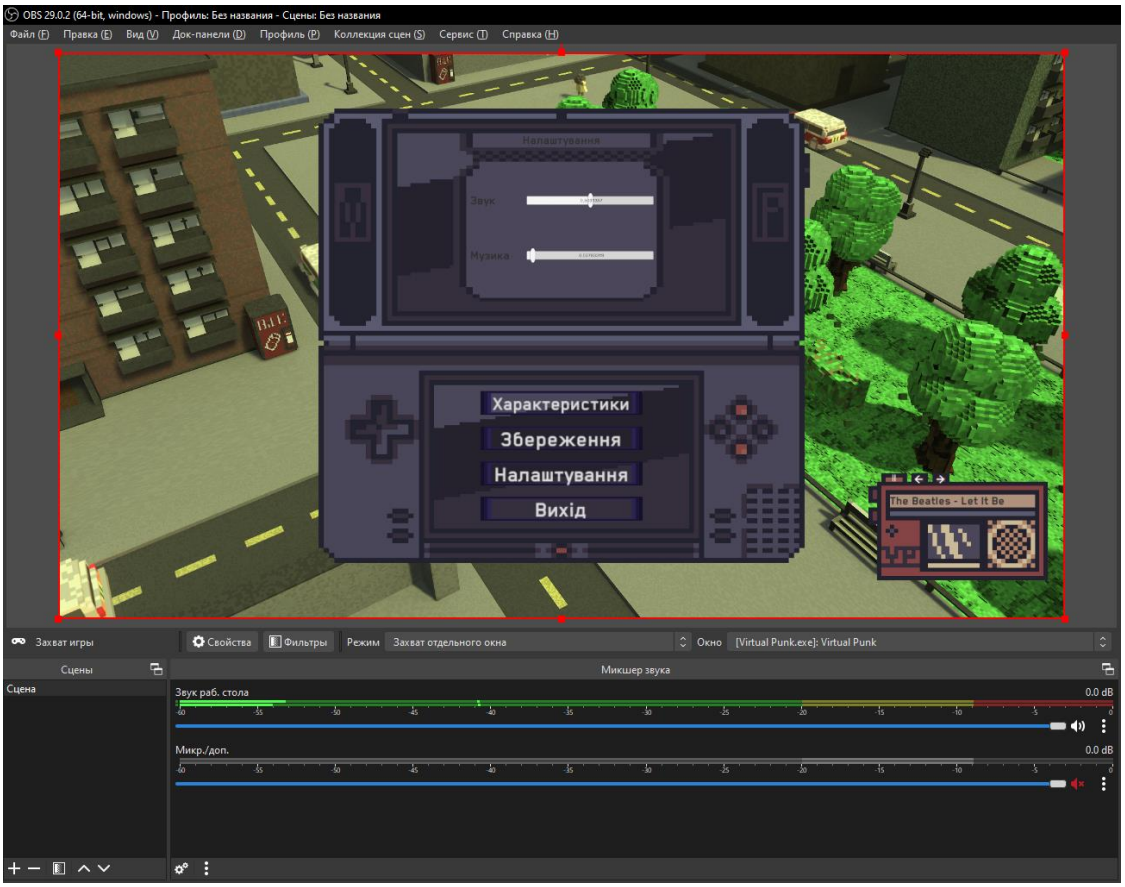

Рисунок 4.7 – перевірка гучності плейеру

При низькому налаштуванні гучності музики, можна спостерігати, що у середньому, рівень гучності тримається на полі -50 Дб. Окрім цього, можна помітити працю самого плейеру, який вивів назву відтвореної композиції, далі змінюємо гучність на максимальну, перевірку цього можна побачити на рисунку 4.8.

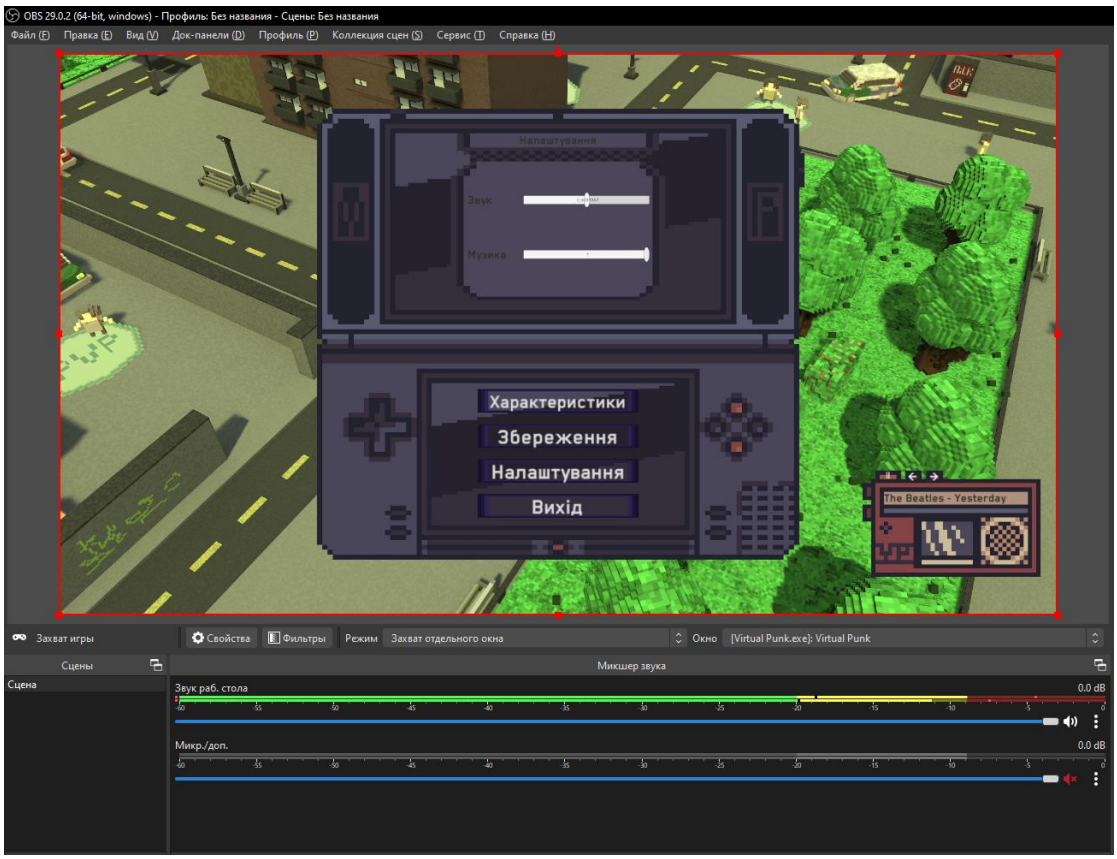

Рисунок 4.8 – перевірка максимальної гучності плейеру

При максимальному налаштуванні гучності, можна помітити, що гучність композиції, яка відтворена плейером досягає у середньому рівень -10 Дб. Різні полоси говорять про те, що композиції, які програються за допомогою плеєру, мають стереозвук. Також продемонстровано те, що плейер дійсно виводить назву пісень, це можна спостерігати на рисунку 4.9.

