МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Кваліфікаційна робота бакалавра

ІНФОРМАЦІЙНЕ І ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ІГРОВОГО ДОДАТКУ З ВИКОРИСТАННЯМ ІНТЕГРОВАНОГО СЕРЕДОВИЩА UNITY 3D

Здобувач освіти гр. ІН-81 Богдан КІПТЕНКО

Науковий керівник,
кандидат ф.-м. наук, доцент Сергій ШАПОВАЛОВ

Завідувач кафедри
доктор технічних наук,професор. Анатолій ДОВБИШ

СУМИ 2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютерних наук

Затверджую_____
Зав. кафедри Довбиш А.С.
"_____" _____2022 р.

ЗАВДАННЯ

до кваліфікаційної роботи бакалавра

Студента 4-го курсу, групи ІН-81 спеціальності 122 - Комп'ютерні науки, денної форми навчання Кіптенка Б. А.

Тема: Інформаційне і програмне забезпечення ігрового додатку з використанням інтегрованого середовища Unity 3D

Затверджена наказом по СумДУ
№_____ від _____2022 р.

Зміст пояснювальної записки: 1) інформаційний огляд існуючих ігрових додатків; 2) вибір методу рішення; 3) інформаційне та програмне забезпечення ігрового додатку; 4.) висновки

Дата видачі завдання "_____"_____2022 р.
Керівник випускної роботи _____ Шаповалов С.П.
Завдання прийняв до виконання _____ Кіптенко Б.А.

РЕФЕРАТ

Записка: 57 стор., 33 рис., 2 табл., 10 джерел інформації, 8 додатків.

Об’єкт дослідження — тривимірне проектування ігрових додатків

Мета роботи — розробка інформаційного та програмного забезпечення ігрового додатку

Методи дослідження — тривимірне проектування, середовище FL Studio 2020, інтегроване середовище Unity

Результати — створено інформаційне та програмне забезпечення ігрового додатку жанру “adventure” (демо-версія). Функціональна основа побудови тривимірних об’єктів та ігрового середовища спроектована за допомогою програмного засобу Blender. Середовищем створення аудіо-оформлення є музичний редактор FL Studio 2020. Основним інструментарієм створення додатку є інтегроване середовище Unity 2020.

СЕРЕДОВИЩЕ РОЗРОБКИ ІГРОВИХ ДОДАТКІВ, UNITY 2020, BLENDER, FL STUDIO 2020, ІГРОВИЙ ОБ’ЄКТ, СЦЕНАРІЙ, ПОЛІГОНАЛЬНЕ МОДЕЛЮВАННЯ, PREFAB.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

MOBA – multiplayer online battle arena

MMORPG – massively multiplayer role playing game

RPG – role playing game

UML – unified modeling language

VST – virtual studio technology

FBX – filmbox

ВСТУП

З початку 21-го століття ігрова індустрія зайняла та укріпила своє місце у світовій економіці. З кожним роком вона розвивається та набирає популярності не лише серед споживачів, а й серед економістів. Інтерес до фінансування та монетизації доходу з ігрового ринку зростає. [1]

Існує величезна кількість жанрів відеоігор, починаючи з пазлів, квестів, платформерів та стратегій і закінчуючи масштабними багатокористувацькими онлайн проектами. Тенденції ігрового ринку час від часу змінюються, і певні жанри стають більш популярними. Великий успіх певного проекту часто спонукає інших створювати схожі свої ігрові продукти, що створили б конкуренцію та принесли прибуток завдяки популярності даного типу ігор. Не зважаючи на це у кожного жанру є свої прихильники, тому кожен проект здатен знайти свою аудиторію.

Розробка відеоігор це досить тривалий процес що потребує великої кількості навичок: програмування, створення візуального, звукового оформлення, дизайну. Для багатокористувацьких онлайн проектів потрібні знання мережі та роботи з серверами. Такий обсяг роботи та знань потребує цілої команди розробників. Найбільші сучасні компанії можуть виділяти на розробку проектні команди більше 500 співробітників. Проте розробкою відеоігор може займатися і одна людина. Все залежить від об’єму роботи та часу що розробник готовий/спроможний виділити на реалізацію проекту.

Сучасним програмним засобом, що поєднує в собі безліч інструментів для створення ігрових додатків є так званий «game engine» – середа розробки ігор. Він є центральним програмним компонентом комп’ютерних, мобільних відеоігор та інших інтерактивних додатків з обробкою графіки в реальному часі. Він забезпечує користувача основними технологіями та інструментами, спрощує розробку і часто дає можливість переносити створені додатки для різних платформ, таких як ігрові консолі, настільні та мобільні операційні системи.

Метою роботи є створення ігрового додатку за допомогою інтегрованого середовища Unity 3D. Так як проект поєднує у собі використання тривимірної графіки та елементів аудіо, додатково використовуються пакет для створення тривимірної комп'ютерної графіки Blender, що включає в себе засоби моделювання, анімації, рендерингу та постобробки, а також музичний редактор FL Studio.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД ІСНУЮЧИХ ІГРОВИХ ДОДАТКІВ

1.1 Перші ігрові додатки жанру “adventure”.

Пригодницький жанр (англ. “Adventure”) бере початок з середини 20-го століття. Першою грою жанру є текстова інтерактивна гра Colossal Cave Adventure розроблена та випущена у 1976 році Уілом Кроутером (англ. Willie Crowther) (див. рис.1.1). [2]

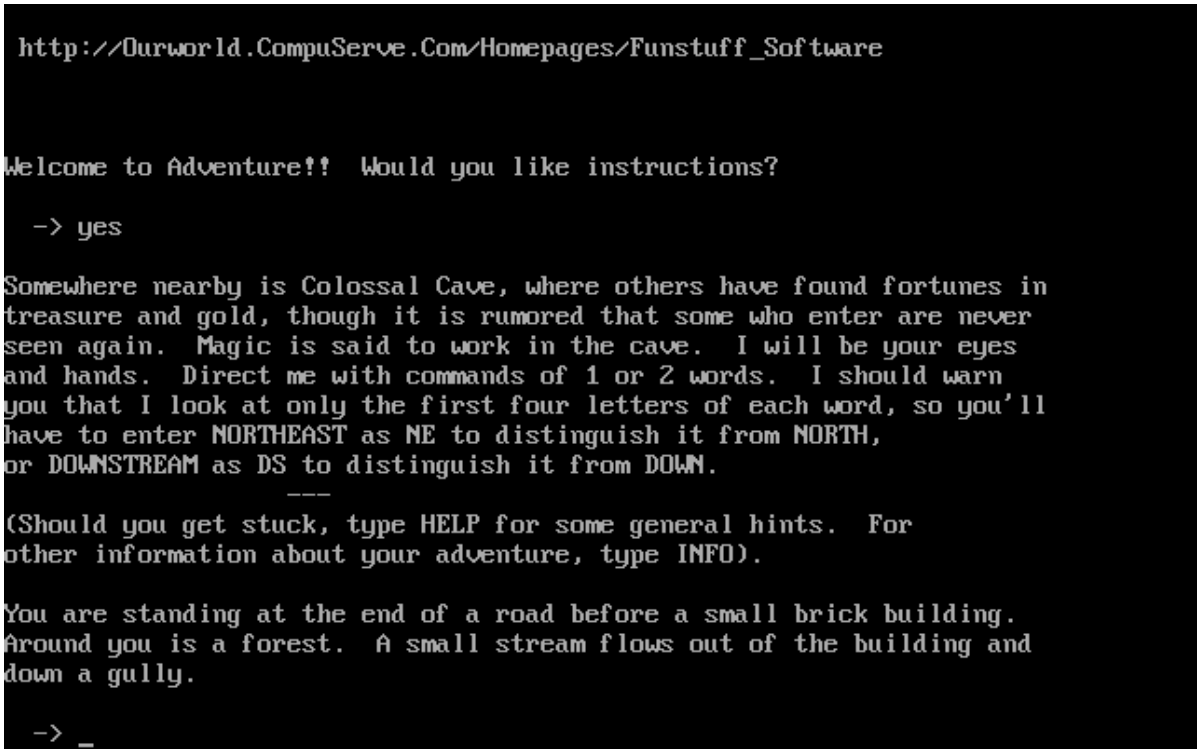

Рисунок 1.1 — Знімок екрана Colossal Cave Adventure

У 1980-х роках з’явилися візуальні пригоди, що використовують векторну графіку для створення візуального оформлення, хоча все ще вимагають від користувача введення текстових команд для взаємодії з ігровим середовищем. Першою візуальною пригодою вважається гра Mystery House від Sierra On-Line (див. рис.1.2).

Рисунок 1.2 — Знімок екрана Mystery House

Сьогодні жанр пригодницьких ігор поєднує в собі як 2D так і 3D проекти, багато різних тематик та особливостей. В залежності від використовуваних елементів жанр можна поділити на кілька різновидів:

- інтерактивні текстові квести, в яких гравець отримує потрібну інформацію через текстовий опис, використовуючи клавіатуру або меню для навігації;
- point-and-click квести, в яких за допомогою комп’ютерної миші треба взаємодіяти з елементами на екрані;
- пазли, в яких у ході оповідання необхідно розв’язувати різні інтелектуальні завдання;
- пригодницький екшн, що поєднує елементи обох жанрів, пропонує гравцю долати перешкоди як інтелектуального роду, так і фізичного, наприклад, битви, випробування витривалості чи швидкість реакції;
- візуальна новела, в якій історія екранізована статичними або анімованими зображеннями та текстовим описом.

1.2 Огляд «A Story About My Uncle».

A Story About My Uncle – пригодницька гра від першої особи про хлопчика, який шукає свого дядька у дивовижному чарівному світі. Гра розроблена шведською студією Gone North Games і випущена Coffee Stain Studios 28 травня 2014 року на платформах Windows, Linux та macOS.

Пересування за допомогою гака, що дозволяє притягуватися до поверхонь, є важливою частиною основного ігрового процесу (див. рис. 1.3). Дослідження світу, допомагає глибше вникнути в історію. Гра отримала переважно позитивні відгуки як від критиків так і від звичайних користувачів. Основними перевагами є приємне управління, дизайн ігрового середовища та атмосфера [3].

Рисунок 1.3 — Знімок екрана A Story About My Uncle

1.3 Огляд «The Stanley Parable».

The Stanley Parable – пригодницька гра від першої особи, випущена у 2013 році Дейві Вреденом (англ. Davey Wreden) у співпраці з Вільямом П'ю (англ. William Pugh). Першою версією гри є випущена у 2011 році модифікація для гри Half Life 2, яку згодом переробили у повноцінний проект.

The Stanley Parable це експеримент який мав на меті створити нову форму оповідання, унікальну для відеоігор. Більшість ігор обмежують можливості гравця, намагаючись вести його по одному заданому шляху. The Stanley Parable дає гравцю вибір, що так чи інакше змінює історію, дає багато різноманітних шляхів та кінцівок.

Гра починається в стандартній кабінці офісу (див. рис. 1.4). Ви – Стенлі, слухняний працівник, який проводить свої дні друкуючи команди на старому комп’ютері. Одного разу всі його колеги зникають, залишаючи Стенлі на самоті досліджувати офіс. Історію веде оповідач, який коментує те що ми – Стенлі зробили, і те що ми “повинні зробити” далі. Гравець може іти по вказаному шляху, проте також може ігнорувати вказівки та обирати інші шляхи.[4]

Рисунок 1.4 — Знімок екрана The Stanley Parable

1.4 Огляд «We Were Here Together».

We Were Here Together – кооперативний пригодницький пазл від першої особи, розроблений та випущений 10 жовтня 2019 року студією Total Mayhem Games.

Слідуючи від дослідницької станції за сигнальним спалахом, гравці опиняються у середньовічному замку, відокремлені один від одного. Використовуючи рацію вони повинні працювати разом, розв’язуючи головоломки удвох, аби знайти вихід (див. рис.1.5).

Рисунок 1.5 — Знімок екрана We Were Here Together

Кожна гра серії отримала позитивні відгуки від критиків та гравців. We Were Here Together номінована як краща кооперативна гра на Indie of the Year Awards 2019.

1.4 Постановка задачі.

