Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка ізометричної 2.5D гри з використанням Unity 6 та SRP фреймворку

Виконав: студент IV курсу, групи СП-43 спеціальності
121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

	(підпис)	Тітков В. В. (прізвище та ініціали)
Керівник	(підпис)	Стоянов Ю .М. (прізвище та ініціали)
Нормоконтроль	(підпис)	Стоянов Ю. М. (прізвище та ініціали)
Завідувач кафедри	(підпис)	Петрик М. Р. (прізвище та ініціали)
Рецензент	(підпис)	(прізвище та ініціали)

Тернопіль 2025

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет Комп'ютерно інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра Програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Петрик М. Р.
(підпис) (прізвище та ініціали)
« » 2025 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавр
(назва освітнього ступеня)

за спеціальністю 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

студенту Тітков Владислав Вікторович
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка ізометричної 2.5D гри з використанням Unity 6 та SRP фреймворку

Керівник роботи к.т.н., доц. каф ПІ Стоянов Ю. М
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «___» _____ 2025 року № _____

2. Термін подання студентом завершеної роботи _____

3. Вихідні дані до роботи Предметна область, технічне завдання, вимоги та специфікація, програмне рішення

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. Розділ 1. Аналіз предметної області та задач. Розділ 2. Проєктування гри. Розділ 3. Тестування гри. Розділ 4. Безпека життєдіяльності та основи охорони праці. Висновок. Список використаних джерел. Додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Діаграми, знімки екрану з проробленою роботою, ілюстративні зображення, інформативні зображення для доповнення тексту.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці			

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Ознайомлення з завданням до кваліфікаційної роботи		
2	Підбір джерел по темі кваліфікаційної роботи		
3	Опрацювання джерел інформації по темі кваліфікаційної роботи		
4	Розробка архітектури програмного рішення		
5	Ознайомлення з новими технологіями		
6	Розробка гри на основі опрацьованої інформації		
7	Виконання звіту по проробленій роботі		
8	Виконання завдання до підрозділу «Безпека життєдіяльності»		
9	Виконання завдання до підрозділу «Основи охорони праці»		
10	Оформлення кваліфікаційної роботи		
11	Нормоконтроль		
12	Перевірка на плагіат		
13	Попередній захист кваліфікаційної роботи		
14	Захист кваліфікаційної роботи		

Студент _____ Тітков В. В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Стоянов Ю. М.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розробка ізометричної 2.5D гри з використанням Unity 6 та SRP фреймворку // Кваліфікаційна робота освітнього рівня «Бакалавр» // Тітков Владислав Вікторович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно інформаційних систем і програмної інженерії, кафедра програмної інженерії, група СП-43 // Тернопіль, 2025 // Ст. – 72, рис. – 43, табл – 2, кресл. – 0, додат. – 3, бібліогр. – 20.

Ключові слова: Unity 6, SRP, URP, ізометрія, C#, комп'ютерна гра, Unity DevOps Version Control, New Input System, Visual Studio Code.

Головною метою цієї кваліфікаційної роботи є дослідження етапів аналізу, проєктування, створення та тестування комп'ютерної гри з використанням сучасних технологій та інструментів у галузі розробки комп'ютерних ігор.

Перший розділ присвячено дослідженню предметної області та аналізу найпоширеніших ігрових рушіїв.

У другому розділі розглянуто процес проєктування комп'ютерної гри, вибір архітектурного рішення, а також підготовку та налаштування середовища для розробки.

У третьому розділі наведено результати тестування гри на комп'ютерах з різними системними характеристиками, а також розглянуто можливі напрями її подальшого розвитку, як окремого продукту.

ABSTRACT

Development of an isometric 2.5D game using Unity 6 and SRP framework // Qualification work for the educational level ‘Bachelor’ / Titkov Vladyslav Viktorovych // Ternopil National Technical University named after Ivan Puluj, Faculty of Computer Information Systems and Software Engineering, Department of Software Engineering, group SP-43 // Ternopil, 2025 // Pp. - 72, Fig. - 43, Table - 2, Drawings - 0, Supplement - 3, Bibliography - 20.

Keywords: Unity 6, SRP, URP, Isometric, C#, computer game, Unity DevOps Version Control, New Input System, Visual Studio Code.

The main purpose of this qualification work is to study the stages of analysis, design, creation and testing of a computer game using modern technologies and tools in the field of computer game development.

The first section is devoted to the study of the subject area and the analysis of the most common game engines.

The second section describes the process of designing a computer game, choosing an architectural solution, as well as preparing and setting up the development environment.

The third section presents the results of testing the game on computers with different system characteristics, as well as possible directions for its further development as a separate product.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Геймдев – це сфера, що розробляє комп’ютерні ігри чи ігри для інших платформ (консолі, мобільні телефони). При розробці використовуються сучасні технології для передання якомога кращого користувацького досвіду.

Unity – це ігровий рушій, що заточений на розробці як 2D ігор так і 3D ігор, а також мобільних чи веб застосунків.

C# – це об’єктно-орієнтована мова програмування.

New Input System – покращена Unity система введення котра працює не лише із стандартними пристроями введення, а й із контролерами. Крім того ця система набагато гнучкіша в налаштуванні порівняно із оригінальною системою Unity.

Visual Studio Code – це безкоштовний, потужний редактор коду, що підтримує багато мов програмування, має вбудований термінал, дебагер для зручної розробки.

SRP (Scriptable Render Pipeline) – система котра дозволяє покращити продуктивність гри та контролювати візуальними ефектами, дає можливість писати свій код для візуальних ефектів.

URP (Universal Render Pipeline) – використовується для широкого кола платформ, оптимізований та легкий для слабких пристроїв.

HDRP (High Definition Render Pipeline) – використовується для досягання високоякісних графічних ефектів на потужних платформах таких як: персональні комп’ютери та консолі нового покоління.

Unity DevOps Version Control – це інструмент контролю версій та керування вихідним кодом для студій розробки ігор будь-якого розміру. Створений на основі технології Plastic SCM, Unity Version Control пропонує оптимізовані робочі процеси для художників і програмістів та чудову швидкість роботи з великими файлами і двійковими файлами.

Isometric/Ізометрія – це стиль перспективи, де об'єкти виглядають об'ємними, але зберігають однакові пропорції, зображаючи простір без перспективних

спотворень, такий метод перспективи дає можливість отримати ширший огляд гравця.

2.5D гра – відеогра котра поєднує в собі елементи 2D та 3D графіки. В таких іграх використовуються або 2D-об’єкти котрі можуть рухатись по осям XYZ, або 3D-об’єкти котрі обмежуються лише осями XY.

Asset – ресурси які використовуються в грі: Скрипти, моделі, текстури, спрайти та звуки.

Спрайт – це окреме двовимірне зображення, що застосовується в комп'ютерній графіці як частина більшого зображення.

Prefab – шаблон об’єкта, котрий може бути використаний безмежну кількість разів.

Шейдер – може описувати складні процеси, такі як поглинання й розсіювання світла, накладення текстур, віддзеркалення й заломлення, затінення, зміщення поверхні, а також застосування ефектів постобробки.

ЗМІСТ

ВСТУП

Відеоігри є одним із сучасних можливостей розважитись, крім того ігрова індустрія як і кіно індустрія має багато жанрів та є досить прибутковою. Найпопулярніші жанри відеоігор є: souls-подібні, MOBA, змагальні стрілянки, жахи, мандрівні бойовики. Ігрова індустрія є невід'ємною частиною сучасної культури та є одним із сильних рушіїв розвитку особистості.

Ця бакалаврська робота несе за собою мету розробити ізометричну 2.5D гру використовуючи сучасний рушій Unity 6, котрий порівняно із минулою версією приніс із собою вбудовану підтримку SRP фреймворку, що дозволить виробляти красивіші та продуктивніші відеоігри.

В процесі розробки гри планується розробити зрозумілий та інтуїтивний дизайн користувацького інтерфейсу, та поєднати два жанри: souls-like та бродилку.

Передбачається, що розробка гри такого типу збільшить приплив до української ігрової індустрії більше інвесторів, крім того це чудова можливість отримати перший крок в ігровій індустрії, як незалежний розробник, показуючи вміння використовувати сучасні технології для: освітлення та затінення, контролю версіями, дизайну рівнів. В складних проєктах, побудованих на основі Unity/Unity 6, взаємодія із тінями та освітленням є однією із важливих цілей для досягання реалістичного зображення, або створення неординарного графічного стилю.

Очікується, що в ході цього дослідження висновок та ідеї будуть сприяти розвитку сфери ігрової індустрії незалежно від жанру та стилю відеоігор котрі розробляються.

РОЗДІЛ 1. ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА АНАЛІЗ ІГРОВИХ РУШІЇВ

1.1 Історія відеоігор та їх якості зображення

Історія відеоігор розпочалася у середині XX століття, коли кожна гра створювалася окремо для індивідуальних систем. Ці проєкти мали просту графіку та функціонували на ігрових автоматах для яких вони були зроблені. Згодом такі відеоігри стали відомі як аркадні. Вони характеризувалися простим візуальним стилем, обмеженим кількістю кольорів та примітивною анімацією.

Однією з найяскравіших аркадних ігор стала “Pac-Man” (1980) [1], яка продемонструвала, наскільки захопливим може бути ігровий процес навіть за мінімального технічного оснащення. Гравець керував жовтим персонажем, що пересувався лабіринтом, збирав крапки й уникав зіткнення з привидами, або із посиленням міг також і їсти їх. Проста концепція використання обмежених ресурсів зробила “Pac-Man” однією з ікон ранньої епохи відеоігор.