Рисунок 4.9 – перевірка відображення назви відтвореної композиції

Можна побачити, що відображається саме та композиція, яка відтворюється у поточний час.

4.3 Зберігання даних

Тестування збереження ігрових даних будемо робити на основі існуючого персонажу, його характеристики зображено на рисунку 4.10.

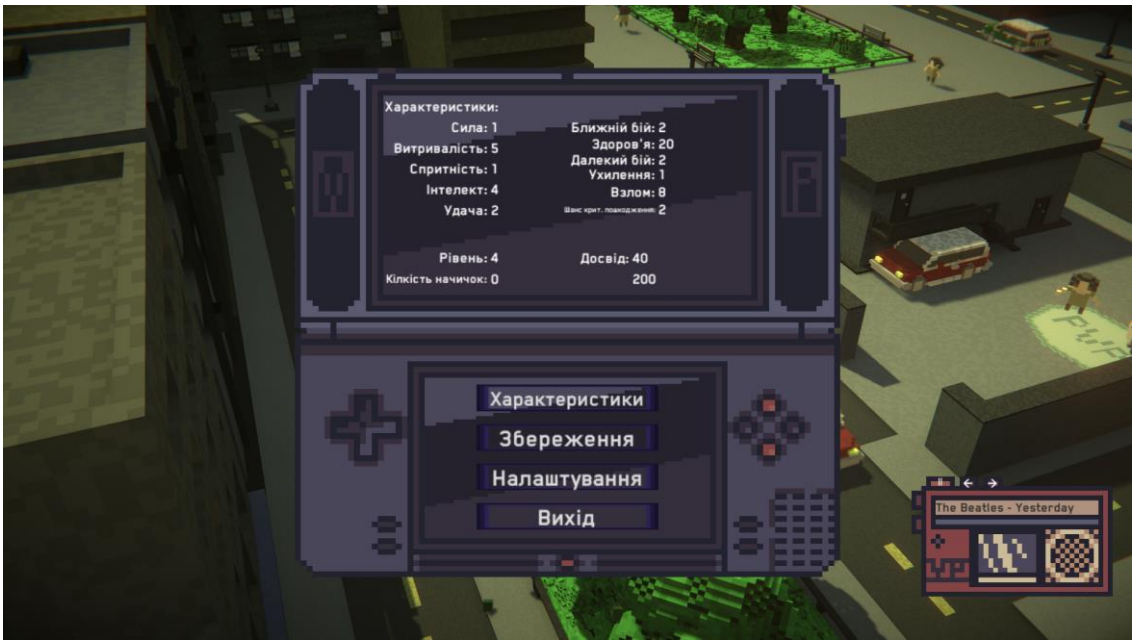

Рисунок 4.10 – характеристики героя

Натискаємо кнопку “Збереження”, та виходимо до головного меню, де, замість нової гри натискаємо кнопку “Продовжити”, завантажена гра після натискання кнопки зображена на рисунку 4.11.

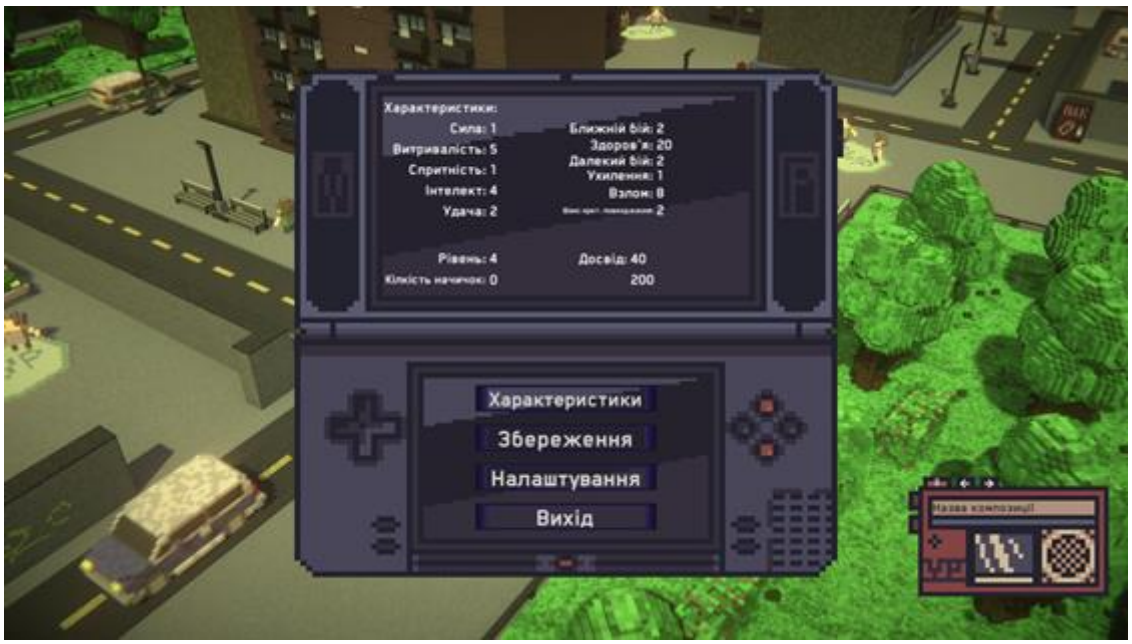

Рисунок 4.11 – характеристики героя після продовження гри

Можна помітити, що характеристики героя залишились незмінними, але сам герой перемістився в точку, на якій його було розміщено в Unity. Окрім характеристик, також збереглися налаштування графіки, та налаштована гучність звуку та музики.