Результатом проведеного огляду існуючих ігрових додатків та аналізу їх інформаційного забезпечення є постановка задач для дослідження в бакалаврській роботі – створити програмне та інформаційне забезпечення ігрового додатку в тривимірному середовищі. Для реалізації поставленого завдання необхідно реалізувати наступні задачі:

1. виконати огляд доступних інструментів для розробки та обрати оптимальні;
2. визначити та розробити основні елементи ігрового процесу;
3. виконати проектування та побудову ігрового середовища;
4. розробити аудіо оформлення;
5. зробити відповідні висновки по роботі.

2 ВИБІР МЕТОДУ РІШЕННЯ

2.1 Огляд та порівняння середовищ розробки ігрових додатків

2.1.1 Огляд Unreal Engine

Unreal Engine – високотехнологічний інструмент для роботи з 3D, що використовується не лише в ігровій індустрії а й в кінематографі. Остання версія, Unreal Engine 5 вперше продемонстрована у 2020 році, дозволяє створювати майже фотореалістичну тривимірну графіку (див. рис. 2.1).

Рисунок 2.1 – Демонстрація Unreal Engine 5

Unreal Editor – інтегроване середовище розробки, що входить в склад Unreal Engine (див. рис.2.2). Для створення сценаріїв використовується технологія «visual scripting» (візуалізоване створення сценаріїв з використанням блоків і з'єднань), що довгий час була особливістю Unreal Engine, та власна мова – UnrealScript, яку замінили на C++ починаючи з версії Unreal Engine 4.

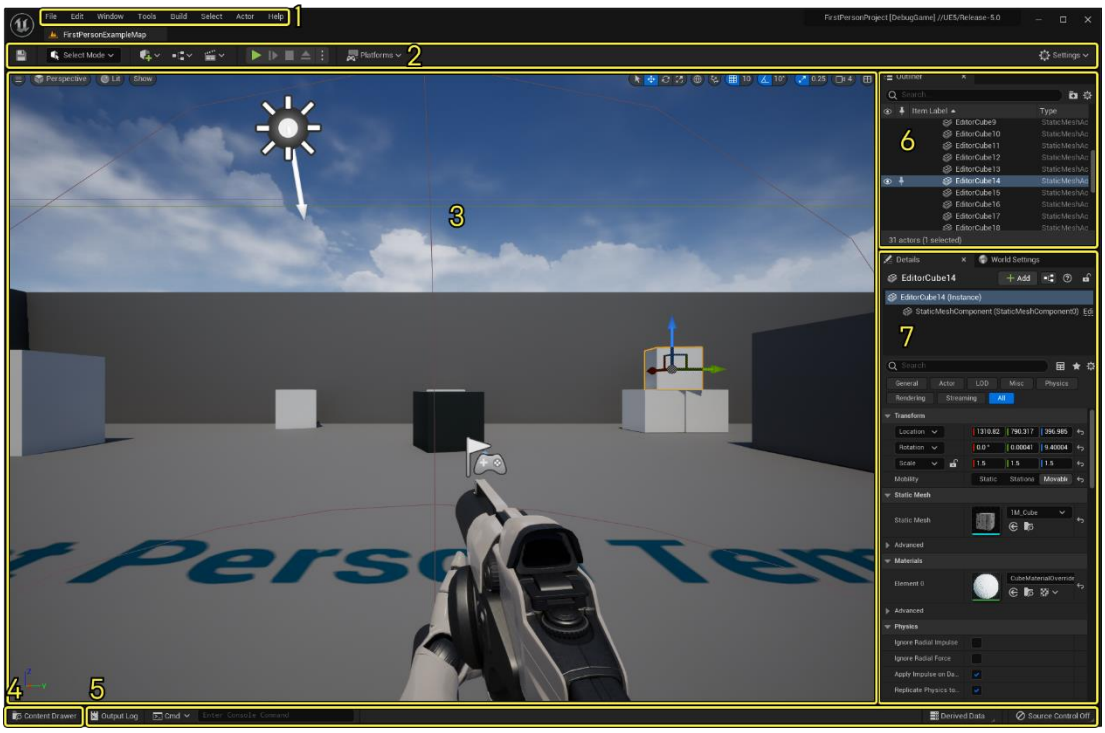

Рисунок 2.2 – Основний інтерфейс Unreal Editor

Unreal Engine 5 дозволяє створювати проекти для ПК з Windows, PlayStation 5, PlayStation 4, Xbox Series X, Xbox Series S, Xbox One, Nintendo Switch, Google Stadia, macOS, iOS, Android, ARKit, ARCore, OpenXR, SteamVR, Oculus , Linux і SteamDeck. Сам редактор Unreal Editor доступний на Windows, macOS та Linux.

Unreal Engine можна безкоштовно використовувати для навчання та розробки персональних проектів. Він також дає змогу розповсюджувати некомерційні та комерційні проекти не сплачуючи комісії Epic Games – компанії-власнику Unreal Engine, якщо дані проекти не приносять доходу або дохід не досягає порогового значення. Комісія в розмірі 5% сплачується лише в тому випадку, якщо ви розповсюджуєте готовий продукт, який містить код Unreal Engine (наприклад, гру), і загальний поточний дохід від нього перевищує 1 мільйон доларів США.

2.1.2 Огляд Godot

Godot – один з найпростіших для освоєння інструментів розробки ігор, що підтримує створення як 2D так і 3D проектів. Хоч в ньому присутні певні недоліки оптимізації для тривимірних проектів, Godot володіє досить простим інтерфейсом та, на відміну від інших середовищ розробки, розповсюджується по системі «open source» (див. рис. 2.3). Широко підтримується спільнотою, що надає допомогу в розвитку даного інструмента, створенні нового функціоналу, виправленні помилок, ведення документації та наданні підтримки новим мовам програмування.

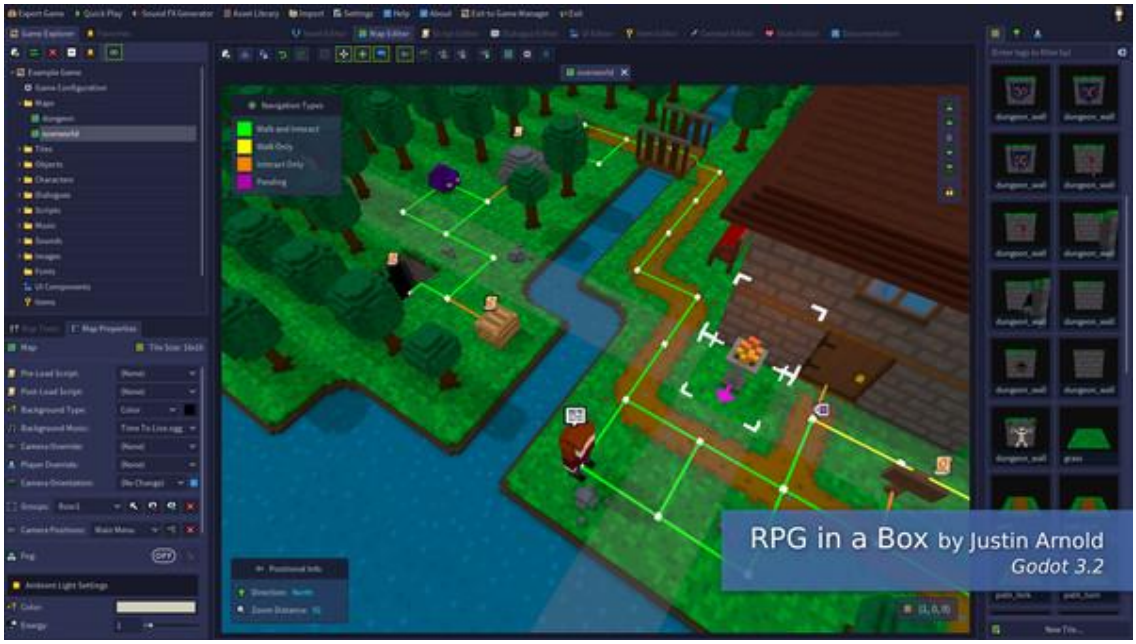

Рисунок 2.3 – Основний інтерфейс Godot 3.2

Godot включає наступні особливості:

- повністю безкоштовний;
- власна мова сценаріїв GDScript (синтаксисом подібна до Python);
- повна підтримка C# 8.0 за допомогою Mono;
- повна підтримка C++ ;
- технологія «visual scripting»;

- надана спільнотою підтримка додаткових мов (Rust, Nim, D та ін.);
- вбудований редактор коду;
- вбудовані інструменти для створення анімації;
- інтегрована документація;
- ігри можна експортувати на настільні платформи (Linux, macOS, Windows), а також мобільні (Android, iOS), веб-платформи (HTML5) та консолі (Nintendo Switch, PS4 та Xbox One – через сторонніх видавців).

2.1.3 Огляд Unity

Unity – найпопулярніше сучасне середовище для створення 2D і 3D-ігор (див. рис. 2.4). Unity підтримує створення сценаріїв мовою C#, включає вбудований редактор коду, проте також підтримує інші редактори, як Visual Studio.

Наразі розробляється нова версія Unity з імплементацією «visual scripting», що дозволяє створювати ігрову механіку чи логіку взаємодії за допомогою візуальної графової системи замість традиційних рядків коду.

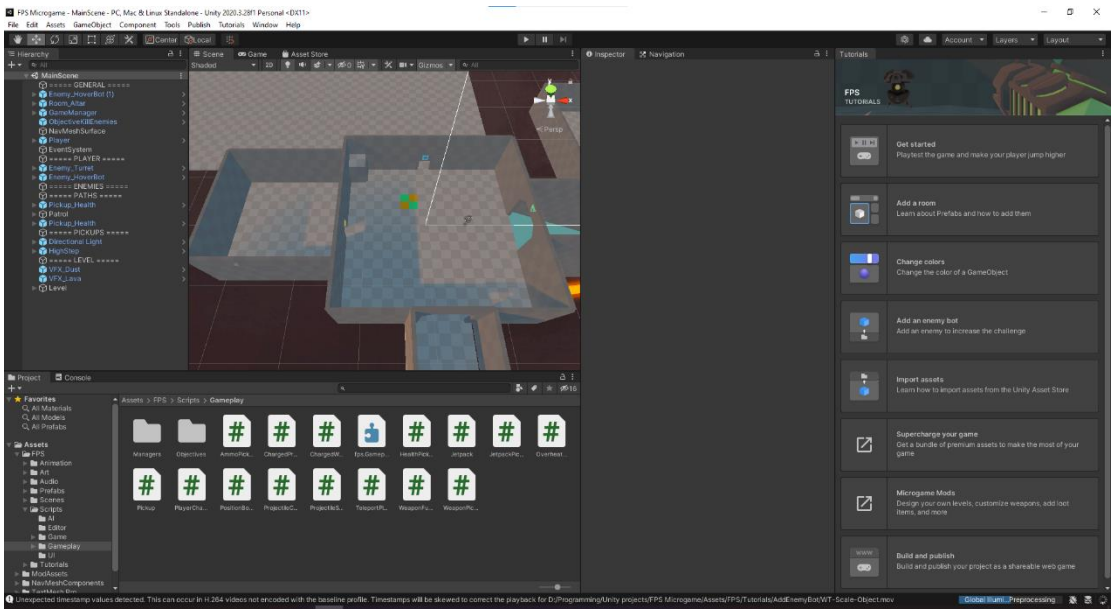

Рисунок 2.4 – Основний інтерфейс Unity 2020

Unity Hub – програмний засіб, що дозволяє управляти встановленими версіями редактора Unity, залежностями та додатковими розширеннями, переглядати та створювати проекти, переглядати та встановлювати навчальні проекти, ділитися своїми проектами та переглядати роботи інших користувачів (див. рис. 2.5).