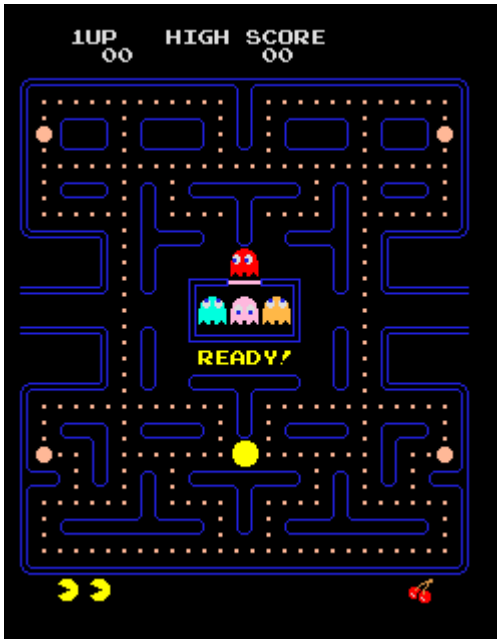


Рисунок 1.1 – Аркадна гра “Pac-Man”

До визначних аркадних відеоігор того часу також належать “Donkey Kong”, “Digger”, “Battle City” та “Mortal Kombat”. Незважаючи на обмеження візуальних засобів, кожна з цих ігор мала власний стиль і механіки, що дозволяли їм виділятися серед численних аналогів. Вони стали джерелом натхнення для цілого покоління розробників, які згодом закладуть основи сучасної ігрової індустрії.

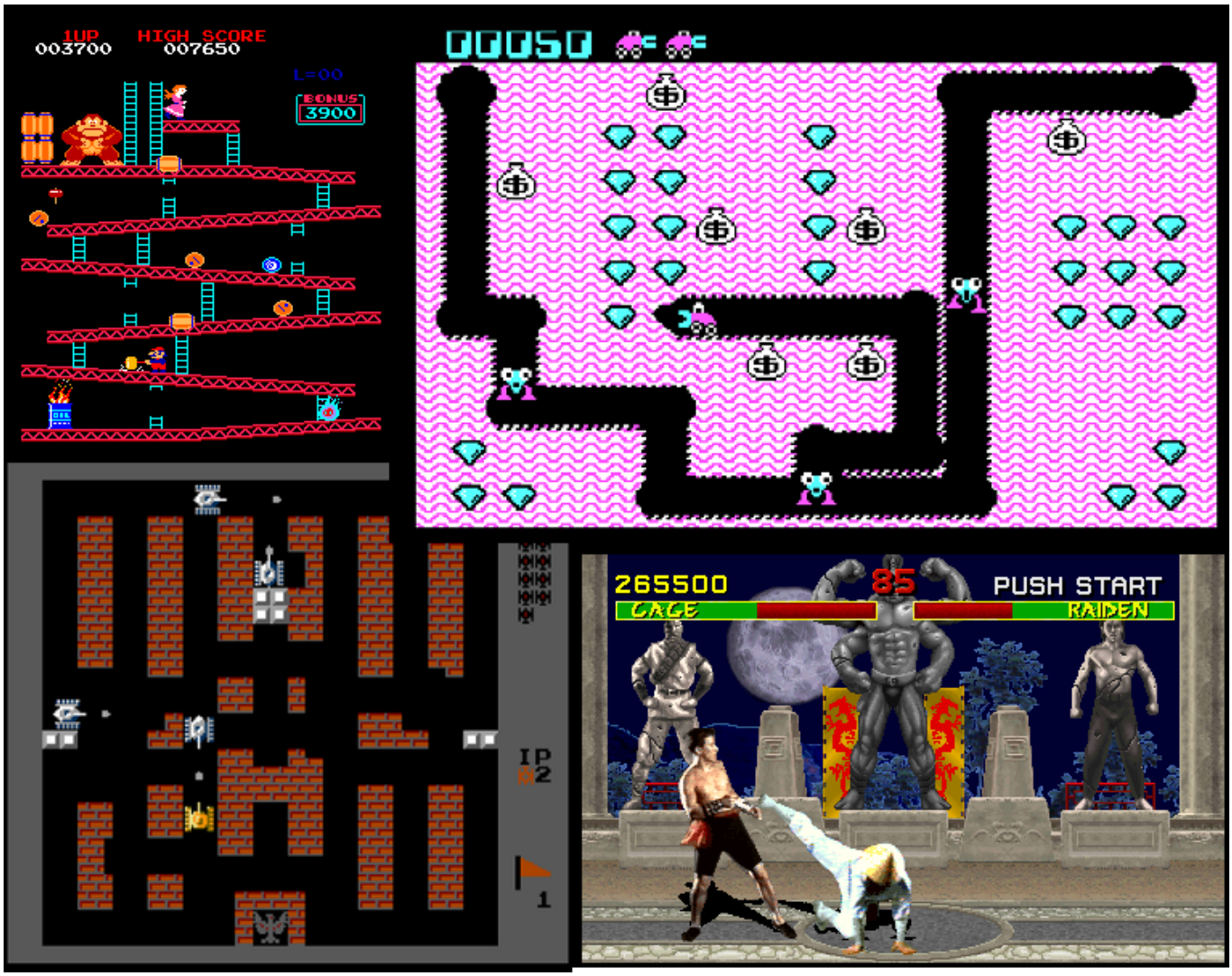


Рисунок 1.2 – Аркадні ігри “Donkey Kong”, “Digger”, “Battle City” та “Mortal Kombat”

У середині 1980-х років із появою 8-бітних ігрових консолей, відбувся суттєвий прорив у розвитку графіки. Зображення стали яскравішими й деталізованішими, а кількість одночасно доступних кольорів значно зросла.

Завдяки картриджам відеоігри тепер можна було грати прямо з дому, що значно розширило їхню аудиторію.

Подальший крок уперед був зроблений із приходом 16-бітної епохи. Ігри цієї генерації демонстрували складніші візуальні ефекти: багатошарові паралакс-фони, більш плавну анімацію та значно вищу роздільну здатність спрайтів. Художники отримали можливість створювати глибокі та живі світи, а ігровий процес став різноманітнішим та насиченішим.

Кардинальні зміни сталися у другій половині 1990-х років із приходом тривимірної графіки. Відеоігри на зразок “Super Mario 64” та “Tomb Raider” відкрили для гравців нові можливості — вільне пересування тривимірним простором, взаємодію з об’єктами та новий рівень занурення у віртуальний світ. Хоча ранні 3D-моделі були відносно простими і містили грубі полігони, саме ця технологічна революція визначила напрямок розвитку відеоігор.

Рисунок 1.3 – Відеоігри “Super Mario 64” та “Tomb Raider”

Сучасні інновації на кшталт трасування променів та HDR-технологій графічна якість у відеоіграх досягла рівня майже фотореалістичного графічного стилю. Сучасні ігри здатні передавати не лише деталі фізичного світу, а й тонкі емоційні стани через світло, тіні та текстури, перетворюючи ігри на справжні мистецькі витвори.

Освітлення й передача тіней відіграють важливу роль не лише у тривимірних, але й у двовимірних відеоіграх. Завдяки правильній роботі із світлом та тінями

навіть 2D-ігри отримують відчуття просторової глибини й об’єму. Освітлення дозволяє підкреслити атмосферу сцени, створити відчуття реалістичності та емоційної насиченості, надаючи статичним зображенням живого та багатошарового характеру. Таким чином, сучасні технології візуалізації впливають на всі жанри та формати ігор, розширюючи їх художні можливості.

1.2 Аналіз рушіїв та вибір оптимального

Насамперед потрібно зрозуміти який рушій підійде для розробки гри, адже від цього буде залежати її структура, якою мовою програмування вона буде написана. Крім того рушій визначає можливості візуалізації та оптимізації.

Для розробки мною було використано рушій Unity, а точніше його сучаснішу версію Unity 6, він дає чисельну кількість переваг порівняно з іншими рушіями, зокрема і в графічному аспекті.

Інструменти розробки: Unity забезпечує модульну структуру та можливість розширення, що дає змогу додати додаткові інструменти. Зокрема за допомогою його asset store можна зручно та без проблем завантажувати придбані модулі.

Графічна база: раніше в Unity URP/HDRP йшли як окремі модулі, та з версією Unity 6 вони почали йти як вбудовані, це забезпечує без лишніх рухів підключити необхідний графічний модуль, що забезпечує, як оптимізацію так і якість зображення та можливість керувати ресурсами, що взаємодіють із графічною складовою гри.

Фізичний рушій: Unity 6 значно вдосконалила фізичний рушій, що забезпечує покращені моделі колізій, точніший розрахунок зіткнень та нові можливості із взаємодією фізичних матеріалів й їх взаємодію із об’єктами. Крім того, підвищено продуктивність симуляції.

Навчальний ресурс: Unity 6 має офіційну документацію по всім аспектам рушія, від створення об’єкту на сцені до підключення офіційних модулів та як

взаємодіяти із ними. Також має велику базу відео-уроків для початківців. Це робить Unity доступним для всіх класів розробників, від початківців до професіоналів[2].

Спільнота: Unity 6 має потужну й активну спільноту, що забезпечує велику кількість користувацьких модулів, ассеті також безмежну кількість відео-уроків по різним аспектам рушія від спільноти що подекуди продуктивніше аніж з офіційних джерел.

Багатоплатфомність: рушій працює на всіх комп'ютерних платформах, таких як: Windows, Linux, MacOS. А самі відеоігри можуть розповсюджуватись окрім цих платформ додатково й на Android, iOS, WebGL та консолі, що забезпечує розробникам видавати свої ігри не лише на якусь одну платформу.

Інтерфейс: порівняно із своїми аналогами Unity 6 має зручний та зрозумілий, для початківця, інтерфейс котрий можна дуже швидко запам'ятати, що забезпечує швидке освоєння в рушії. Крім того інтерфейс можна змінити під себе та свої зручності.