4.4 Висновок за розділом

Було проведено тестування ігрового додатку, в головному меню усі налаштування ефектів пост-обробки та гучності звуку та музики зберігаються, та остаються такими ж після повторного включення гри. Також було перевірено плеєр гравця, котрий відтворював музику користувача, яка була завантажена в каталог “Music”, який знаходиться у кореневому каталозі гри. Була перевірена робота налаштування гучності музики, при перевірці якої, використовувалась програма OBS для спостереження рівня гучності.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи, було проаналізовано та використано інструменти для облегшення створювання ігрових додатків. Окрім цього, було проведено аналіз магазину доповнень Unity, в якому не знайшлось аналогів музичного плейеру.

Ігровий додаток було створено у ігровому двигуні Unity 3D з допомогою доповнення візуального програмування PlayMaker. Музичний плеєр було створено на мові C#.

Для розробки ігрового додатку було використано програми, та онлайн додаток, а саме:

- MagicaVoxel
- VoxelShop
- Blender
- FL Studio
- PixilArt

Під час тестування додатку, було використано програму OBS для визначення рівня гучності.

Всі поставлені задачі виконано, мета роботи досягнута.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Unity User Manual. URL: https://docs.unity3d.com/Manual/index.html

2. Unreal Engine. URL: https://www.unrealengine.com/en-US

3. PlayMaker Manual. URL: https://hutonggames.fogbugz.com/

4. Воксельний редактор MagicaVoxel. URL: https://ephtracy.github.io/

5. Програма для моделювання Blender. URL: https://www.blender.org/

6. Pixilart Main Page. URL: https://www.pixilart.com/

7. Visual Scripting with Bolt. URL: https://docs.unity3d.com/bolt/1.4/manual/index.html

8. Voxel Shop сторінка на Github. URL: https://github.com/simlu/voxelshop

9. Хокінг Д. Unity в дії: мультиплатформена розробка ігор на C# / Д. Хокінг.К.: ДМК Пресс, 2018. 432 с.

10. Маккі С. Створення воксельного мистецтва за допомогою MagicaVoxel. Київ: Довкілля-К, 2019. 240 с.

11. Фрідман Ш. Кулінарна книга FL Studio. Київ: Довкілля-К, 2014. 348 с

12. Сімондс Б. Blender 3D: майстер-майстер-клас: практичний посібник із моделювання, ліплення, матеріалів і візуалізації. Київ: Довкілля-К, 2013. 352 с.

13. Фуллертон Т. Семінар з дизайну ігор: орієнтований на ігри підхід до створення інноваційних ігор. Київ: Довкілля-К, 2014. 520 с.

14. Свінк С. Відчуття гри: посібник дизайнера ігор щодо візуальних відчуттів – Київ: Альтерпрес, 2013. 376 с.

15. Роджерс С. Підвищуйте рівень! Посібник із чудового дизайну відеоігор Київ:.Альтерпрес, 2014. Київ: Довкілля-К, 2018. 1038 с.

16. Ністром Р. Шаблони ігрового програмування. Київ: Довкілля-К, 2014. 354 с.

17. Whitaker R.B. Посібник гравця C# - Київ: Альтерпрес, 2020. 800 с.

18. Майлз Р. Жовта книга з програмування на C# . Київ: Альтерпрес, 2020. 256 с.

19. Моґрідж Б. Проєктування взаємодії. Київ: Довкілля, 2014. 766 с.

Додаток А

Зауваження нормоконтролера

Таблиця А.1 – Зауваження нормоконтролера

Сторінка	Позначення	Зміст зауваження
ПЗ		Нещільне заповнення сторінок
С. 10-12		Рисунки 1.1-1.3 мають зайву крапку у номерах
Розділ 2, …	Окрім 3.1	Назви рисунків з малої літери і не завжди центровані
ПЗ		Нема посилань на джерела

Додаток Б

Фрагмент лістингу програми

Б1. Лістинг коду аудіоплеєру, MusicPlayer

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

public class MusicPlayer : MonoBehaviour
{
    private AudioSource audioSource;
    private List<AudioClip> audioClips = new List<AudioClip>();
    private int currentClipIndex = 0;
    private bool isPaused = false;

    [SerializeField] private Text songTitleText;
    [SerializeField] private Button previousButton;
    [SerializeField] private Button playPauseButton;
    [SerializeField] private Button nextButton;

    private void Start()
    {
        audioSource = GetComponent<AudioSource>();
        LoadAudioClips();
            isPaused = true;
    }

    private void LoadAudioClips()
    {
        DirectoryInfo dir = new DirectoryInfo(Application.dataPath + "/Music");
        FileInfo[] files = dir.GetFiles("*.mp3");

        foreach (FileInfo file in files)
        {
            StartCoroutine(LoadAudioClip(file.FullName));
        }
    }

    private IEnumerator LoadAudioClip(string path)
    {
        WWW www = new WWW("file://" + path);
        yield return www;
```

```
    AudioClip audioClip = www.GetAudioClip();
    audioClip.name = Path.GetFileNameWithoutExtension(path);

    audioClips.Add(audioClip);
}

private void Update()
{
    if (audioClips.Count > 0 && !isPaused && !audioSource.isPlaying)
    {
        currentClipIndex = (currentClipIndex + 1) % audioClips.Count;
        audioSource.clip = audioClips[currentClipIndex];
            audioSource.Play();
        songTitleText.text = audioSource.clip.name;
    }
}

public void Previous()
{
    if (audioClips.Count > 0)
    {
        currentClipIndex = (currentClipIndex - 1 + audioClips.Count) % audioClips.Count;
        audioSource.clip = audioClips[currentClipIndex];
            isPaused = false;
        audioSource.Play();
        songTitleText.text = audioSource.clip.name;
    }
}

public void PlayPause()
{
    if (audioClips.Count > 0)
    {
        if (isPaused)
        {
            isPaused = false;
            audioSource.UnPause();
            playPauseButton.GetComponentInChildren<Text>().text = "||";
        }
        else
        {
            isPaused = true;
            audioSource.Pause();
            playPauseButton.GetComponentInChildren<Text>().text = "||";
        }
    }
}
```

```
    public void Next()
    {
        if (audioClips.Count > 0)
        {
            currentClipIndex = (currentClipIndex + 1) % audioClips.Count;
            audioSource.clip = audioClips[currentClipIndex];
                isPaused = false;
            audioSource.Play();
            songTitleText.text = audioSource.clip.name;
        }
    }
}
```

Б2. Лістинг коду створення шляху, PathController

```
using UnityEngine;

public class PathController : MonoBehaviour
{
    [SerializeField] private Transform[] points;
    [SerializeField] private GameObject[] carPrefabs;
    [SerializeField] private float spawnInterval = 1f;

    private void Start()
    {
        SpawnCars();
    }

    private void SpawnCars()
    {
        foreach (GameObject carPrefab in carPrefabs)
        {
            int randomIndex = Random.Range(0, points.Length);
            Transform spawnPoint = points[randomIndex];