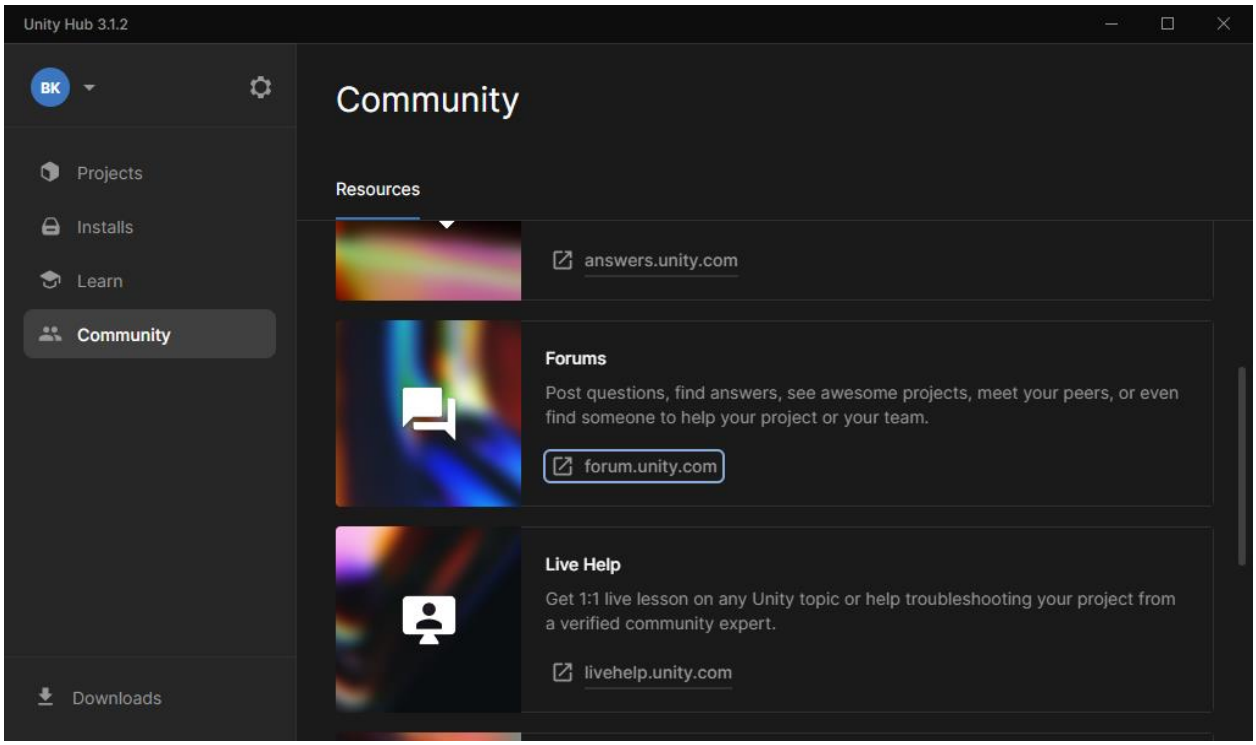

Рисунок 2.5 – Основний інтерфейс Unity Hub 3.1.2

Документація Unity доволі якісна, при цьому присутня величезна кількість як офіційних, так і сторонніх навчальних матеріалів. Навчальна програма Unity Learn – інтерактивний навчальний курс, під час якого користувач може вивчити основи програмування та роботи з середовищем Unity (див. рис. 2.6). Даний курс є повністю безкоштовним та дозволяє новачкам вивчити все необхідне для початку роботи з Unity та почати власноруч розробляти ігри.

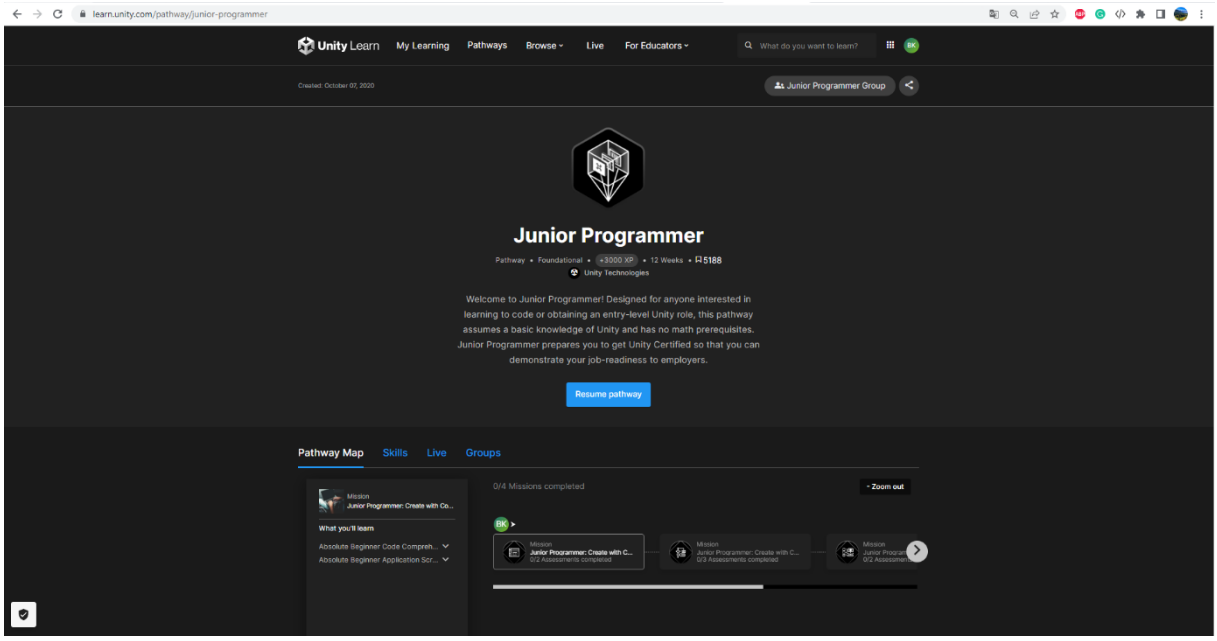

Рисунок 2.6 – Інтерактивний курс Unity Learn

2.2 Середовище створення тривимірної графіки Blender

Blender — це безкоштовний пакет для створення 3D з відкритим вихідним кодом. Він володіє всім необхідним інструментарієм для роботи з тривимірною графікою — моделювання, такелаж, анімацію, симуляцію, візуалізацію, композицію та відстеження руху, а також редагування відео. Blender дозволяє використовувати свій API для створення сценаріїв Python для налаштування програми та написання спеціалізованих інструментів. Часто такі інструменти додають у нові версії програми (див. рис. 2.7).

Blender підтримує експорт моделей різних форматів, та безпосередньо підтримується середовищем Unity. Unity може самостійно імпортувати файли Blender за допомогою експортера Blender FBX. Достатньо зберегти файл .blend в папці Assets вашого проекту. Коли ви повертаєтеся до Unity, файл імпортується автоматично і відображатиметься в браузері проекту.

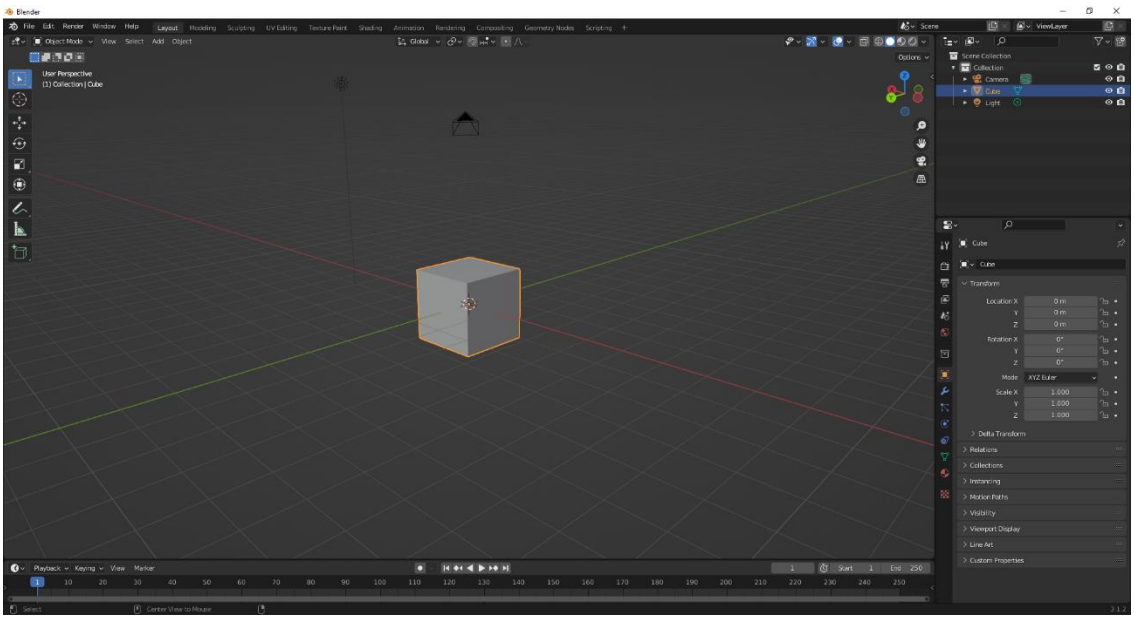

Рисунок 2.7 – Основний інтерфейс Blender 3.1

2.3 Музичний редактор FL Studio

FL Studio – це цифрова звукова робоча станція (англ. Digital Audio Workstation), тип програмного забезпечення, призначеного для редагування звуку, але зазвичай відомого як програмне середовище для створення музики (див. рис. 2.8).

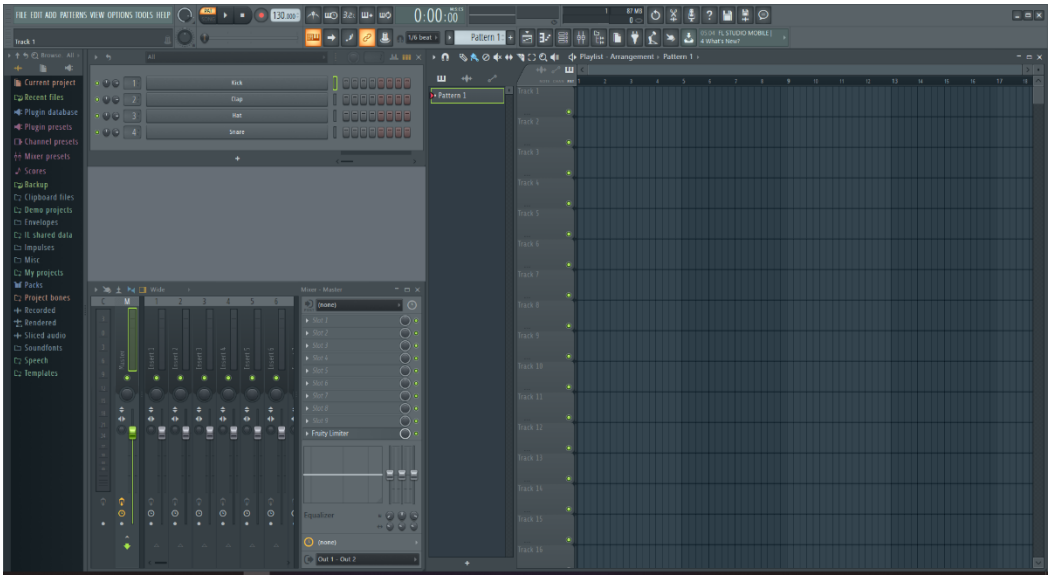

Рисунок 2.8 – Основний інтерфейс FL Studio 2020

Всі використовувані елементи проекту: послідовності нот (англ. pattern), записи аудіо та елементи автоматизації (динамічна зміна гучності, темпу, накладання ефектів і т.д.) – зберігаються та упорядковуються у браузері проекту. Елементи можна розмістити будь-де на доріжці та накладати їх один на одного.

Piano roll – цифровий інструмент, що дозволяє створювати нотні послідовності та відтворювати музику. Свою назву він отримав через візуальну схожість та принцип роботи схожий на перфоровану стрічку для механічного піаніно. Piano roll у FL Studio використовується для надсилання даних про ноти та елементи автоматизації до віртуальних інструментів.

FL Studio підтримує всі стандарти VST (Virtual Studio Technology) – формат залежних від середи виконання плагінів, які підключаються до звукових редакторів, секвенсорів, цифрових звукових робочих станцій. VST надає доступ до широкого діапазону сторонніх плагінів, доступних на даний момент. Також є можливість використовувати саму FL Studio як плагін VST на іншому хості VST.

2.4 Аналіз популярності жанрів відеоігор

Проаналізуємо статистичні дані про актуальність жанру «adventure» за останні роки.

За даними онлайн опитування statista.com, на момент третьої чверті 2021 року, «adventure» є одним з найпопулярніших жанрів відеоігор серед користувачів [5]. Конкретніше, “action adventure”, “puzzle platform” та “action platform”, що мають в середньому досить високий відсоток голосів, особливо серед користувачів віком 16-34 роки. Найпопулярнішим є жанр “shooter”, частково через те, що на сьогодні він часто орієнтований на багатокористувацькі сесіонні онлайн проекти. Результати опитування наведені у таблиці 2.1.