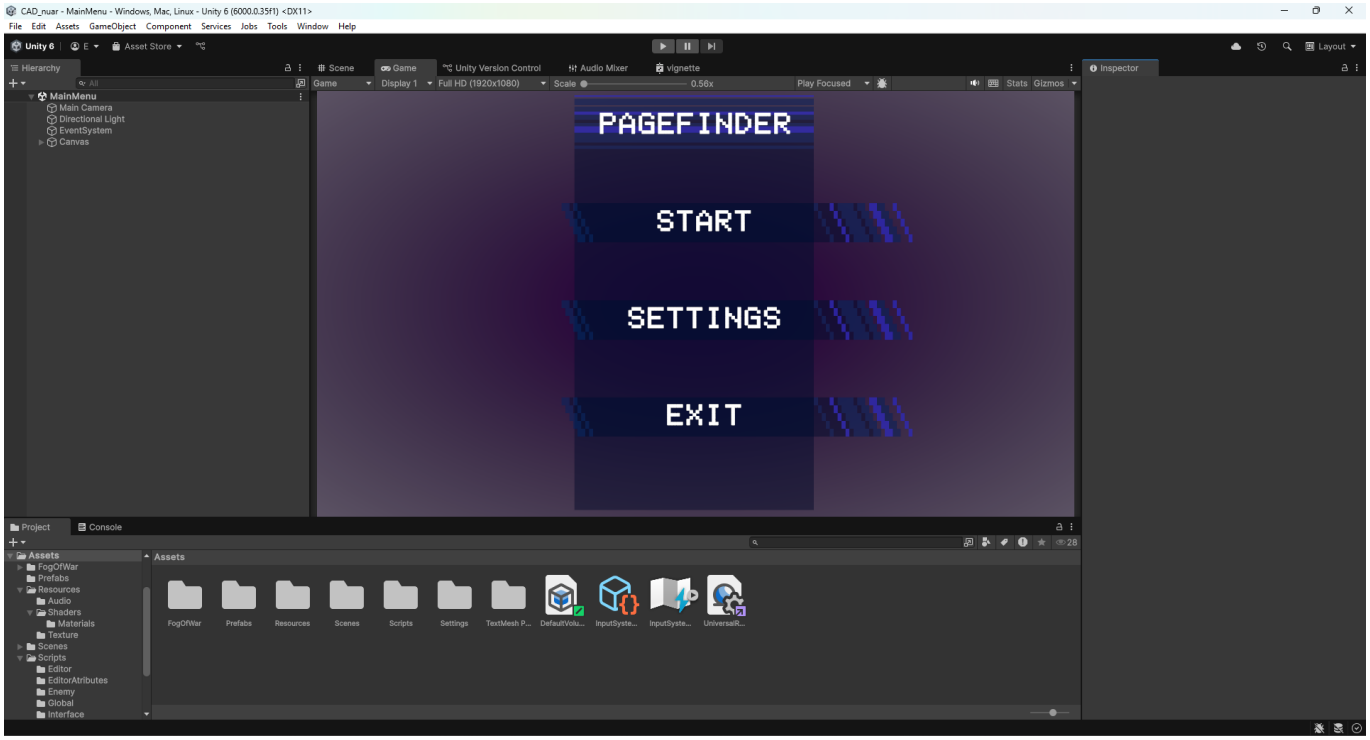


Рисунок 1.4 – Користувацький інтерфейс Unity 6

Поміж інших рушіїв відеоігор можу виділити такі: Godot[3], Unreal Engine[4] та Source 2 [5].

Godot – це рушій з відкритим кодом, що дає йому значну перевагу порівняно з іншими рушіями, про те він не такий гнучкий в налаштуваннях тіней, освітлення та візуальних ефектів. Чудовий вибір для незалежних розробників початківців через його скриптованість [3].

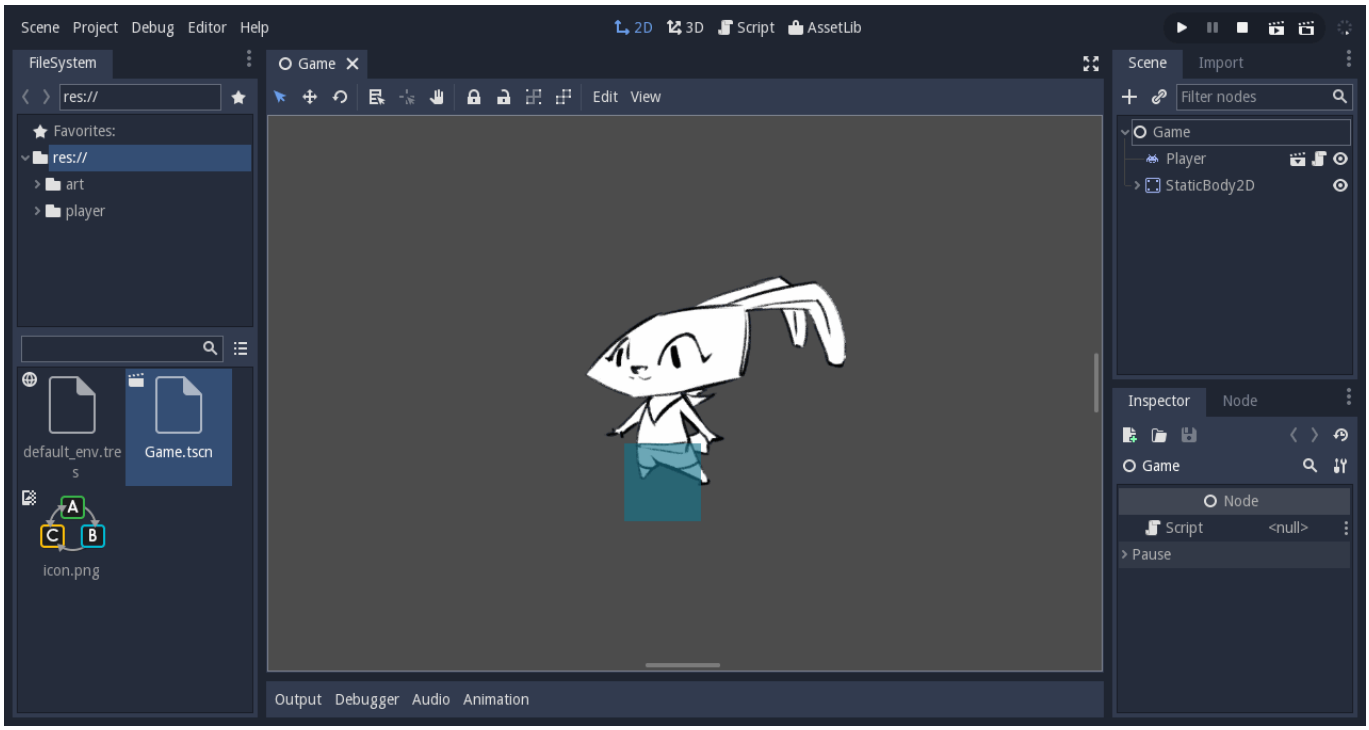


Рисунок 1.5 – Користувацький інтерфейс Godot

Unreal Engine має потужні інструменти як графічні так і для розробки. Чудово підходить для створення ігор із реалістичною графікою. Також серед переваг є підтримка Blueprint що дозволяє створювати ігри без написання коду. Проте в Unreal Engine є вагомі мінуси: його оптимізація та його продуктів, високий поріг входження, складний інтерфейс. Крім того Unreal Engine зосереджений суто на розробці 3D ігор, то ж створювати 2D буде проблематично. Також він підходить для створення фото-реалістичних 3D мультфільмів [4].

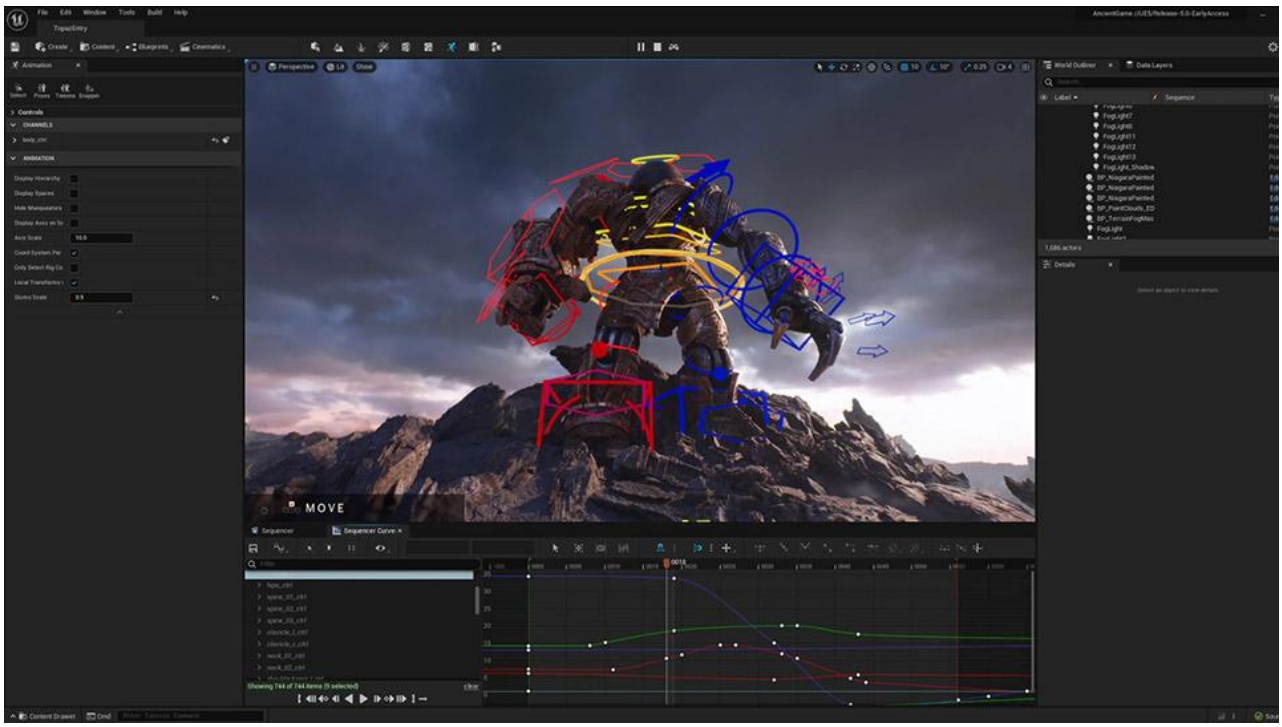
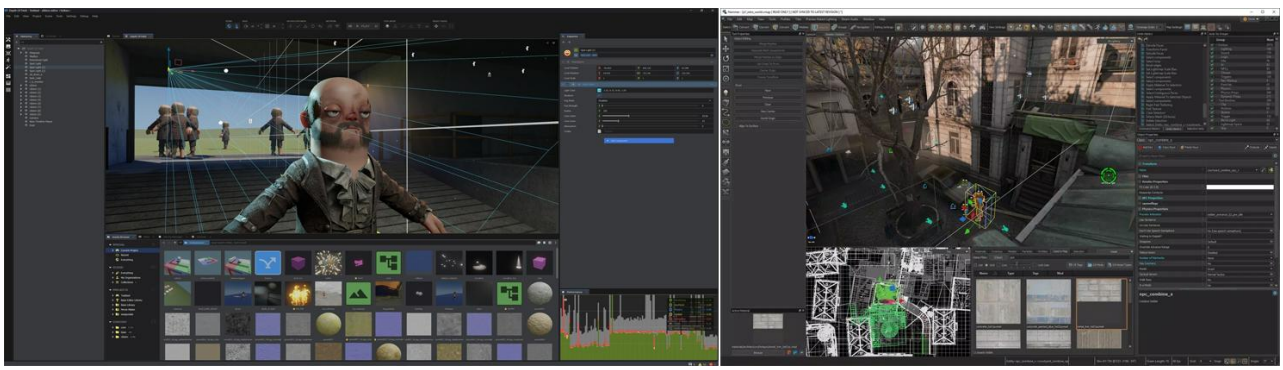