            GameObject newCar = Instantiate(carPrefab, spawnPoint.position, Quaternion.identity);
            Car carScript = newCar.GetComponent<Car>();
            carScript.SetPathPoints(points);
            carScript.StartMoving();
        }
    }
}
```

Б3. Лістинг коду переміщення об’єкта, Car

```
using System.Collections;
using UnityEngine;

public class Car : MonoBehaviour
{
    private Transform[] pathPoints;
    private int currentPointIndex = 0;
    private bool isMoving = false;

    [SerializeField] private float speed = 10f;
    [SerializeField] private float rotationSpeed = 5f;

    public void SetPathPoints(Transform[] points)
    {
        pathPoints = points;
    }

    public void StartMoving()
    {
        if (pathPoints.Length > 0)
        {
            float closestDistance = Mathf.Infinity;
            int closestIndex = 0;
            for (int i = 0; i < pathPoints.Length; i++)
            {
                float distance = Vector3.Distance(transform.position, pathPoints[i].position);
                if (distance < closestDistance)
                {
                    closestDistance = distance;
                    closestIndex = i;
                }
            }

            currentPointIndex = closestIndex;
            StartCoroutine(MoveAlongPath());
        }
    }

    private IEnumerator MoveAlongPath()
    {
        isMoving = true;

        while (isMoving)
        {
            Transform targetPoint = pathPoints[currentPointIndex];
            Vector3 targetDirection = targetPoint.position - transform.position;
            Quaternion targetRotation = Quaternion.LookRotation(targetDirection);
```

```
        transform.rotation = Quaternion.RotateTowards(transform.rotation, targetRotation, rotationSpeed * Time.deltaTime);

        transform.position = Vector3.MoveTowards(transform.position, targetPoint.position, speed * Time.deltaTime);

        if (transform.position == targetPoint.position)
        {
            currentPointIndex++;
            if (currentPointIndex >= pathPoints.Length)
            {
                currentPointIndex = 0;
            }
        }