Таблиця 2.1 – Найпопулярніші жанри відеоігор серед інтернет користувачів (3 чверть 2021 р.)

Жанр	**Вікова категорія**				
	16-24 роки	**25-34 роки**	**35-44 роки**	**45-54 роки**	**55-64 роки**
Shooter	60%	57%	48%	35%	21%
Action adventure	56%	54%	46%	35%	21%
Simulator	39%	38%	32%	34%	15%
Racing	38%	40%	35%	26%	15%
MOBA	36%	36%	28%	20%	-
Battle royale	36%	-	-	-	-
Sports	34%	38%	33%	23%	15%
Strategy	34%	36%	31%	22%	15%
Puzzle platform	33%	36%	34%	27%	23%
Action platform	31%	33%	28%	20%	13%
Online board games	-	-	-	-	14%
Fighting	-	34%	28%	19%	-
Free-to-play casino	-	-	-	-	13%

За даними зі статті Анастасії Дейна на новинному онлайн ресурсі «The Village», жанр “puzzle” є найпопулярнішим в Україні на момент лютого 2021 року [6]. За частотою гри в категорії “Найчастіше за все” жанр “puzzle” посідає 1-ше місце, в категорії “Час від часу” жанри “adventure” та “action” посідають 3-е місце з однаковим результатом 31% голосів. Результати опитування наведені у таблиці 2.2.

Таблиця 2.2 – Найпопулярніші жанри відеоігор серед українців (лютий 2021 р.)

Жанр	**За частотою гри**	
	Граю частіше за все	**Граю час від часу**
Puzzle	12%	28%
Vehicle shooter	11%	24%
Shooter	10%	38%
Simulator	10%	32%
Strategy	9%	30%
Sport	9%	26%
MMORPG	6%	21%
Action	6%	31%
Adventure	6%	31%
MOBA	6%	19%
Sandbox	4%	19%
RPG	3%	18%
Battle Royale	3%	20%
Fighting	2%	22%
Card Game	2%	12%

2.5 Проектування ігрового додатку

За результатами дослідження для проекту обрано жанр “adventure” з елементами пазлу та платформінгу (puzzle platforming). Основу ігрового процесу складають пересування ігровим середовищем та розв’язування інтелектуальних задач-пазлів шляхом знаходження та взаємодії з об’єктами (див. рис. 2.9).

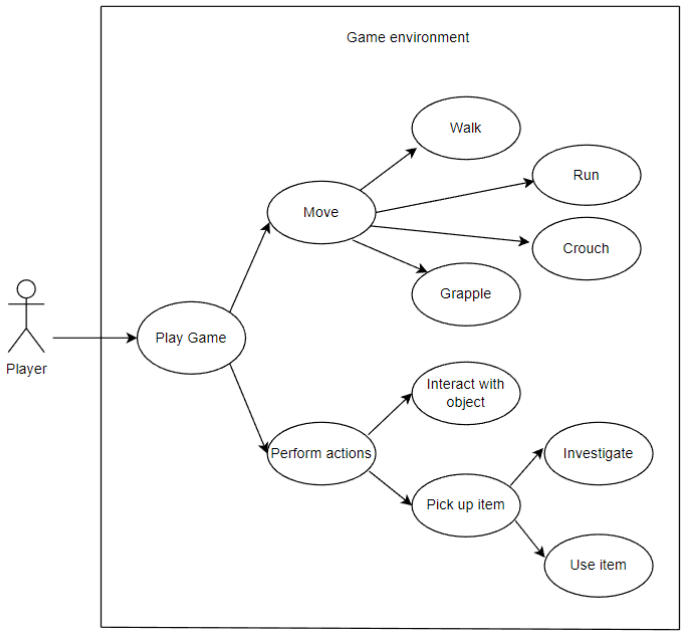

Рисунок 2.9 – UML діаграма основних можливостей гравця

Дизайн ігрового середовища, відповідно до особливостей ігрового процесу, повинен мати фізичні перешкоди – перепони які гравець повинен долати використовуючи можливості персонажа до пересування місцевістю (стрибки, гак з мотузкою), та елементи головоломок (пошук предметів, взаємодія з ними). Стилістика печери добре підходить для втілення даної задачі. За допомогою особливостей ландшафту можливо створити природні перешкоди для секцій платформінгу, а тематика стародавніх підземних руїн створює простір для дизайну різноманітних елементів пазлів.

Для проектування сцени створено скетч, зображений на рисунку 2.10. Сцена ігрового середовища складається з двох “приміщень”. Перше приміщення є початковою точкою для гравця та має зачинений прохід, ключ від якого знаходиться в іншій секції. Гравець має дістатися до ключа долаючи набір перешкод.

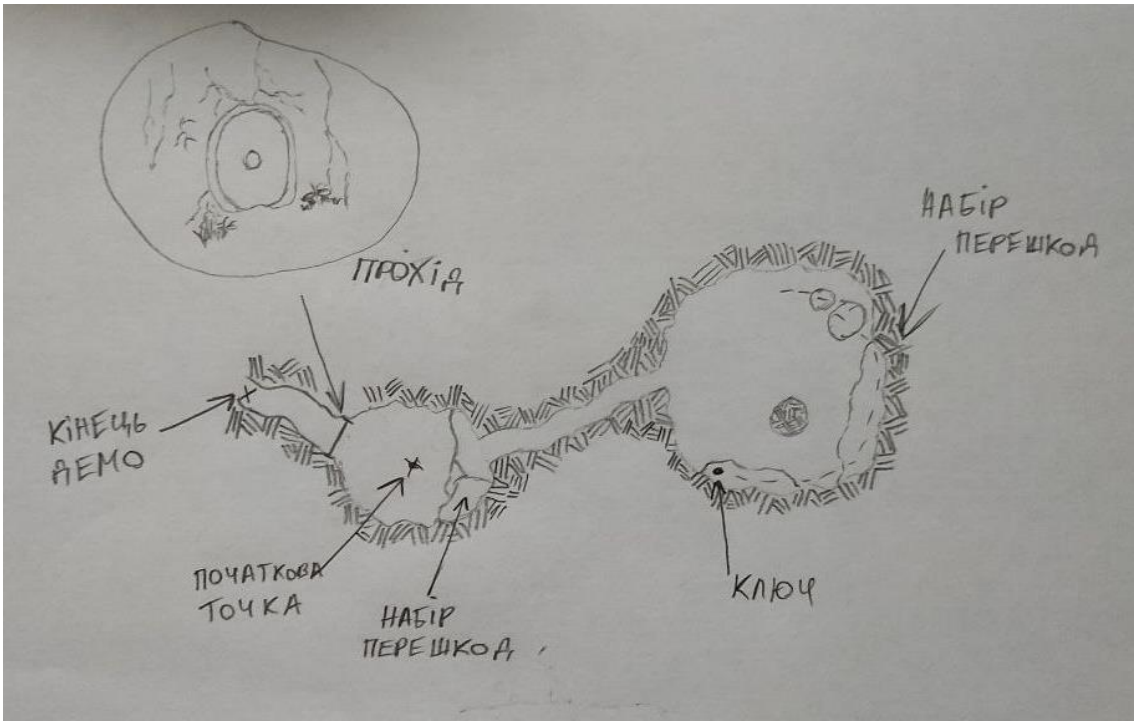

Рисунок 2.10 – Скетч-прототип ігрового середовища

3 ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

3.1 Створення сценаріїв та персонажа гравця

Сценарії (Scripts) є невід’ємним компонентом усіх проектів Unity. Вони дозволяють налагодити взаємодію між користувачем (гравцем) та програмою (ігровим середовищем). Використання сценаріїв дає можливість керування фізичною поведінкою об’єктів та їх взаємодією, застосування графічних ефектів а також проектування системи штучного інтелекту [7]. Для написання сценаріїв у Unity використовується мова програмування C# .

Ігровий об'єкт (GameObject) – будь-який об'єкт сцени в Unity (камера, персонаж, елементи оточення, джерела світла, тощо). Властивості ігровим об’єктам надаються за допомогою компонентів (Components) – функціональних складових які характеризують поведінку ігрового об’єкта. Сценарії також додаються до ігрових об’єктів як компоненти (див. рис. 3.1).

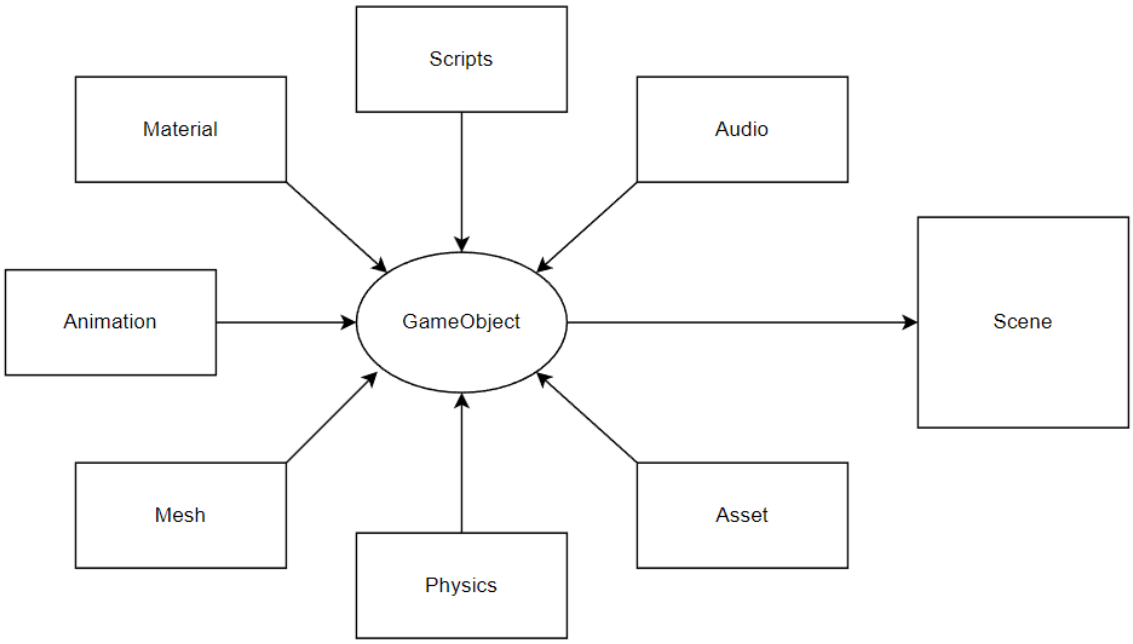

Рисунок 3.1 – Схема базової побудови ігрових об’єктів

Таким чином за допомогою сценаріїв потрібно реалізувати:

- пересування персонажа (поворот камери, ходьба, біг, присідання, пересування за допомогою гака);
- систему взаємодії з об’єктами (підбирання, огляд та застосування предметів);
- засоби програвання звукових ефектів та фонової музики.

Оскільки для ігрового додатку обрана перспектива від першої особи, персонаж гравця не обов’язково повинен мати окрему деталізовану тривимірну модель. Основними складовими персонажа є камера та капсула з колайдером (сіткою для відстеження зіткнення з поверхнею). Додатково присутні ігрові об’єкти що зберігатимуть дані про орієнтацію персонажа та камери, положення предметів при триманні їх «у руці» та реалізовуватимуть програвання звукових ефектів та музики (див. рис. 3.2).