Рисунок 1.6 – Користувацький інтерфейс Unreal Engine

Source 2 новий рушій розроблений компанією Valve має в собі потужні інструменти для: редагування ігрової місцевості, взаємодії фізичних об’єктів, керування погодою та освітленням. Проте він має вагомий мінус – його доступність, на даний момент доступ до Source 2 має сама компанія Valve та декілька розробників які вже довгий час співпрацюють із ними. Дотягнутись до частини інструментарію рушія можна за допомогою гри “S&Box”, котра є на даний момент в закритому тестуванні [6], або в редакторах мап інших продуктів Valve. Рушій використовує мову програмування C# для скриптингу [5].



Зображення 1.7 – Користувацькі інтерфейси “S&Box” та редактора мап

Таким чином рушій Unity 6 є оптимальним варіантом для розробки завдяки своїй доступності, простоті освоєння, низьким навантаженням на систему та широким спектром бібліотек та модулів. Unity 6 надає все необхідне аби створити продукт котрий задовільнить потреби сучасних гравців, та буде підтримувати сучасні тенденції ігроробства.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ВІДЕОГРИ

2.1 Аналіз жанру та стилю відеогри

Після вибору ігрового рушія потрібно визначити жанр та стиль гри. В моєму випадку це має бути 2.5D [12] гра з жанрами: souls-like [7], adventure [8]. Гра, котра буде розроблена, має дати змогу гравцеві досліджувати локації і отримувати корисні предмети, а також має дати певну складність під час боїв. Тож результатом має бути поєднання цих двох жанрів ігор.

Souls-like – жанр ігор який має походження від відеогри “Demon Souls” [9], яка стала матір’ю всіх ігор в цьому жанрі і почала вибудовувати його, далі наслідником стала гра “Dark Souls” [10] котра і закріпила цей жанр, як “Souls-like” [7].

Рисунок 2.1 – Відеоігри “Demon Souls” та “Dark Souls”

Adventure – жанр відеоігор, який сфокусований на дослідженні та взаємодії зі світом, розвитком та взаємодією із персонажами. Прикладом таких ігор може бути “The Last of Us” [11]. У багатьох adventure-іграх ключову роль відіграє сюжет, що веде гравця через емоційно насичені події та складні моральні вибори. Часто геймплей включає вирішення головоломок, дослідження локацій і побудову стосунків із персонажами [8].

Рисунок 2.2 – Відеогра “The Last of Us”

2.5D ігри мають в собі певний шарм та поєднують в собі стилі 3D та 2D ігор. Чудовим прикладом такого стилю є гра “Star Ocean: The Second Story R” [13] котра поєднує в собі 2D спрайти персонажів та 3D світ із піксельною графікою [12].

Рисунок 2.3 – Відеогра гра “Star Ocean: The Second Story R”

Визначившись із жанром та стилем гри, було прийнято рішення створювати проєкт на основі 3D-об’єктів із поступовим додаванням 2D-елементів. Проте процес розробки може викликати певні ускладнення, зокрема у роботі з освітленням, тінями та візуальними ефектами. Крім того, необхідно буде

правильно налаштувати ракурс камери й обертання об’єктів під однаковим кутом, щоб усе виглядало природньо й органічно. Таким чином, проєкт є дещо складнішим у реалізації, однак це рішення дозволяє маніпулювати двовимірними об’єктами як тривимірними. В результаті буде отримамано ізометричний вид камери та характерний 2.5D-стиль гри.

2.2 Використання шейдерів рушія в відеогрі

Насамперед потрібно детальніше розглянути що ж таке шейдер та зрозуміти різницю між вбудованими шейдерами в Unity та URP/HDRP.

Шейдери – це програми одного із ступенів графічного конвеєра, що використовується в тривимірній графіці для визначення остаточних параметрів об'єкта чи зображення. Вона може містити в собі довільної складності опис поглинання та розсіювання світла, накладення текстури, віддзеркалення і заломлення, затінення, зміщення поверхні і ефекти пост-обробки [14].

Також шейдери поділяються на три типи:

- Вершинні шейдери (Vertex Shader) – оперують даними вершин багатогранників: координати вершин в просторі, текстурні координати, тангенс-вектор, вектор бінормалі та вектор нормалі. Даний тип шейдерів може бути використаний для видового та перспективного перетворення вершин, розрахунку освітлення, генерації текстурних координат і тому подібне.
- Геометричні шейдери (Geometry Shader) – порівняно із вершинними шейдерами, геометричні можуть обробити не лише одну вершину, а й цілий примітив. Він може обробити до шести вершин трикутного примітиву. Крім того геометричні шейдери здатні генерувати примітиви в реальному часі, при цьому не залучаючи центрального процесора.

- Фрагментні шейдери (Pixel Shader) – використовуються на останній стадії графічного конвеєра для формування фрагмента зображення. Під фрагментом зображення вважається піксель, котрому поставлено у відповідність деякий набір атрибутів: колір, глибина, текстурні координати.

Різниця між вбудованими шейдерами Unity та URP/HDRP є досить суттєвою. Вбудована система дозволяє створювати власні шейдери, однак вона обмежена в можливостях оптимізації. Основна оптимізація тут зводиться до налаштування якості тіней та освітлення. Через це при низьких графічних налаштуваннях можуть з’являтися артефакти, такі як фіолетові відблиски чи тіні з різко окресленими піксельними краями.

На відміну від вбудованих шейдерів, URP та HDRP вже мають вбудовану оптимізацію та сучасні графічні можливості. Вони дозволяють створювати більш складні візуальні ефекти, наприклад, візуальні ілюзії, де в кубі розміром 1x1x1 можуть "поміститися" кілька будівель з різною формою. Крім того, URP/HDRP можна гнучко налаштовувати під потреби проєкту: змінювати дальність промальовки тіней залежно від графічних налаштувань, вмикати згладжування тіней або застосовувати ефекти пост-обробки.

Саме з причин оптимізації URP/HDRP стали вбудованими шейдерами в Unity 6. До того ж шейдери URP та HDRP побудовані на фреймворку SRP, через що зручно та легко можна налаштовувати їхню поведінку під конкретні задачі: додавати власні ефекти та забезпечити кращу масштабованість проєкту завдяки низькорівневому доступу до процесу рендерингу. Крім того нові вбудовані шейдери мають в собі візуальний редактор, через що створювати шейдери стало набагато зручніше за зрозуміліше.