        yield return null;
    }
  }
}
```

Додаток В

Матеріали презентації

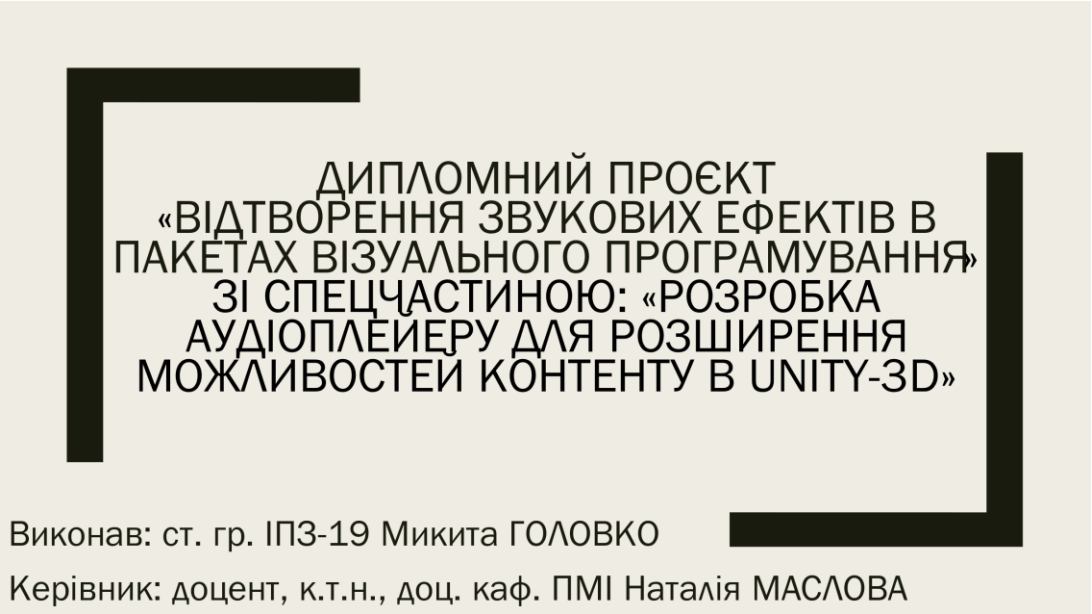

Рисунок В.1 – Титульний слайд

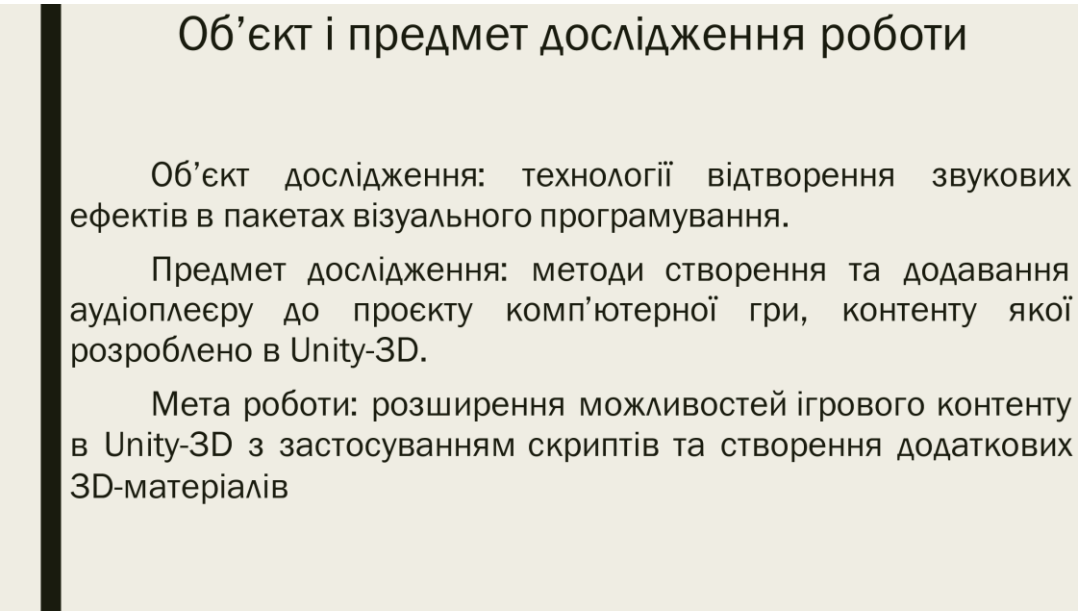

Рисунок В.2 – Об’єкт і предмет дослідження роботи

Рисунок В.3 – Візуальне програмування

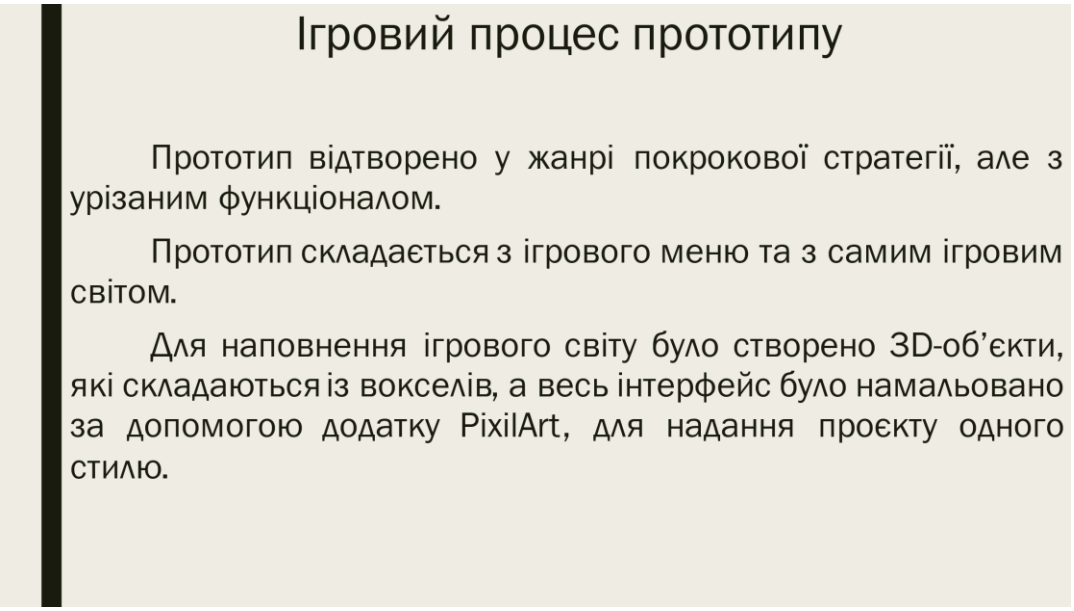

Рисунок В.4 – Ігровий процес прототипу

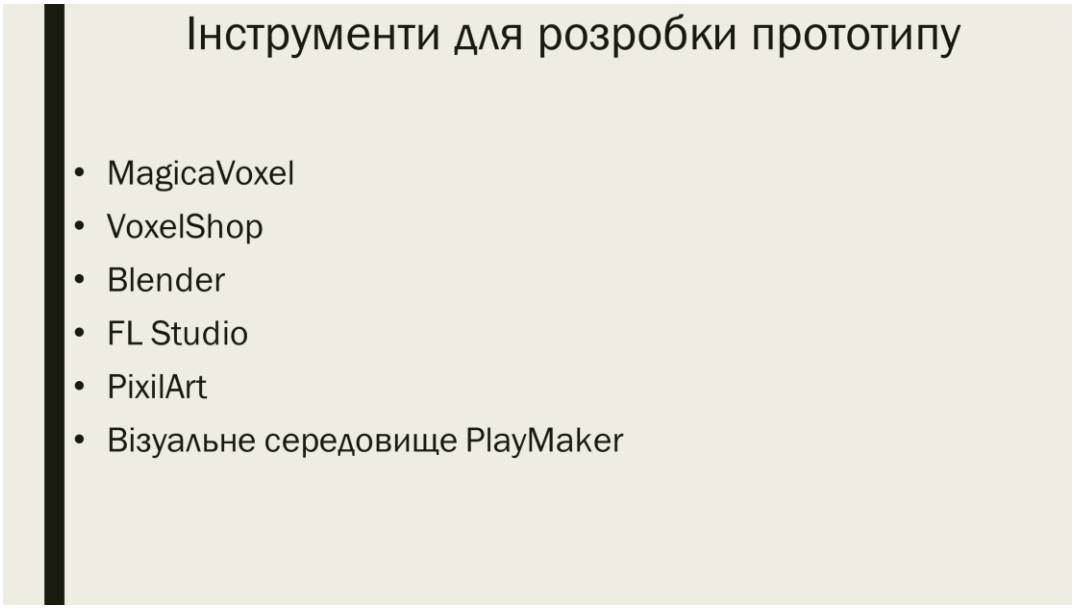

Рисунок В.5 – Інструменти для розробки прототипу

Рисунок В.6 – Діаграма використання

Рисунок В.7 – Діаграми станів

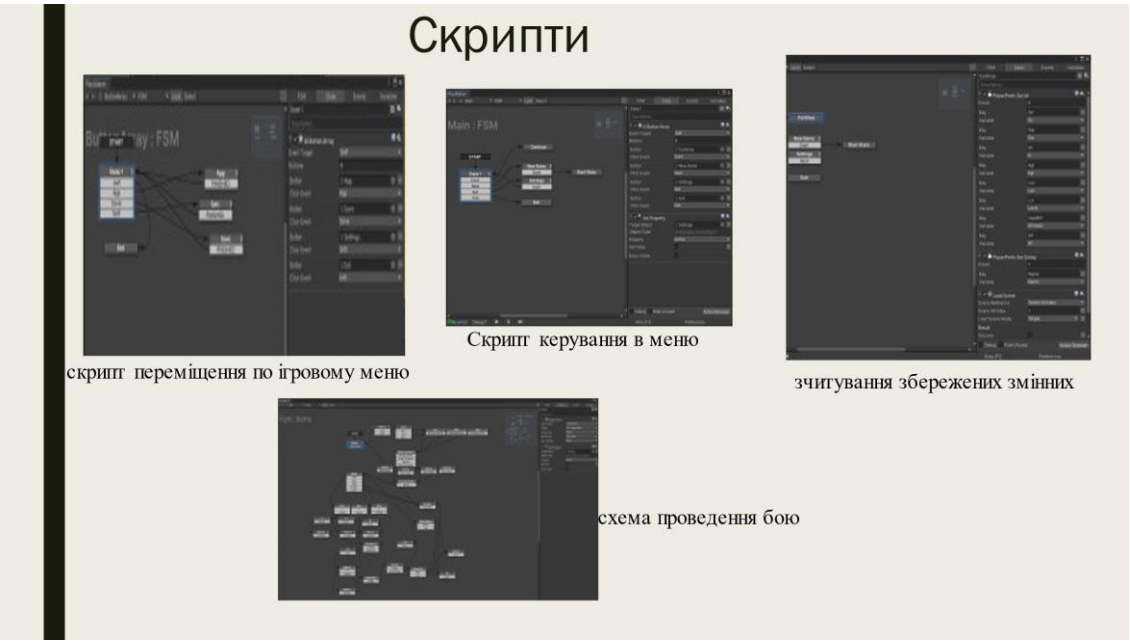

Рисунок В.8 – Скрипти

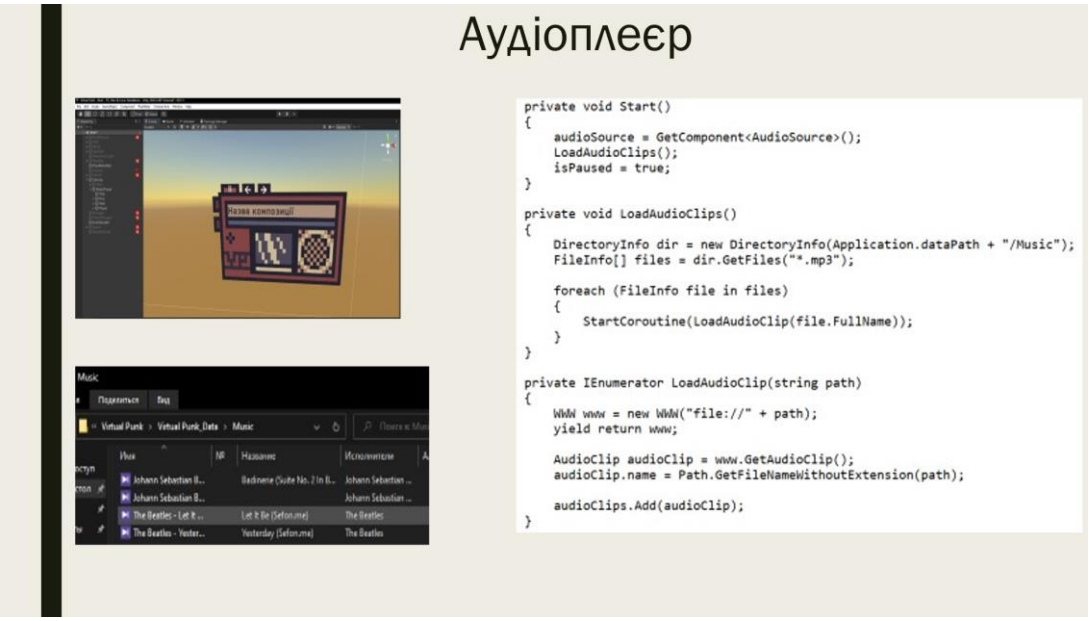

Рисунок В.9 – Аудіоплеєр

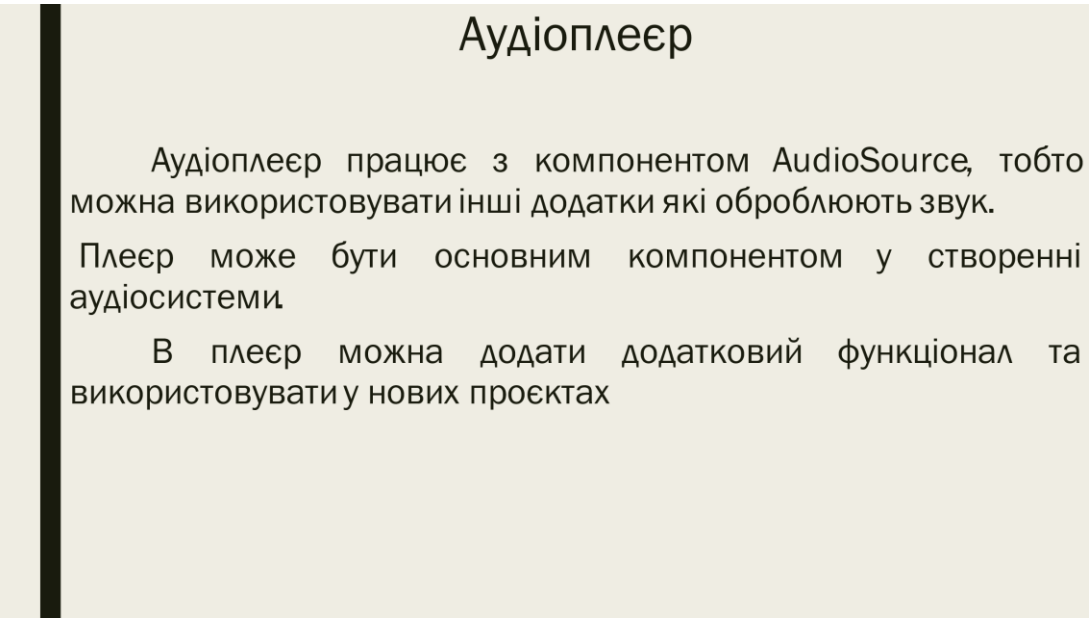

Рисунок В.10 – Аудіоплеєр

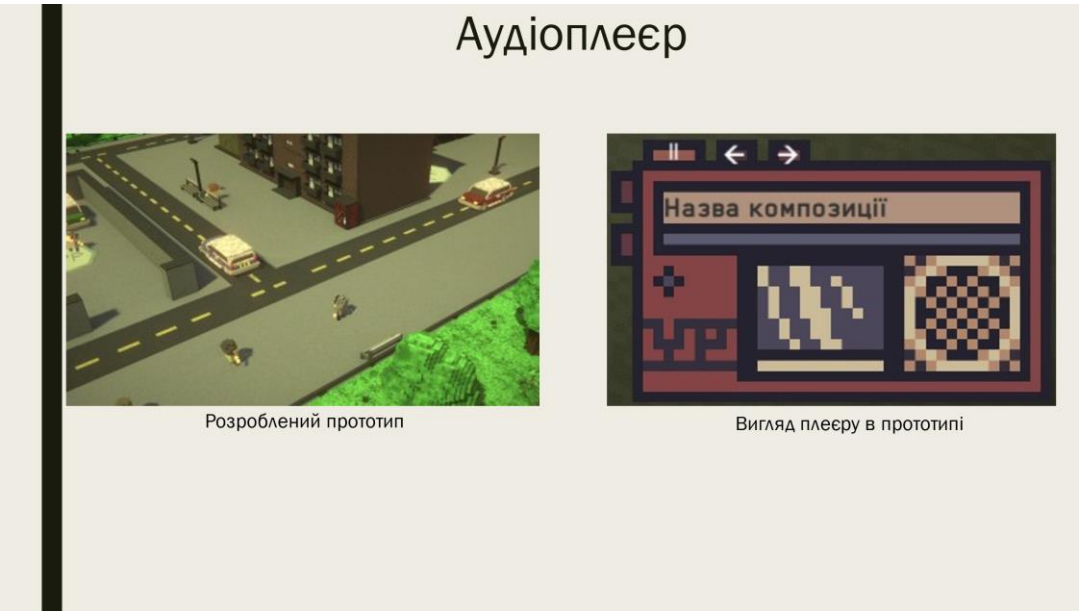

Рисунок В.11 – Аудіоплеєр

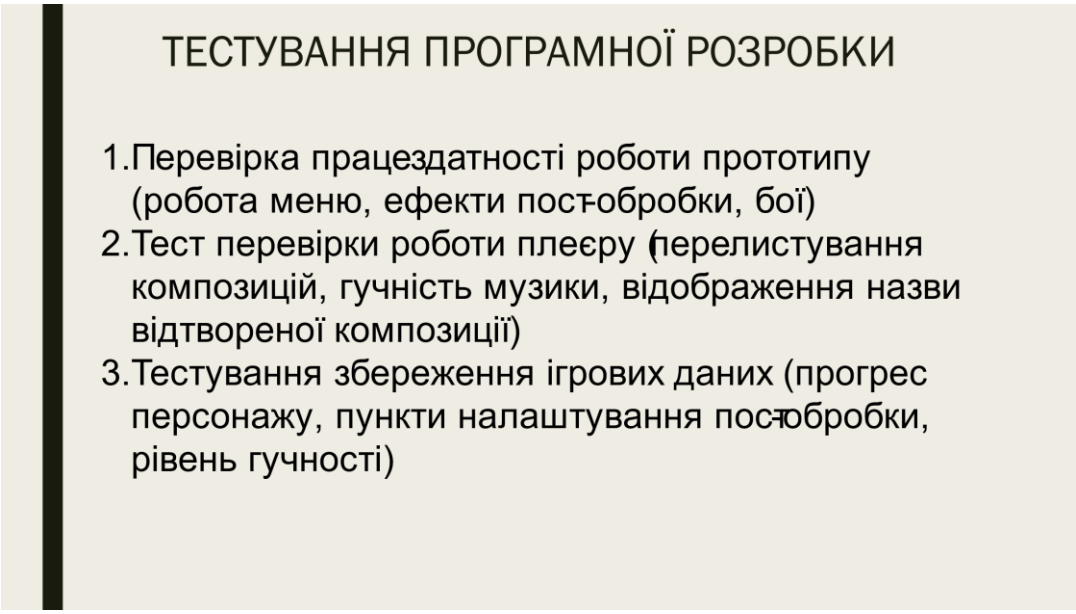

Рисунок В.12 – Тестування програмної розробки

Додаток Г

Встановлення плеєру

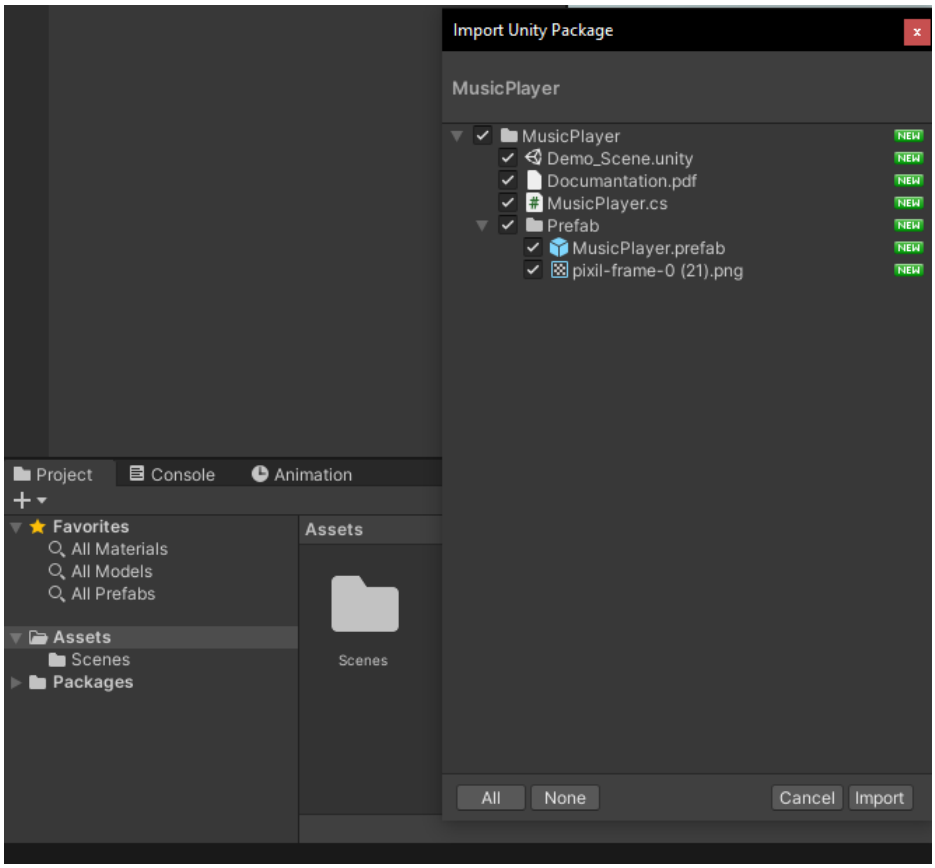

Рисунок Г.1 – Встановлення доповнення

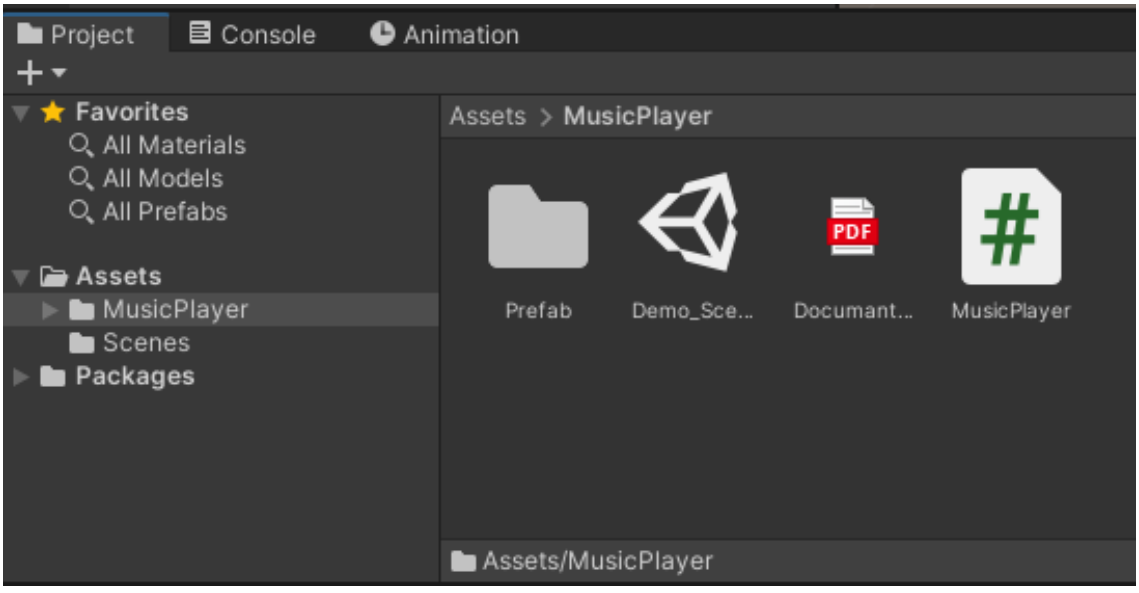

Рисунок Г.2 – Зміст імпортованого доповнення