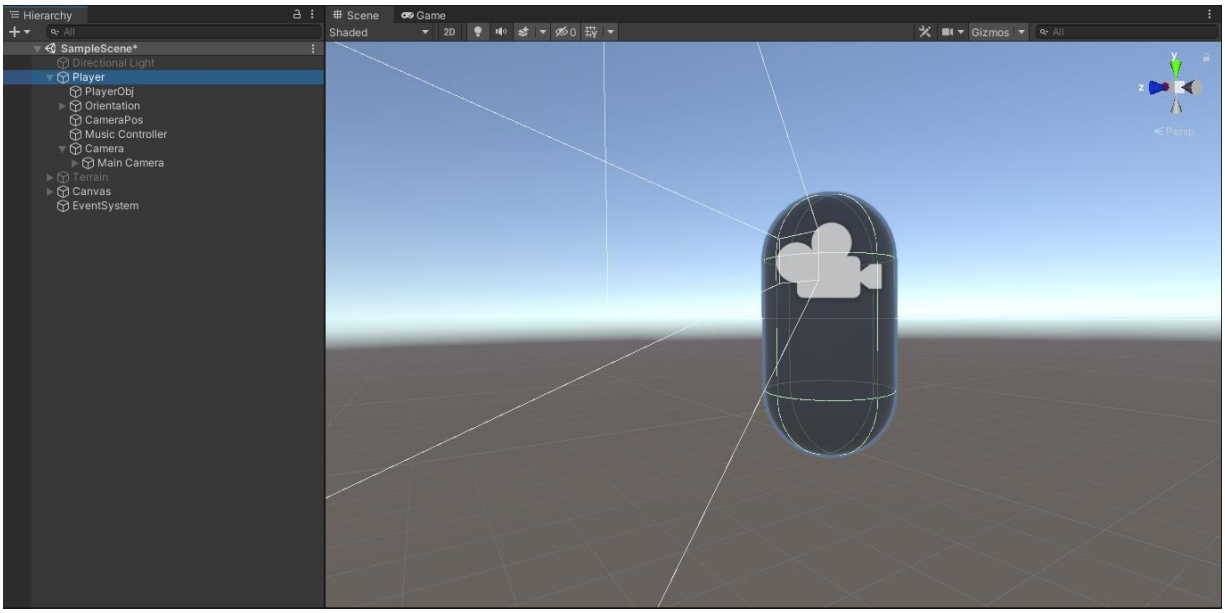

Рисунок 3.2 – Персонаж гравця

Положення та пересування об’єктів у Unity відбувається за допомогою системи координат (див. рис. 3.3). Одиниці вимірювання відповідають метричній системі (1 дорівнює 1 метр). Кожен ігровий об’єкт за замовчуванням володіє компонентом Transform (трансформація), що містить дані про його поло-

ження, кути нахилу та масштаб. Поворот камери гравця відбувається шляхом зміни значень кутів нахилу компонента Transform камери пропорційно руху миші по осям X (горизонталі) та Y(вертикалі). Сценарій реалізації повороту камери наведено в додатку А.

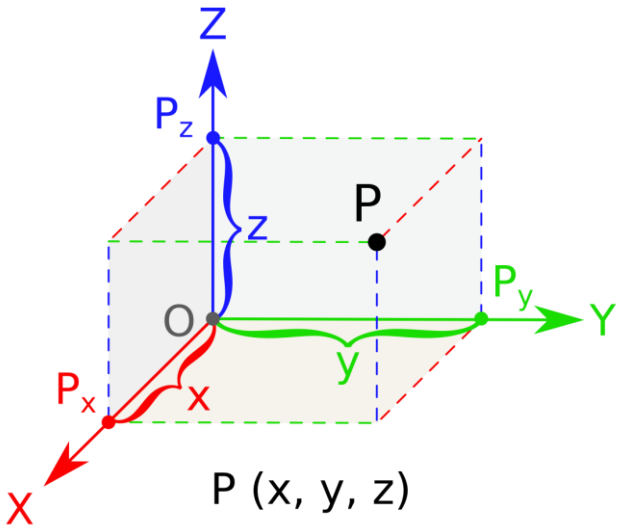

Рисунок 3.3 – Тривимірний простір

У Unity присутня вбудована технологія Nvidia PhysX, що дозволяє імітувати фізику об’єктів у 3D просторі: прискорення, зіткнення, сила тяжіння, тощо. Для застосування даних властивостей ігровий об’єкт повинен містити компонент Rigidbody (тверде тіло). Пересування персонажа відбувається шляхом застосування сили у напрямку, залежному від натиснутих клавіш клавіатури. Використовуються змінні для перемикання між «станами» об’єкту: ходьба, біг, присідання, знаходження у повітрі (див. додаток Б). Це дозволяє змінювати параметри швидкості, повітряного опору чи висоти колайдеру для ліпшої імітації пересування ігровим середовищем (див. рис. 3.4).

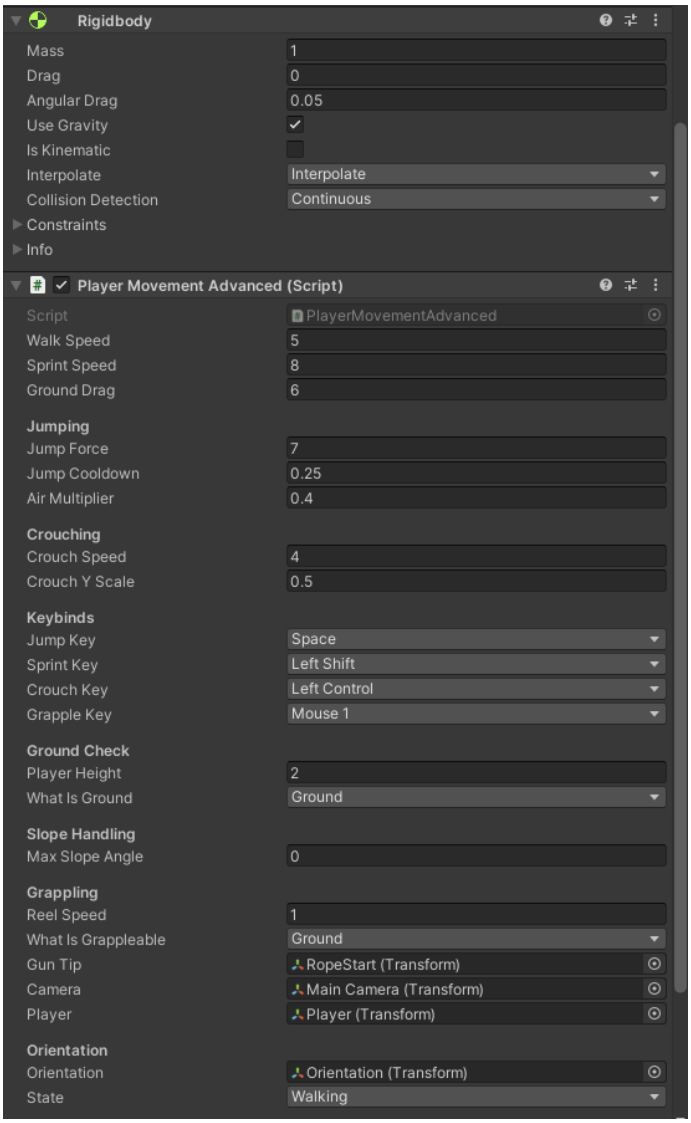

Рисунок 3.4 – Компоненти персонажа гравця

Відтворення звукових ефектів та музики в Unity виконується із застосуванням ігрових об’єктів, що виконують роль «джерел звуку». Для призначення об’єктам аудіо файлів для програвання використовується сценарій, при цьому ініціація програвання може відбуватися у інших сценаріях. Таким чином можна контролювати який елемент аудіо та за яких умов буде програватися. Для відтворення звуку потрібно додати компонент прослуховувач (Audio Listener) до персонажа гравця. Сценарії реалізації програвання елементів аудіо наведені у додатках В-Г.

Взаємодія з предметами та певними об’єктами середовища також виконується за допомогою сценаріїв. Сценарій можливості взаємодії є компонентом

персонажа гравця, в той час як сценарії результату взаємодії присвоюються об’єктам з якими відбувається взаємодія. При піднятті предмета змінюються параметри його положення відповідно до головної камери. Таким чином, доки об’єкт знаходиться «у руці», він зберігає своє положення відносно персонажа та камери при пересуванні гравця. Сценарій реалізації взаємодії з предметами наведено в додатку Ґ.

В результаті реалізації програмної частини отримано діаграму класів (див. рис. 3.5). Діаграма відображає існуючі класи проекту, змінні та методи кожного з них, а також їх відносини до інших типів даних (компонентів GameObject, Transform, тощо). Діаграма створена використовуючи розширення Codartis в середовищі Visual Studio. Даний інструмент дозволяє створювати інтерактивну візуалізацію коду проекту використовуючи стандарти UML для побудови діаграм. Повнорозмірну діаграму наведено в додатку Є.

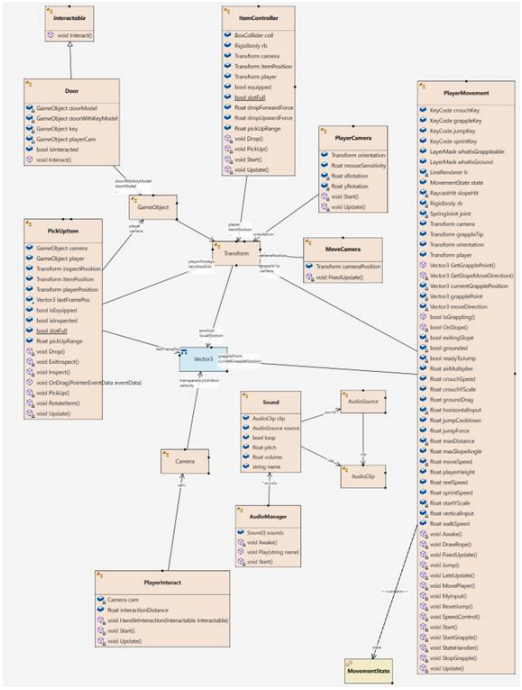

Рисунок 3.5 – Діаграма класів проекту

3.2 Створення тривимірної графіки

Полігональне моделювання – підхід створення тривимірних об’єктів шляхом апроксимації вигляду їх поверхні за допомогою багатокутників. Поверхня об’єкта складається з сітки точок у тривимірному просторі, що з’єднуються у ребра, а ті в свою чергу, формують багатокутні поверхні – полігони.

Для візуальної складової додатку обрано «low poly» стилістику, моделі складаються з малої кількість полігонів. Даний підхід водночас спрощує процес моделювання об’єктів та зменшує затрати на рендер – візуалізацію об’єктів під час роботи додатку. Оскільки тематикою ігрового середовища обрано печеру, тому об’єктами моделювання є різні варіації каміння, валунів, коріння, мінералів, елементів рослинності, тощо.

Основу об’єкта складає проста геометрична фігура, форма якої в режимі редагування (Edit mode) за допомогою інструментів видавлювання поверхонь (Extrude), масштабування (Scale) та пересування (Move) змінюється до бажаного вигляду (див. рис. 3.6) [8].

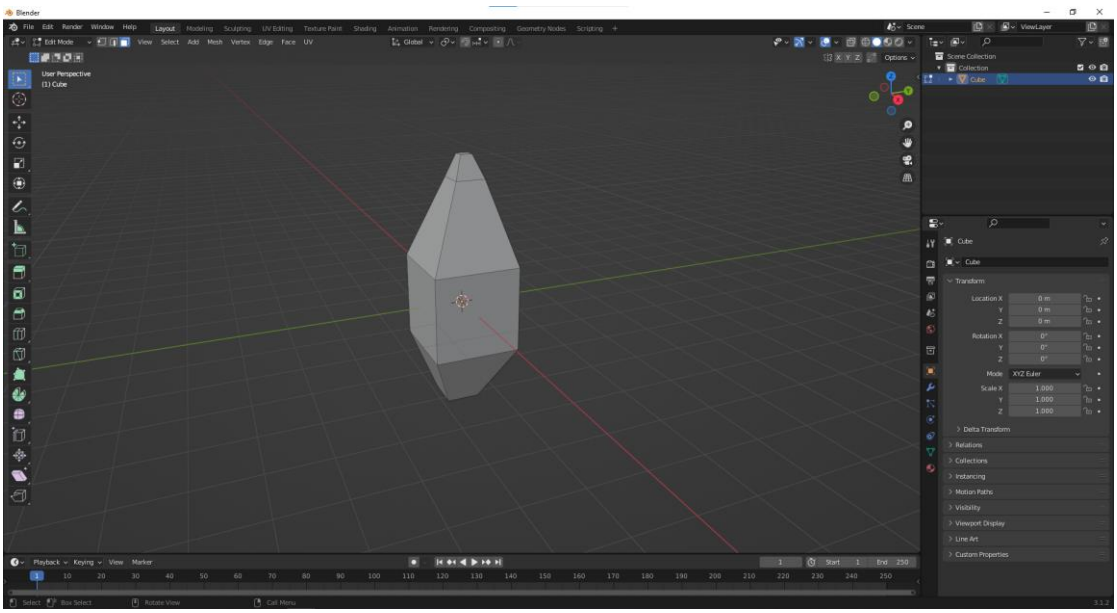

Рисунок 3.6 – Надання базової форми об’єкту

Далі в режимі ліплення (Sculpt mode) інструментами малювання (Draw) та зладження (Smooth), з увімкненою динамічною топологією (Dyntopo), об’єкту надається більш точна форма (див. рис. 3.7).