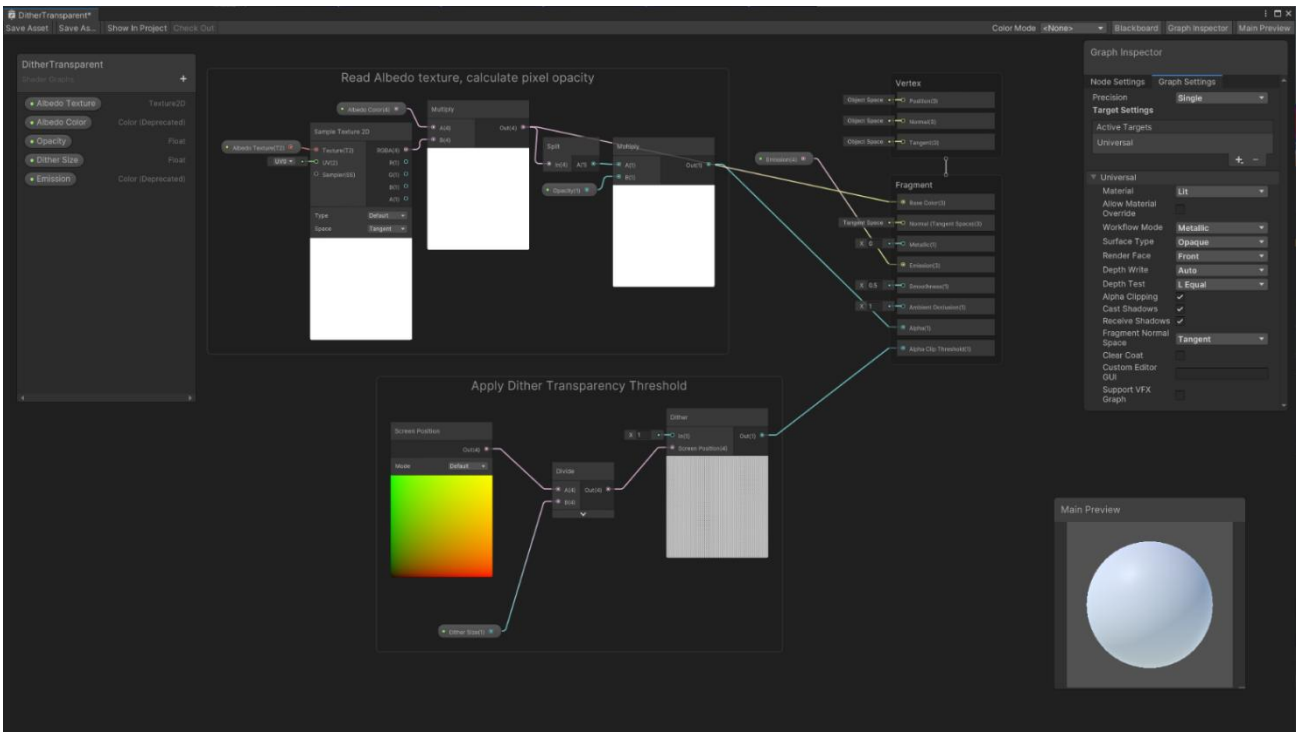


Рисунок 2.4 – Візуальний редактор шейдерів URP/HDRP

2.3 Проєктування відеогри

Проєктування відеогри припало на час проведення “Global Game Jame 2025” [15], тому основою гри стала тема протидії дезінформації. Після визначення теми гри, я приступив до формування основних функцій відеогри. Так як термін створення був обмежений в часі, було вирішено зробити гру з трьох коридорів. Основною механікою гри повинна бути бойова система. Основний процес буде полягати в підбиранні записок, взаємодії із НІПами (Не Ігровий Персонаж). Кожен коридор характеризує складність вибору, так як в першім коридорі легше зрозуміти де інформація правдива, а де брехня чи просто слух, і так з кожним із коридорів буде важче. Кінцевою метою цього навчального рівня буде: перемогти дезінформацію всередині голови НІПа, який нам постійно буде зустрічатись на шляху. Основним аспектом гри є записки, так як на кожній із них є певна інформація котра підтверджує слова НІПа, або спростовує їх.

Реалізація системи ефектів:

- Записки з правдою: збільшують максимальну кількість здоров’я в гравця, та зменшують його в ворогів.
- Записки з слухами: не впливають на гравця чи ворогів, про те можуть дати підказку де є правда, а де брехня.
- Записки з брехнею: зменшують максимальну кількість здоров’я в гравця, та збільшують його в ворогів.
- Також система ефектів дає можливість в майбутньому додати інші предмети з різними ефектами, що робить її гнучкою.

Реалізація бойової системи:

- Удар зброєю ближнього бою та постріл зброєю дальнього бою.
- Можливість швидше пересуватись та використовувати ривок.
- Заряди енергії: заряди енергії потрібні для того, аби запускати снаряд зброєю дальнього бою, або застосовувати бар’єр чи відновлювати втрачене здоров’я. Енергія крутиться орбітою довкола гравця.
- Відновлення здоров’я: витрачає поточну кількість енергії та відновлює здоров’я за такою формулою: кількість енергії * значення для відновлення.
- Бар’єр: витрачає поточну кількість енергії та збільшує абсорбцію шкоди за такою формулою: кількість енергії * значення для абсорбції. Додатково бар’єри крутяться швидше від кількості використаної енергії та значення абсорбції.

Реалізація ворогів:

- Вороги поділяються на три типи: ближній бій, дальній бій, мішаний тип бою.
- Вороги мають можливість, або ухилитись від атаки гравця, або заблокувати. При блокуванні атаки гравець буде збитий з пантелику, та не зможе рухатись протягом однієї секунди. Також вороги мають логіку відступу якщо це дальній тип, та логіку нападу якщо це ближній або мішаний тип.

- Коли вороги стають довкола гравця, аби не накладатись на одному місці в них є логіка оточення, тому вони будуть намагатись не просто встати і бити гравця, а й оточити його (працює лише для ближнього типу ворогів).
- Вороги мають радіус, в якому вони почнуть помічати гравця та окремий радіус в якому вони почнуть помічати снаряд, чи атаку, аби приступити до дії ухилення чи захисту.

Реалізація шейдерів:

- Створення ефекту віньєтки.
- Затемнення між переходами по коридорах.
- Сітка текстур для об’єкту підлоги.
- Туман війни.

Однією із важливих складових гри є подання сюжету через взаємодію із оточенням, що є частиною adventure жанру, та складністю боїв, що стало частиною souls-like жанру. Таким чином у проєктуванні відеогри я зміг поєднати два жанри.

Також було створено діаграми варіантів використання та послідовностей, що дозволяє візуально відобразити сценарії взаємодії гравця із системою гри. Та додано таблиці функціональних та нефункціональних вимог, задля зручного аналізу потреб гри

Таблиця 2.1 – Основні функціональні вимоги

Код	Назва функції	Опис
FR-01	Використання активних предметів (записок)	Гравець може підбирати та використовувати записки з правдою, брехнею або слухами
FR-02	Вплив записок на характеристики	Записка з правдою → +HP гравцю, -HP ворогу; з брехнею – навпаки; слух – підказки
FR-03	Атака зброєю ближнього бою	Гравець може наносити удар зблизька
FR-04	Стрільба з дальньої зброї	За витрату енергії запускається снаряд

Продовження таблиці 2.1

FR-05	Використання ривка	Гравець може робити швидкий ривок для уникнення атак
FR-06	Відновлення здоров’я	Енергія витрачається на лікування: енергія × коеф. відновлення
FR-07	Бар’єр	Захист: енергія × коеф. абсорбції; швидкість обертання залежить від енергії
FR-08	Вороже ШІ	Вороги реагують на дії гравця (ухилення, блок, атака)
FR-09	Логіка оточення	Вороги ближнього бою намагаються не стояти в одному місці, а оточити гравця
FR-10	Радіуси виявлення	Вороги провокуються на гравця в певнім радіусі

Таблиця 2.2 – Нефункціональні вимоги

Код	Тип	Вимога
NFR-01	Продуктивність	Гра має працювати з FPS не менше 60 на середніх ПК
NFR-02	Зручність використання	Інтерфейс інтуїтивно зрозумілий, керування – стандарт WASD + миша
NFR-03	Надійність	Гра не повинна зависати при одночасному використанні кількох механік
NFR-04	Гнучкість	Система ефектів має підтримувати легке додавання нових предметів
NFR-05	Естетика	Графічні ефекти відповідають стилю гри, посилюють атмосферу дезінформації

Рисунок 2.5 – Діаграма варіантів використання гравця

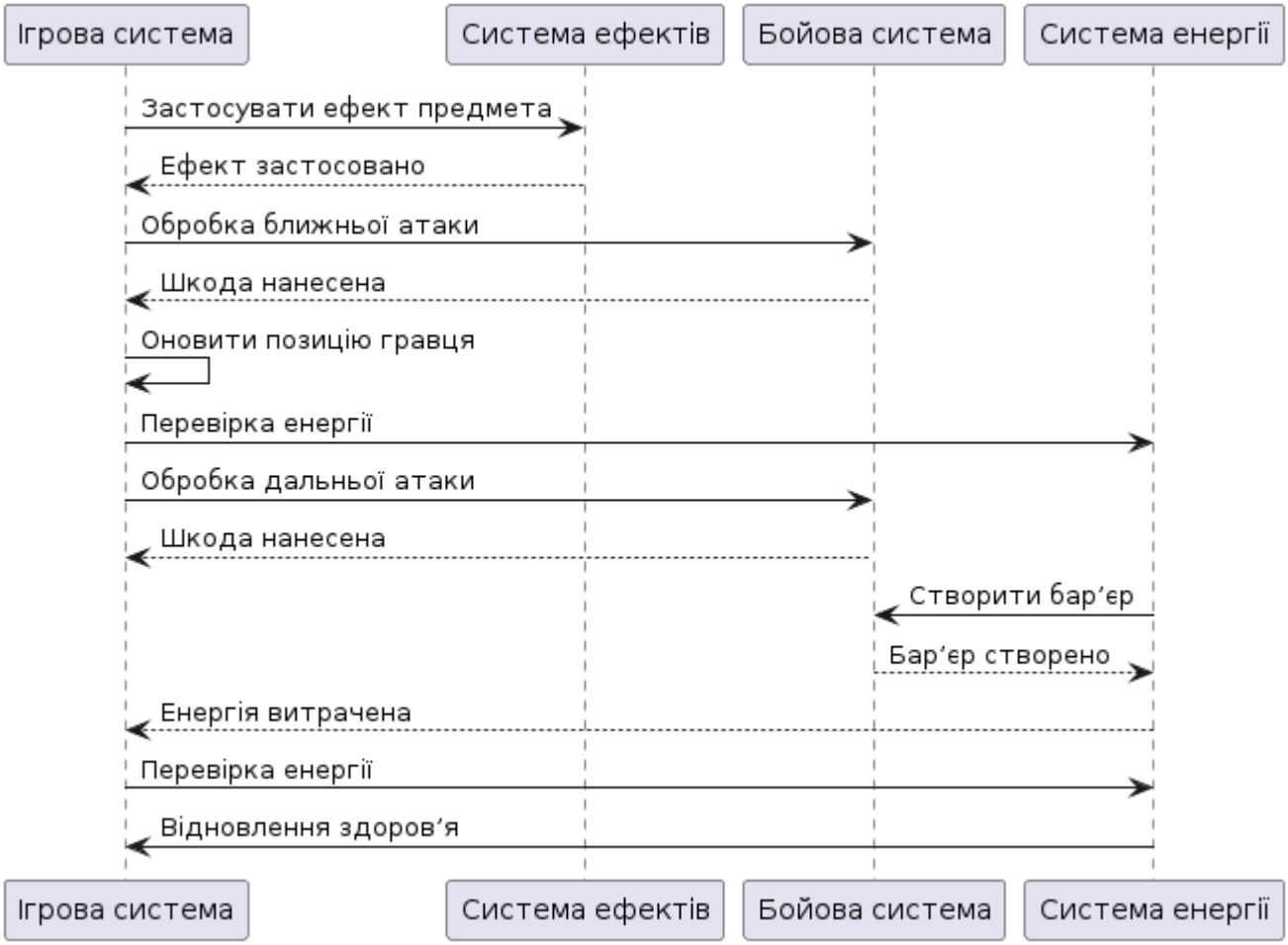


Рисунок 2.6 – Діаграма послідовностей

2.4 Підготовка до розробки

Після вибору рушія для розробки та детального опрацювання проєктування, наступним кроком є підготовка допоміжних інструментів, які використовуватимуться протягом усього процесу створення ігрового продукту.