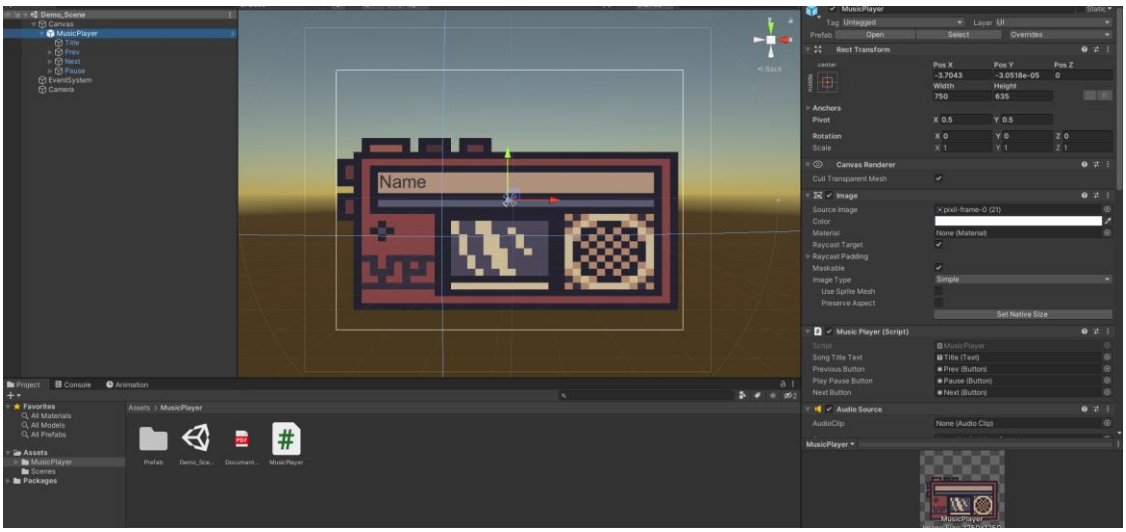

Рисунок Г.3 – Сцена с налаштованим плеєром

Документація для скрипту MusicPlayer

Вступ

Скрипт **MusicPlayer** є компонентом Unity, який дозволяє відтворювати музику у вашій грі або додатку. Він здатний завантажувати аудіофайли з папки "Music" і програвати їх послідовно. Крім того, він надає кнопки для керування відтворенням (попередній трек, пауза/відтворення, наступний трек) та відображає назву поточного треку.

Використання

Для використання скрипту **MusicPlayer** потрібно виконати наступні кроки:

1. Додайте компонент **MusicPlayer** до будь-якого об'єкта у вашій сцені.
2. Створіть папку з назвою "Music" у кореневій папці вашого проекту Unity.
3. Розмістіть аудіофайли формату MP3 у папці "Music".
4. Додайте на сцену необхідні компоненти для відображення назви треку та керування відтворенням:
    - Додайте текстове поле (**Text**) і призначте його для поля **songTitleText** у скрипті **MusicPlayer**. Це поле буде використовуватись для відображення назви поточного треку.