Динамічна топологія – це метод динамічної теселяції (підподіл поверхні на комірки) при ліпленні, що додає та видаляє точки сітки на льоту, тоді як звичайне ліплення лише впливає на існуючу форму сітки об’єкта. Цей метод дає змогу виліплювати складні форми поверхонь [9].

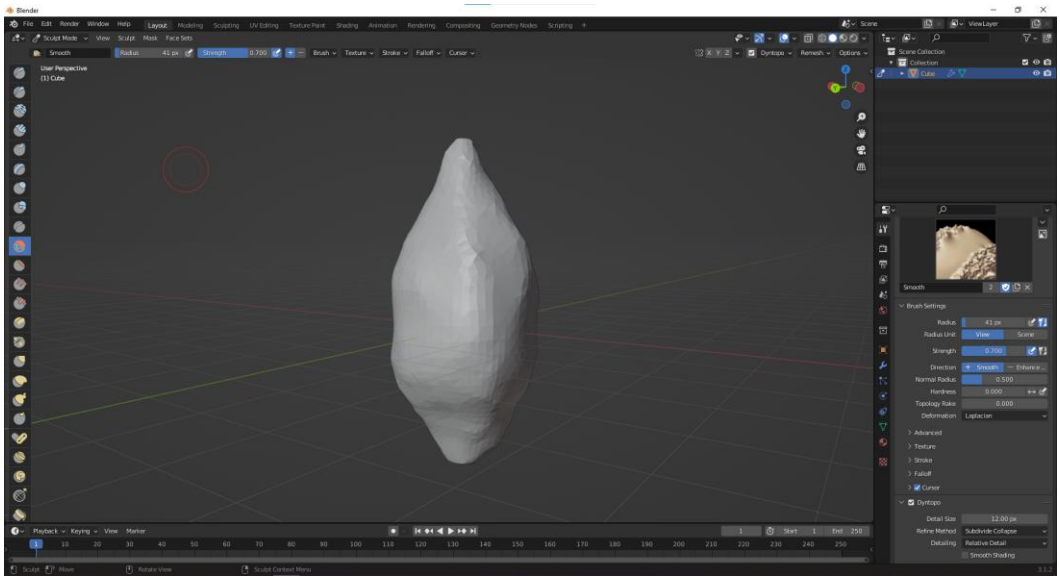

Рисунок 3.7 – Надання форми об’єкту в режимі ліплення

Далі до об’єкта додається модифікатор спрощення (Decimate), що скорочує кількість вершин/граней сітки з мінімальними змінами форми. Стягування (Collapse) – вид спрощення під час якого вершини поступово зливаються, зберігаючи форму сітки та пропорцію загальної кількості граней (див. рис. 3.8).

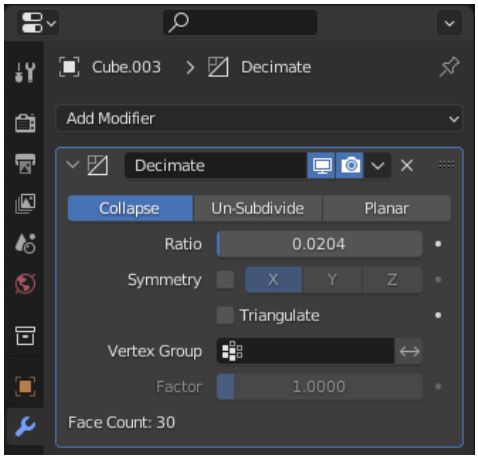

Рисунок 3.8 – Модифікатор Decimate

Unity та Blender використовують одиниці вимірювання метричної системи, тому розмір об’єктів простіше підібрати для масштабування відносно інших моделей. Для корегування пропорцій та кутів нахилу виконується операція застосування масштабу та обертання (Apply Rotation & Scale) у підменю *Object->Apply->Rotation & Scale*. Завершений об’єкт експортується у форматі .FBX зі збереженням змін трансформації (Apply Transform) (див. рис. 3.9).

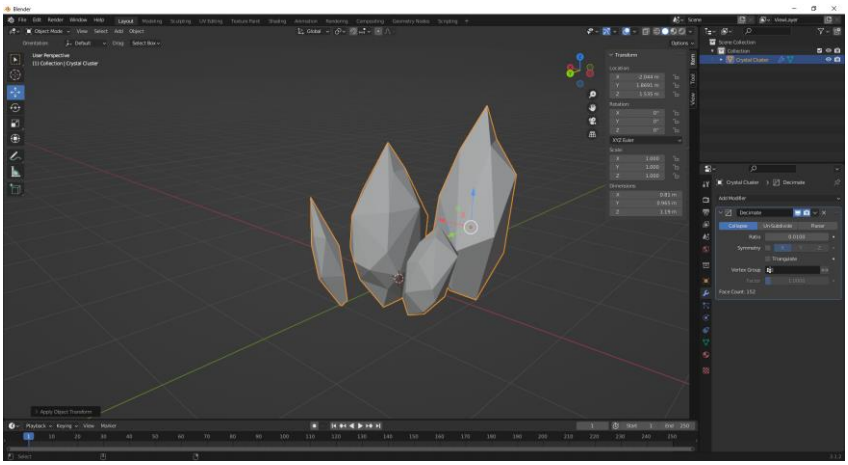

Рисунок 3.9 – Фінальний вигляд кристалічного кластеру

3.3 Створення звукового оформлення

Звукове оформлення ігрового додатку може складатися з фонового музичного супроводу та звукових ефектів певних об’єктів та результату їх взаємодії. Воно поліпшує атмосферу та занурення гравця в ігровий процес.

Для фонової музики створено просту мелодію за допомогою програмного засобу FL Studio. Для основної мелодії використовуються скляні дзвоники, звук яких можливо відтворити за допомогою вбудованого синтезатора Sytrus (див. рис.3.10).

Рисунок 3.10 – Вікно синтезатора Sytrus

В меню *Presets* існує набір параметрів *Bell => Glass*. Дані параметри дозволяють відтворити звук, що найближче схожий на скляний дзвоник.

Гармонічний осцилятор дозволяє змінювати амплітуду та вигляд звукової хвилі, що впливає на саме звучання інструмента. У підменю *OSC* гармонічні величини змінюються для отримання бажаного звучання (див. рис. 3.11).

Рисунок 3.11 – Гармонічний осцилятор у Sytrus

Piano roll є основним інструментом для створення та редагування послідовностей нот для мелодії. (див. рис. 3.12). Довжина ноти редагується розтягуванням її курсором, або комбінацією клавіш *Shift* + *Alt* та відповідною стрілкою вліво або вправо для зміни довжини виділених нот на поділку, що відповідає обраному значенню темпу у проекті.

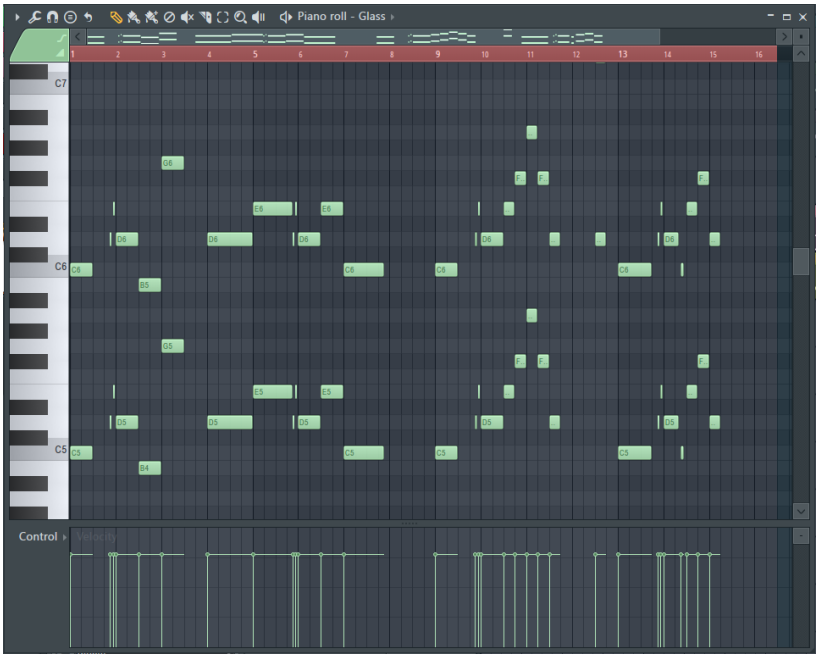

Рисунок 3.12 – Редактор послідовності нот (Piano roll)

До основної мелодії додається звучання арфи для ритмічної складової за допомогою розширення Sonatina Orchestra (див. рис. 3.13). Дане розширення пропонує набір різноманітних віртуальних оркестрових музичних інструментів для використання в проектах FL Studio. Для арфи створюється окрема відповідна послідовність нот.

Рисунок 3.13 – Редактор послідовності нот (Piano roll)

Панорамування — це розподіл аудіо сигналу у нове двухканальне або багатоканальне звукове поле. Розподіл на канали дозволяє окремо регулювати гучність кожного інструменту, а також напрямок для створення більш об’ємного звучання . Sonatina Orchestra та Sytrus розміщуються на окремі канали за допомогою вікна редактора Channel rack (див. рис. 3.14).

Рисунок 3.14 – Редактор каналів (Channel rack)

Для компоновки музики послідовності нот розміщуються на робочу доріжку, регулюються налаштування темпу і гучності (див. рис. 3.15). Кінцевий результат експортується у формат MP3 через меню *File => Export => MP3 file…*

Рисунок 3.15 – Результат, головне вікно FL Studio

3.4 Компоновка сцени

Всі елементи зберігаються у дереві проекту. Для моделей, матеріалів, сценаріїв, елементів аудіо та інших категорій створено власний каталог. Елементи можна переглядати та розміщати на сцену безпосередньо з браузера проекту (див. рис. 3.16).

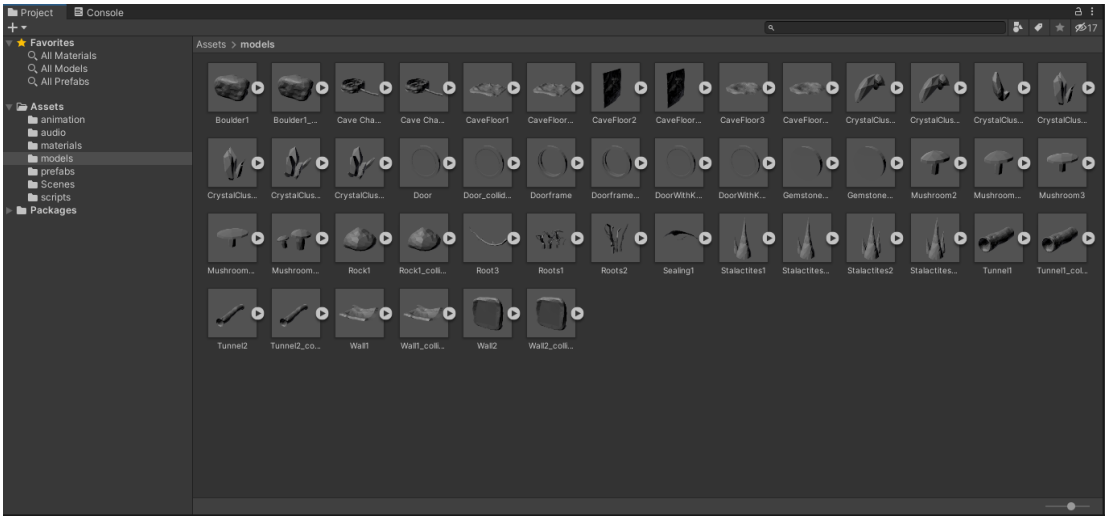

Рисунок 3.16 – Браузер проекту

Елементи оточення мають компонент сітку-колайдер (Mesh Collider), що дублює форму самого об’єкта, для коректного відстеження зіткнення з персонажем гравця. Для простоти редагування ігрових об’єктів існує спеціальний вид збірних об’єктів (prefabs). Prefab дозволяє дублювати ігровий об’єкт з усіма його компонентами та параметрами. При редагуванні параметрів, зміни застосовуються до всіх інших об’єктів цього типу. Використовуючи варіації об’єктів, створених у середовищі Blender, поступово будується більш деталізоване ігрове середовище (див. рис. 3.17).