Так як реалізація туману війни є складною задачею, мною було вирішено придбати ассет із офіційного магазину Unity.

Pixel-Perfect Fog Of War надає гнучкий функціонал налаштування туману, підтримує всі рендер-пайплайни, має високу продуктивність, необмежений в масштабі сцени та не споживає пам'яті підчас виконання [16].

Fog Of War Quick Start Guide

First of all, thank you for choosing Pixel-Perfect Fog Of War. **As a small creator, your patronage makes a big difference.** If you're a fan of the system, **I would also greatly appreciate a review on the asset store, as they help me out a ton :)**

Рисунок 2.7 – Гугл документ із інструкціями використання

Таким чином Pixel-Perfect Fog Of War є доцільним рішенням використання туману війни в проєкті, завдяки його широкій сумісності, та оптимізації. Це дає змогу робити якісний продукт із меншим навантаженням на пристрої гравців, що є важливим чинником для успішної розробки та реалізації гри.

2.5 Налаштування середовища розробки

Після проєктування відеогри та вибору допоміжних інструментів розробки, потрібно обрати версію Unity в якому все буде створюватись, обрати необхідний рендер-пайплайн та створити проєкт гри.

Спершу потрібно про інсталювати лаунчер з якого буде запускатись рушай, цим лаунчером є Unity Hub [17].

Для розробки відеогри була обрана версія Unity 6 (6000.0.35f1), котра на момент написання кваліфікаційної роботи є найновішою та найстабільнішою.

При інсталяції рушія потрібно обрати модулі, аби мати можливість запустити відеогру на різних платформах: комп’ютери, консолі, тощо.

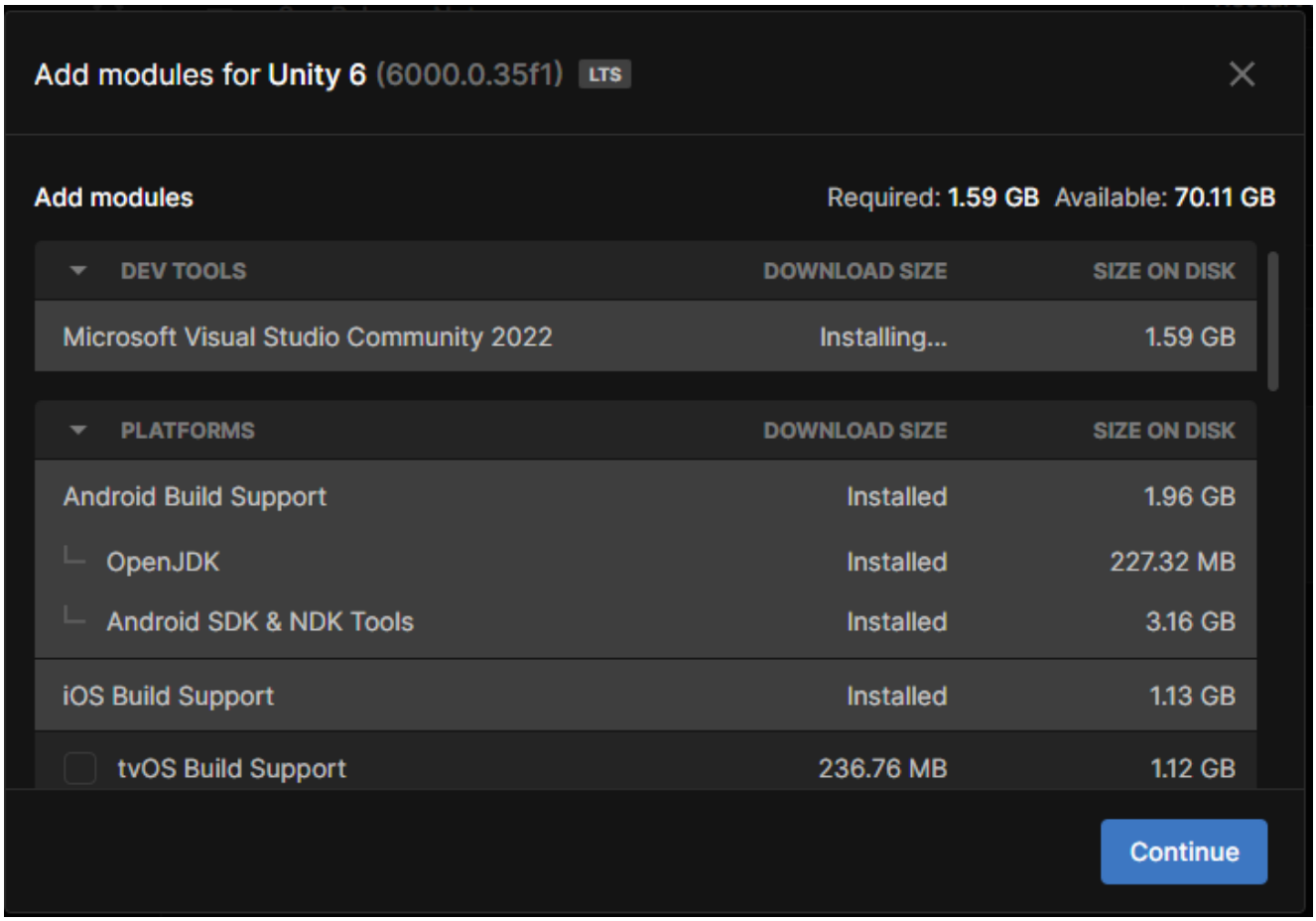


Рисунок 2.8 – Інстальовані необхідні модулі

Після інсталяції модулів настає етап вибору на якому рендер-пайплайні буде створено проєкт. Так, як HDRP використовується для більш якісних тіней і більшого розширення текстур, мною було прийняте рішення використовувати URP, котрий буде надавати не таку якісну картинку шейдерів, проте буде легшим для слабких пристроїв. Крім того HDRP працює лише для 3D проєктів, що частково мені не підходить.

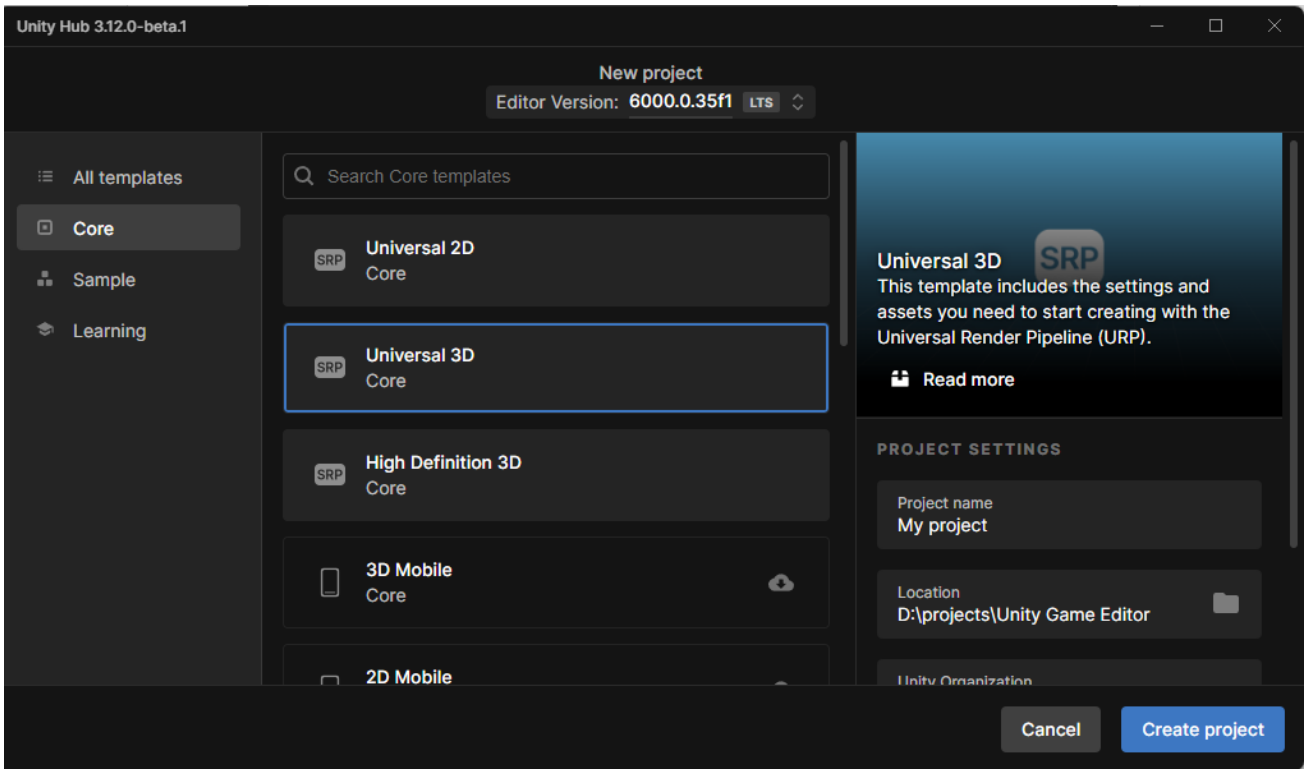


Рисунок 2.9 – Вибір рендер-пайплайну при створенні проєкту

Після створення проєкту потрібно про інсталювати допоміжний ассет туману вйни. Для того аби його інсталювати, після придбання, потрібно: на сторінці ассету натиснути кнопку “Open in Unity” – погодитись відкрити “Unity Editor”, після чого потрібно буде імпортувати його.