    - Додайте три кнопки (**Button**): одну для попереднього треку, одну для паузи/відтворення та одну для наступного треку. Призначте ці кнопки відповідно для полів **previousButton**, **playPauseButton** і **nextButton** у скрипті **MusicPlayer**.
5. Запустіть вашу гру або додаток. Тепер ви зможете керувати відтворенням музики за допомогою кнопок на сцені.

Залежності

Скрипт **MusicPlayer** використовує наступні компоненти Unity:

- **AudioSource**: відповідає за відтворення аудіофайлів.
- **Text**: використовується для відображення назви поточного треку.
- **Button**: використовується для керування відтворенням музики.

Documentation for MusicPlayer Script

Introduction

The **MusicPlayer** script is a Unity component that allows you to play music in your game or application. It can load audio clips from the "Music" folder and play them sequentially. Additionally, it provides buttons for playback control (previous track, play/pause, next track) and displays the name of the current track.

Usage

To use the **MusicPlayer** script, follow these steps:

1. Add the **MusicPlayer** component to any GameObject in your scene.
2. Create a folder named "Music" in the root directory of your Unity project.
3. Place MP3 audio files inside the "Music" folder.
4. Add the necessary components to your scene for displaying the track name and controlling playback:
    - Add a Text component and assign it to the **songTitleText** field in the **MusicPlayer** script. This field will be used to display the name of the current track.
    - Add three buttons: one for the previous track, one for play/pause, and one for the next track. Assign these buttons to the **previousButton**, **playPauseButton**, and **nextButton** fields respectively in the **MusicPlayer** script.
5. Run your game or application. Now you can control music playback using the buttons in your scene.

Dependencies

The **MusicPlayer** script relies on the following Unity components:

- **AudioSource**: Responsible for playing audio clips.
- **Text**: Used to display the current track name.
- **Button**: Used for controlling music playback.

Рисунок Г.4 – Створена документація