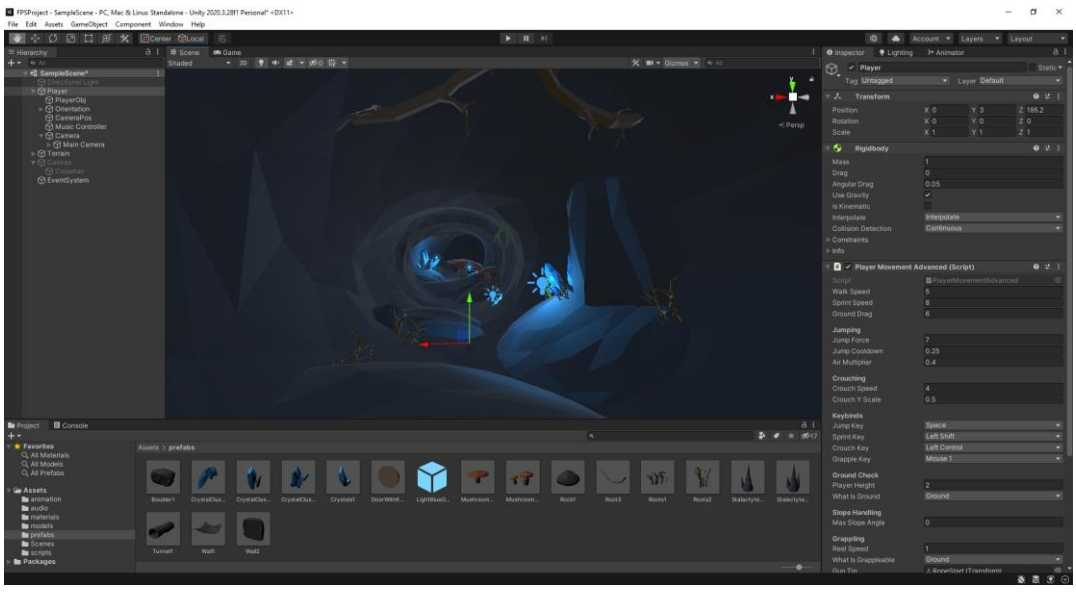

Рисунок 3.17 – Побудова ігрового середовища

Unity також володіє вбудованим інструментом створення анімації, хоч і простішим у порівнянні з Blender. Доступ до вікна анімації можливо отримати через меню *Window => Animation => Animation*, комбінацією клавіш *Ctrl + 6* або подвійним натисканням на файл анімації у браузері проекту. Інструмент дозволяє створювати анімацію ігрових об'єктів змінюючи параметри їх компоненту Transform (див. рис. 3.18). Апроксимація анімації між ключовими кадрами створюється автоматично.

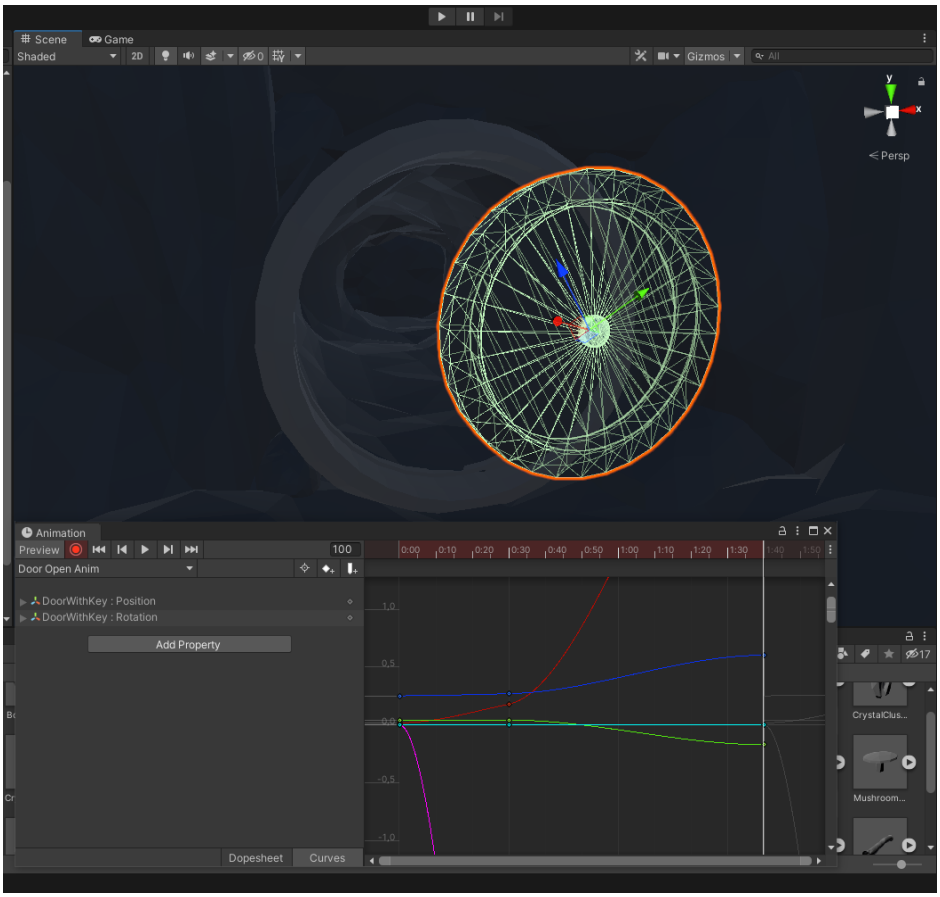

Рисунок 3.18 – Запис анімації у Unity

Для присвоєння елементів анімації ігровим об'єктам застосовується компонент аніматор (Animator) [10]. Він вимагає посилання на спеціальний контролер анімацій (Animator Controller) який визначає, які анімаційні кліпи вико-

ристовувати, а також контролює, коли і як переключатися між ними. Сценарії реалізації анімації відчинення проходу наведені у додатках Д-Е.

Фінальним етапом є створення збірки проекту – побудова програмного рішення на основі проекту з редактора. Доступ до параметрів відбувається через меню *File => Build Settings…* У рамках роботи відбувається збірка проекту для персональних комп’ютерів, з операційною системою Windows архітектури x86_64.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи бакалавра проведені дослідження сучасних джерел за тематикою розробки ігрових додатків. За результатами проведених досліджень створено інформаційне та програмне забезпечення ігрового додатку жанру “adventure” (демо-версію) засобами інтегрованого середовища Unity. Виконано основні завдання для реалізації:

1. Реалізувати керування ігровим персонажем.
2. Реалізувати функції побудови тривимірних об’єктів та середовища.
3. Реалізувати функції аудіо-оформлення на фоні.

Побудову тривимірних об’єктів для ігрового середовища у “low poly” стилістиці виконано за допомогою інструментарію програмного засобу Blender. Елементи аудіо-оформлення створено за допомогою музичного редактору FL Studio.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. А. В. Гречко, Н. В. Захаров, М. О. Фалько, АНАЛІЗ ДИНАМІКИ РОЗВИТКУ РИНКУ ВІДЕОІГОР, ДЖЕРЕЛ ЙОГО ФІНАНСУВАННЯ ТА ОСОБЛИВОСТЕЙ МОНЕТИЗАЦІЇ ПРОДУКЦІЇ В ДАНІЙ СФЕРІ. *Ефективна економіка*. 2021. № 5 [Електронний ресурс] // http://www.economy.nayka.com / – Режим доступу до ресурсу: http://www.economy.nayka.com.ua/?op=1&z=8871

2. Joe Keeley, What Are Adventure Games and How Have They Evolved? [Електронний ресурс] // https://www.makeuseof.com / – Режим доступу до ресурсу: https://www.makeuseof.com/what-are-adventure-games/

3. A Story About My Uncle [Електронний ресурс] // https://www.metacritic.com / – Режим доступу до ресурсу: https://www.metacritic.com/game/pc/a-story-about-my-uncle

4. Keza MacDonald, The Stanley Parable Review [Електронний ресурс] // https://www.ign.com / – Режим доступу до ресурсу: https://www.ign.com/articles/2013/10/18/the-stanley-parable-review

5. Most popular video game genres among internet users worldwide as of 3rd quarter 2021, by age group [Електронний ресурс] // https://www.statista.com / – Режим доступу до ресурсу: https://www.statista.com/statistics/1263585/top-video-game-genres-worldwide-by-age/

6. Які відеоігри найпопулярніші в Україні та скільки за них платять – дослідження [Електронний ресурс] // https://www.the-village.com.ua / – Режим доступу до ресурсу: https://www.the-village.com.ua/village/business/news/307871-yaka-videogra-naypopulyarnisha-v-ukrayini-doslidzhennya

7. Creating and Using Scripts [Електронний ресурс] //https://docs.unity3d.com / – Режим доступу до ресурсу: https://docs.unity3d.com/Manual/CreatingAndUsingScripts.html

8. James Chronister Blender Basics Classroom Tutorial Book 4-th edition. / James Chronister, 2012. - 416 с.

9. Динамічна топологія – Dyntopo [Електронний ресурс] // https://docs.blender.org / – Режим доступу до ресурсу: https://docs.blender.org/manual/uk/latest/sculpt_paint/sculpting/tool_settings/dyntopo.html

10. Animator component [Електронний ресурс] //https://docs.unity3d.com / – Режим доступу до ресурсу: https://docs.unity3d.com/Manual/class-Animator.html

ДОДАТОК А

Сценарій реалізації повороту камери

```
using UnityEngine;

public class PlayerCamera : MonoBehaviour
{

    public float mouseSensitivity;

    public Transform orientation;

    float xRotation = 0f;
    float yRotation = 0f;

    void Start()
    {
        Cursor.lockState = CursorLockMode.Locked;
        Cursor.visible = false;
    }

    void Update()
    {
        float mouseX = Input.GetAxisRaw("Mouse X") * mouseSensitivity * Time.deltaTime;
        float mouseY = Input.GetAxisRaw("Mouse Y") * mouseSensitivity * Time.deltaTime;

        xRotation -= mouseY;
        xRotation = Mathf.Clamp(xRotation, -90f, 90f);
        yRotation += mouseX;

        transform.rotation = Quaternion.Euler(xRotation, yRotation, 0);
        orientation.rotation = Quaternion.Euler(0, yRotation, 0f);
    }
}
```