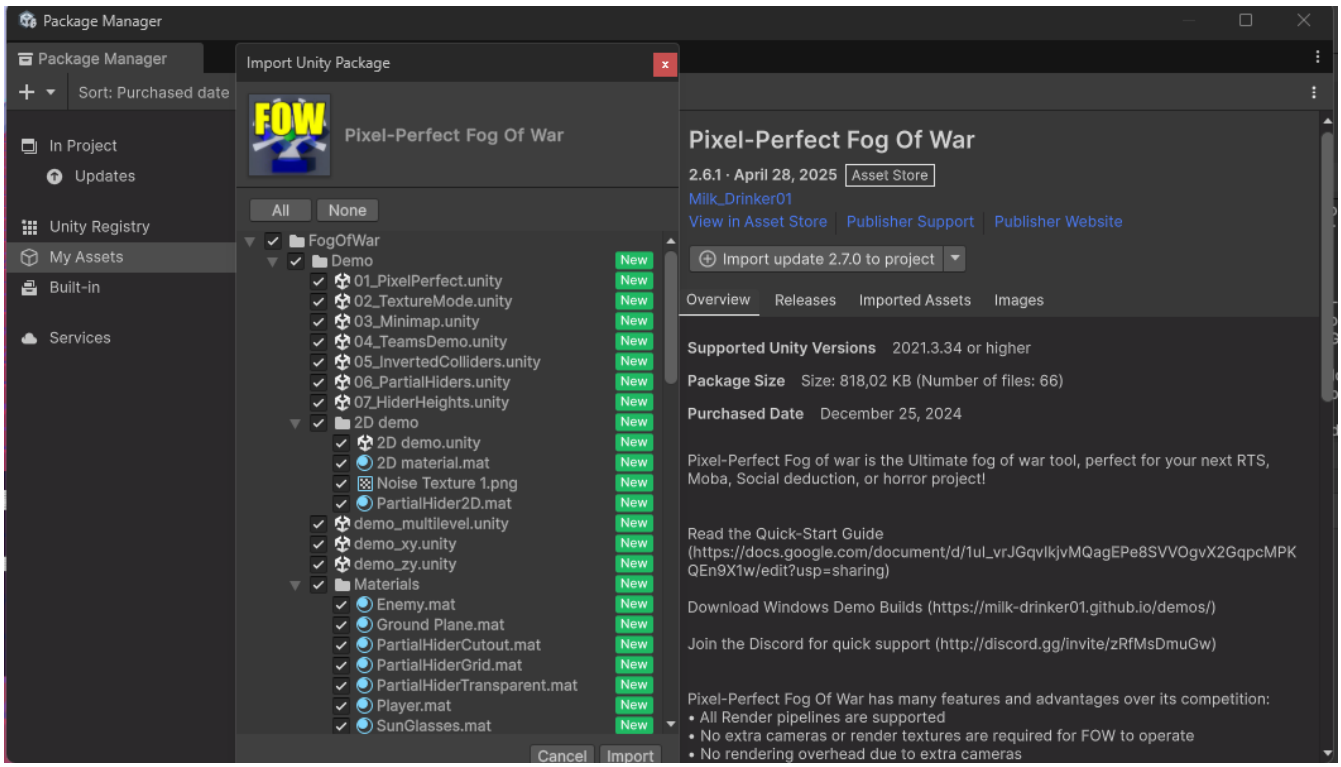


Рисунок 2.10 – Інсталяція “Pixel-Perfect Fog of War” в проєкт

Далі, аби туман війни працював на URP потрібно зайти в теку із всіма компонентами та обрати файл з назвою “FOW-URP” та імпортувати все запропоноване.

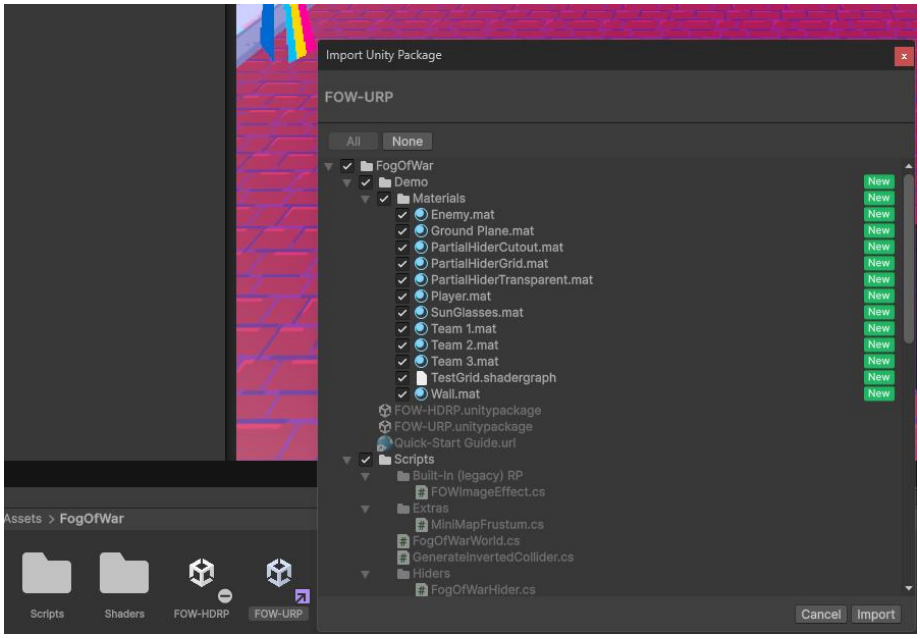


Рисунок 2.11 – Імпортування підтримки URP

Після цього, згідно з інструкцією, потрібно зайти в налаштування рендер-пайплайну, та додати в комірку “Renderer Feature” компонент “Fog Of War Render Feature” та активувати підтримку глибини текстур в “URP Asset”.

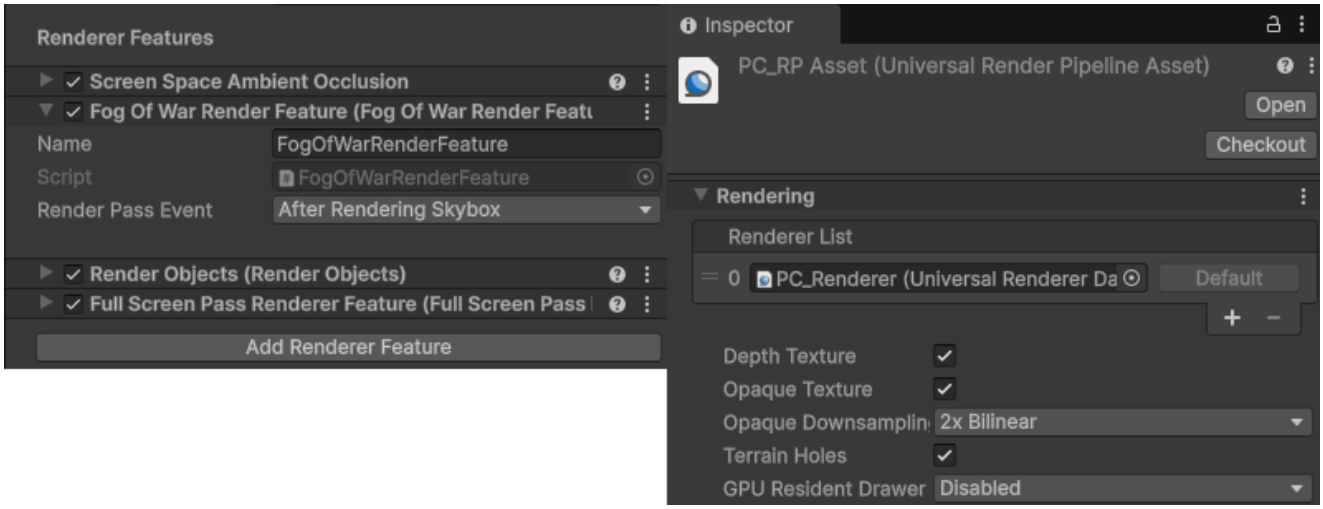


Рисунок 2.12 – Налаштування роботи взаємодія із “Renderer Feature” та “URP Asset”

Тепер ассет туману війни для URP шейдерів налаштований і готовий до того аби його використовувати.

2.6 Проєктування архітектури відеогри

Архітектура програмного забезпечення є ключовим елементом при його створенні, адже саме вона задає основу для подальшої розробки. Від якості архітектурних рішень залежить, наскільки система буде стабільною, масштабованою та легкою у супроводі. У контексті розробки ігор це має особливе значення, оскільки відеогра складається з великої кількості взаємозалежних компонентів, які мають працювати узгоджено та без збоїв.

Процес проєктування починається зі структурування ключових частин програми та моделювання взаємодії між ними. Гра повинна включати в себе низку

важливих функціональних блоків: систему здоров’я, логіку створення зарядів, систему ефектів, та предметів, головне меню, вікно налаштувань, логіку поведінки противників, систему діалогів та інше.

Маючи загальне бачення архітектурної побудови, далі доцільно перейти до аналізу основних класів, які формують ядро відеогри, їхніх властивостей і методів, що забезпечують функціональність гри.