ДОДАТОК Б

Сценарій реалізації пересування ігрового персонажа

```
using UnityEngine;

public class PlayerMovement : MonoBehaviour
{
    [Header("Movement")]
    private float moveSpeed;
    public float walkSpeed;
    public float sprintSpeed;

    public float groundDrag;

    [Header("Jumping")]
    public float jumpForce;
    public float jumpCooldown;
    public float airMultiplier;
    bool readyToJump;

    [Header("Crouching")]
    public float crouchSpeed;
    public float crouchYScale;
    private float startYScale;

    [Header("Keybinds")]
    public KeyCode jumpKey = KeyCode.Space;
    public KeyCode sprintKey = KeyCode.LeftShift;
    public KeyCode crouchKey = KeyCode.LeftControl;
    public KeyCode grappleKey = KeyCode.Mouse1;

    [Header("Ground Check")]
    public float playerHeight;
    public LayerMask whatIsGround;
    bool grounded;

    [Header("Slope Handling")]
    public float maxSlopeAngle;
    private RaycastHit slopeHit;
    private bool exitingSlope;

    [Header("Grappling")]
    public float reelSpeed;
    public LayerMask whatIsGrappleable;
    public Transform gunTip, camera, player;
    private LineRenderer lr;
    private Vector3 grapplePoint;
    private Vector3 currentGrapplePosition;
    private float maxDistance = 20f;
    private SpringJoint joint;

    [Header("Orientation")]
    public Transform orientation;

    float horizontalInput;
    float verticalInput;

    Vector3 moveDirection;

    Rigidbody rb;
```

```
    public MovementState state;
    public enum MovementState
    {
        walking,
        sprinting,
        crouching,
        grappling,
        air
    }

    void Awake()
    {
        lr = GetComponent<LineRenderer>();
    }

    private void Start()
    {
        rb = GetComponent<Rigidbody>();
        rb.freezeRotation = true;

        readyToJump = true;

        startYScale = transform.localScale.y;
    }

    private void Update()
    {
        // ground check
        grounded = Physics.Raycast(transform.position, Vector3.down, playerHeight * 0.5f +
0.2f, whatIsGround);

        MyInput();
        SpeedControl();
        StateHandler();

        // handle drag
        if (grounded)
            rb.drag = groundDrag;
        else
            rb.drag = 0;
    }

    private void FixedUpdate()
    {
        MovePlayer();
    }

    void LateUpdate()
    {
        DrawRope();
    }

    private void MyInput()
    {
        horizontalInput = Input.GetAxisRaw("Horizontal");
        verticalInput = Input.GetAxisRaw("Vertical");

        // when to jump
        if(Input.GetKey(jumpKey) && readyToJump && grounded)
        {
            readyToJump = false;

            Jump();
```

```
            Invoke(nameof(ResetJump), jumpCooldown);
        }

        // start crouch
        if (Input.GetKeyDown(crouchKey))
        {

            GameObject.Find("PlayerObj").transform.localScale = new Vec-
tor3(transform.localScale.x, crouchYScale, transform.localScale.z);
            rb.AddForce(Vector3.down * 5f, ForceMode.Impulse);
        }

        // stop crouch
        if (Input.GetKeyUp(crouchKey))
        {
            GameObject.Find("PlayerObj").transform.localScale = new Vec-
tor3(transform.localScale.x, startYScale, transform.localScale.z);
        }
        // start grapple
        if (Input.GetKeyDown(grappleKey))
        {
            StartGrapple();
            //state = MovementState.grappling;
        }
        else if (Input.GetKeyUp(grappleKey))
        {
            StopGrapple();
        }
    }

    private void StateHandler()
    {
        // Mode - Crouching
        if (Input.GetKey(crouchKey))
        {
            state = MovementState.crouching;
            moveSpeed = crouchSpeed;
        }

        // Mode - Sprinting
        else if(grounded && Input.GetKey(sprintKey))
        {
            state = MovementState.sprinting;
            moveSpeed = sprintSpeed;
        }

        // Mode - Walking
        else if (grounded)
        {
            state = MovementState.walking;
            moveSpeed = walkSpeed;
        }

        else if (Input.GetKey(grappleKey))
        {
            state = MovementState.grappling;
            //moveSpeed = swingSpeed;
        }

        // Mode - Air
        else
        {
            state = MovementState.air;
        }
```

```
    }

    private void MovePlayer()
    {
        // calculate movement direction
        moveDirection = orientation.forward * verticalInput + orientation.right * horizon-
talInput;

        // on slope
        if (OnSlope() && !exitingSlope)
        {
            rb.AddForce(GetSlopeMoveDirection() * moveSpeed * 20f, ForceMode.Force);

            if (rb.velocity.y > 0)
                rb.AddForce(Vector3.down * 80f, ForceMode.Force);
        }

        // on ground
        else if(grounded)
            rb.AddForce(moveDirection.normalized * moveSpeed * 10f, ForceMode.Force);

        // in air
        else if(!grounded)
            rb.AddForce(moveDirection.normalized * moveSpeed * 10f * airMultiplier, Force-
Mode.Force);

        // turn gravity off while on slope
        rb.useGravity = !OnSlope();
    }

    private void SpeedControl()
    {
        // limiting speed on slope
        if (OnSlope() && !exitingSlope)
        {
            if (rb.velocity.magnitude > moveSpeed)
                rb.velocity = rb.velocity.normalized * moveSpeed;
        }

        // limiting speed on ground or in air
        else
        {
            Vector3 flatVel = new Vector3(rb.velocity.x, 0f, rb.velocity.z);

            // limit velocity if needed
            if (!grounded && flatVel.magnitude > moveSpeed)
            {
                Vector3 limitedVel = flatVel.normalized * moveSpeed;
                rb.velocity = new Vector3(limitedVel.x, rb.velocity.y, limitedVel.z);
            }
        }
    }

    private void Jump()
    {
        exitingSlope = true;

        // reset y velocity
        rb.velocity = new Vector3(rb.velocity.x, 0f, rb.velocity.z);

        rb.AddForce(transform.up * jumpForce, ForceMode.Impulse);
    }
    private void ResetJump()
    {
```

```
        readyToJump = true;

        exitingSlope = false;
    }

    private bool OnSlope()
    {
        if(Physics.Raycast(transform.position, Vector3.down, out slopeHit, playerHeight *
0.5f + 0.3f))
        {
            float angle = Vector3.Angle(Vector3.up, slopeHit.normal);
            return angle < maxSlopeAngle && angle != 0;
        }

        return false;
    }

    private Vector3 GetSlopeMoveDirection()
    {
        return Vector3.ProjectOnPlane(moveDirection, slopeHit.normal).normalized;
    }

    void StartGrapple()
    {
        RaycastHit hit;
        if (Physics.Raycast(camera.position, camera.forward, out hit, maxDistance, whatIs-
Grappleable))
        {
            grapplePoint = hit.point;
            joint = player.gameObject.AddComponent<SpringJoint>();
            joint.autoConfigureConnectedAnchor = false;
            joint.connectedAnchor = grapplePoint;

            float distanceFromPoint = Vector3.Distance(player.position, grapplePoint);

            //The distance grapple will try to keep from grapple point.
            joint.maxDistance = distanceFromPoint * 0.2f;
            joint.minDistance = distanceFromPoint * 0.1f;

            //Adjust these values to fit your game.
            joint.spring = 1.5f;
            joint.damper = 5f;
            joint.massScale = 4.5f;

            lr.positionCount = 2;
            currentGrapplePosition = gunTip.position;

            Vector3 grappleDirection = (gunTip.position - transform.position).normalized;
            rb.AddForce(orientation.forward * reelSpeed * 10f, ForceMode.Force);
        }
    }

    void StopGrapple()
    {
        lr.positionCount = 0;
        Destroy(joint);
    }

    void DrawRope()
    {
        //If not grappling, don't draw rope
        if (!joint) return;
```

```
        currentGrapplePosition = Vector3.Lerp(currentGrapplePosition, grapplePoint,
Time.deltaTime * 8f);

        lr.SetPosition(0, gunTip.position);
        lr.SetPosition(1, currentGrapplePosition);
    }

    public bool IsGrappling()
    {
        return joint != null;
    }

    public Vector3 GetGrapplePoint()
    {
        return grapplePoint;
    }
}
```

ДОДАТОК В

Клас для характеристики елементів аудіо

```
using UnityEngine;

[System.Serializable]
public class Sound
{
    public string name;
    public AudioClip clip;
    [Range (0f, 1f)]
    public float volume;
    [Range(0.1f, 3f)]
    public float pitch;
    public bool loop;

    [HideInInspector]
    public AudioSource source;

}
```

ДОДАТОК Г

Сценарій реалізації програвання елементів аудіо

```
using UnityEngine;
using System;

public class AudioManager : MonoBehaviour
{
    public Sound[] sounds;

    void Awake()
    {
        foreach(Sound s in sounds)
        {
            s.source = gameObject.AddComponent<AudioSource>();
            s.source.clip = s.clip;
            s.source.volume = s.volume;
            s.source.pitch = s.pitch;
            s.source.loop = s.loop;
        }
    }

    void Start()
    {
        Play("Chamber1");
    }

    public void Play(string name)
    {
        Sound s = Array.Find(sounds, sound => sound.name == name);
        s.source.Play();
    }
}
```

ДОДАТОК Ґ

Сценарій реалізації взаємодії гравця з предметами (підбирання та огляд)

```
using UnityEngine;
using UnityEngine.EventSystems;

public class PickUpItem : MonoBehaviour, IDragHandler
{
    public bool isEquipped;
    public bool isInspected;
    public static bool slotFull;
    public float pickUpRange;
    public Transform playerPosition, itemPosition, inspectPosition;
    public GameObject player, camera;
    protected Vector3 lastFramePos;

    private void Update()
    {
        //Check if player is in range and pick up button is pressed
        Vector3 distanceToPlayer = playerPosition.position - transform.position;
        if (!isEquipped && distanceToPlayer.magnitude <= pickUpRange && In-
put.GetKeyDown(KeyCode.E) && !slotFull) PickUp();

        //Drop if equipped and drop button is pressed
        if (isEquipped && Input.GetKeyDown(KeyCode.Q)) Drop();

        //Check if player pressed inspect button
        if (isEquipped && !isInspected && Input.GetKeyDown(KeyCode.F)) Inspect();

        //Check if player pressed inspect button again
        if (isEquipped && isInspected && Input.GetKeyDown(KeyCode.E)) ExitInspect();

        //Rotate item if being inspected
        if (isEquipped && isInspected && Input.GetMouseButtonDown(0)) lastFramePos = In-
put.mousePosition;
        if (isEquipped && isInspected && Input.GetMouseButton(0)) RotateItem();
    }

    private void PickUp()
    {
        isEquipped = true;
        slotFull = true;

        GetComponent<MeshCollider>().enabled = false;
        GetComponent<Rigidbody>().useGravity = false;
        GetComponent<Rigidbody>().isKinematic = true;
        this.transform.position = itemPosition.position;
        this.transform.parent = GameObject.Find("Main Camera").transform;
    }

    private void Drop()
    {
        isEquipped = false;
        slotFull = false;

        GetComponent<MeshCollider>().enabled = true;
        GetComponent<Rigidbody>().useGravity = true;
        GetComponent<Rigidbody>().isKinematic = false;

        //Set parent to null
        transform.SetParent(null);
```

```
    }

    private void Inspect()
    {
        isInspected = true;
        this.transform.position = inspectPosition.position;
        this.transform.rotation = new Quaternion(0, 0, 0, 0);
        player.GetComponent<PlayerMovement>().enabled = false;
        camera.GetComponent<PlayerCamera>().enabled = false;
        Cursor.lockState = CursorLockMode.None;
        Cursor.visible = true;
    }

    private void RotateItem()
    {
        var delta = Input.mousePosition - lastFramePos;
        lastFramePos = Input.mousePosition;
        var axis = Quaternion.AngleAxis(-90f, Vector3.forward) * delta;
        transform.rotation = Quaternion.AngleAxis(delta.magnitude * 0.1f, axis) * trans-
form.rotation;
    }

    private void ExitInspect()
    {
        isInspected = false;
        this.transform.position = itemPosition.position;
        this.transform.rotation = new Quaternion(0, 0, 0, 0);
        player.GetComponent<PlayerMovement>().enabled = true;
        camera.GetComponent<PlayerCamera>().enabled = true;
        Cursor.lockState = CursorLockMode.Locked;
        Cursor.visible = false;
    }

    public void OnDrag(PointerEventData eventData)
    {
        inspectPosition.eulerAngles = new Vector3(-eventData.delta.y, -eventData.delta.x);
    }
}
```

ДОДАТОК Д

Сценарій реалізації взаємодії гравця з елементами оточення (відчинення проходу за наявності ключа)

```
using UnityEngine;

public class PlayerInteract : MonoBehaviour
{
    public float interactionDistance;

    Camera cam;

    void Start()
    {
        cam = Camera.main;
    }

    void Update()
    {
        Ray ray = cam.ScreenPointToRay(new Vector3(Screen.width / 2f, Screen.height / 2f,
0f));
        RaycastHit hit;

        bool successfulHit = false;

        if (Physics.Raycast(ray, out hit, interactionDistance))
        {
            Interactable interactable = hit.collider.GetComponent<Interactable>();

            if (interactable != null)
            {
                HandleInteraction(interactable);
                successfulHit = true;
            }
        }
    }

    void HandleInteraction(Interactable interactable)
    {
        KeyCode key = KeyCode.Mouse0;
        if (Input.GetKeyDown(key))
        {
            interactable.Interact();
        }
    }
}
```

ДОДАТОК Е

Сценарій реалізації реакції об’єкту на дію гравця (відчинення проходу)

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

public class Door : Interactable
{
    [SerializeField]
    private GameObject doorWithKeyModel;
    [SerializeField]
    private GameObject doorModel;
    [SerializeField]
    private GameObject key;
    [SerializeField]
    private GameObject playerCam;
    [SerializeField]
    public bool isInteracted;
  public override void Interact()
    {
        if (!isInteracted && key.transform.IsChildOf(playerCam.transform))
        {
            isInteracted = true;
            doorWithKeyModel.SetActive(true);
            Destroy(doorModel);
            Destroy(key);
            doorWithKeyModel.GetComponent<Animator>().Play("DoorOpenAnim");
        }
    }
}
```

ДОДАТОК Є

Діаграма класів проекту