Почати варто із глобальних класів котрі будуть належати або взаємодіяти, як із гравцем так і ворогами, чи впливати на всю сцену.

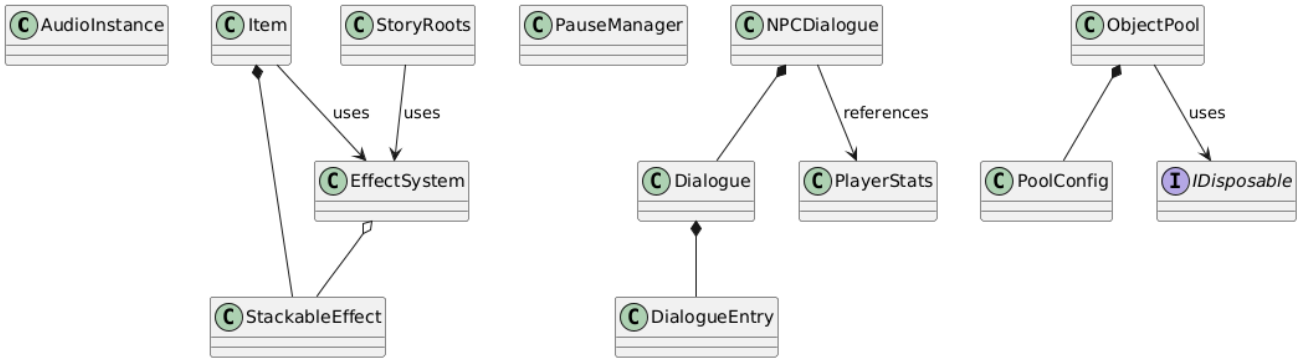


Рисунок 2.13 – Діаграма класів Global

Тепер можна детальніше розглянути деякі класи теки Global.

Клас AudioInstance дає змогу викликати звукові файли не створюючи окремі компоненти в класах котрі викликають звук, в додачу до цього це зручне місце для менеджменту всіх звукових файлів що вже використовуються.

Рисунок 2.14 – Клас AudioInstance

Розглянемо два, тісно пов’язаних між собою класи Item та StackableEffect. Клас Item дає змогу гравцеві підбирати предмет, що лежить на землі. Клас міг би знаходитись в теці Player, проте через те, що предмети та ефекти це більш глобальна взаємодія із світом цей клас знаходиться в теці Global.

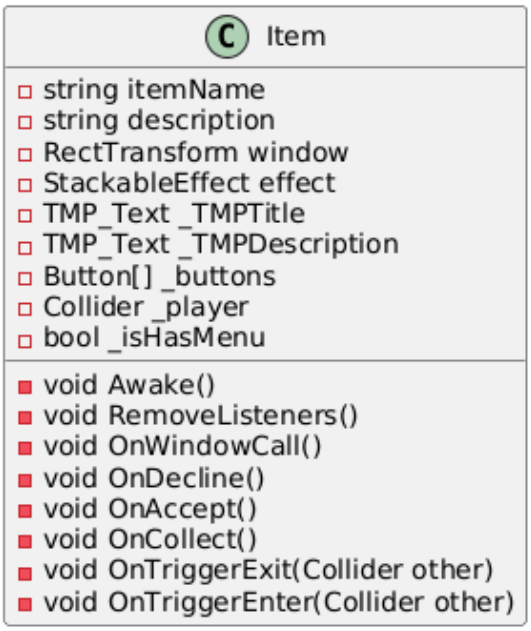


Рисунок 2.15 – Клас Item

Клас StackableEffect в свою чергу вказує на ім’я ефекту, скільки разів він може накладатись один на одного, та вплив котрий він надає предмету.

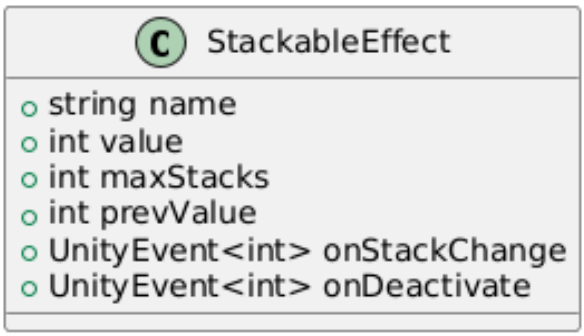


Рисунок 2.16 – Клас StackableEffect

Клас StoryRoots дає можливість пройти гру на різні кінцівки. Ці кінцівки залежать від кількості правильно підібраних записок.

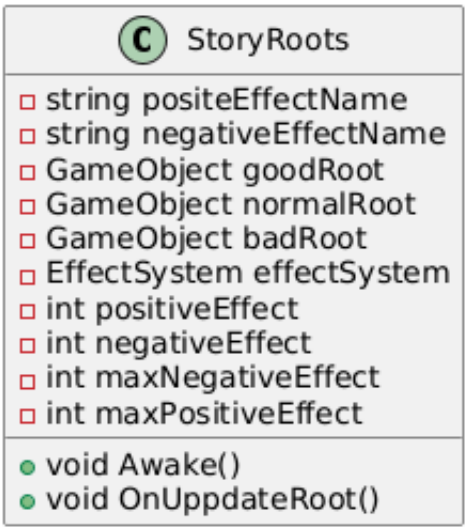


Рисунок 2.17 – Клас StoryRoots

EffectSystem – клас котрий надає логіку і сенс класам StoryRoots, Item та StackableEffect. Взаємодіє напряму лише із гравцем, проте може впливати на ворогів також з тіла об’єкта гравця, що дає більше варіативності використання для негативних ефектів.

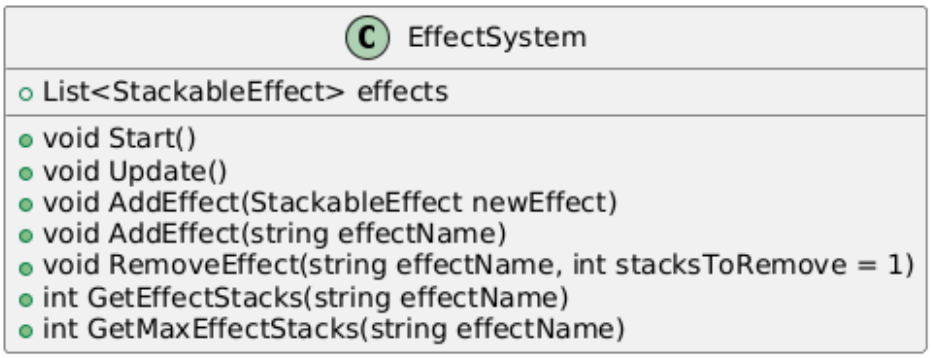


Рисунок 2.18 – Клас EffectSystem

Клас NPCDialogue – потрібен для того аби в діалоговому вікні виводився текст із вмістом діалогу при контакті об’єкту гравця із об’єктом НІПа, чи зоною в якій має виникнути діалог.

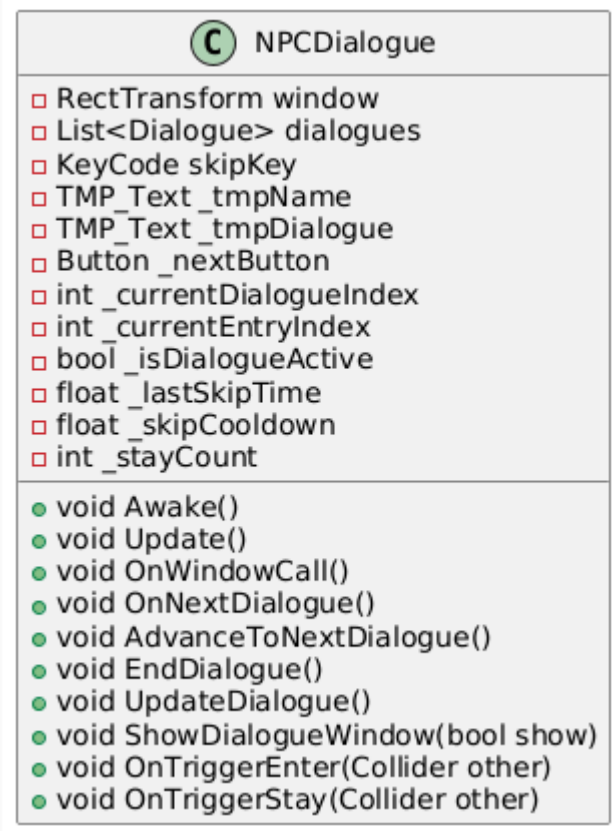


Рисунок 2.19 – Клас NPCDialogue

DialogueEntry – це клас котрий необхідний для того аби заповнити текст діалогу, та дати можливість діалогу впливати на світ за потреби.

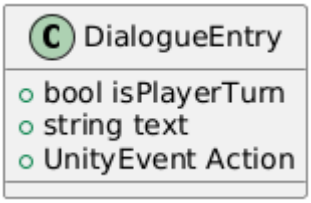


Рисунок 2.20 – Клас DialogueEntry

Dialogue має основну інформацію для передачі в клас NPCDialogue, такі як ім’я гравця, ім’я НІПа з яким той говорить, та одну із сторінок діалогу.

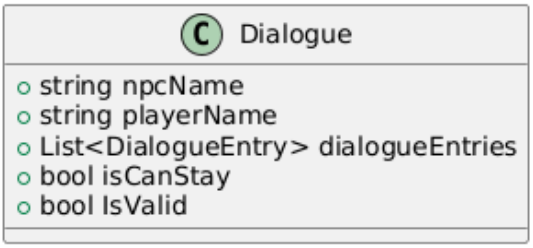


Рисунок 2.21 – Клас Dialogue

Після огляду теки Global, можна перейти до огляду однієї із важливих тек у грі, до теки гравця. Розглянемо його діаграму класів та розглянемо деякі класи детальніше.

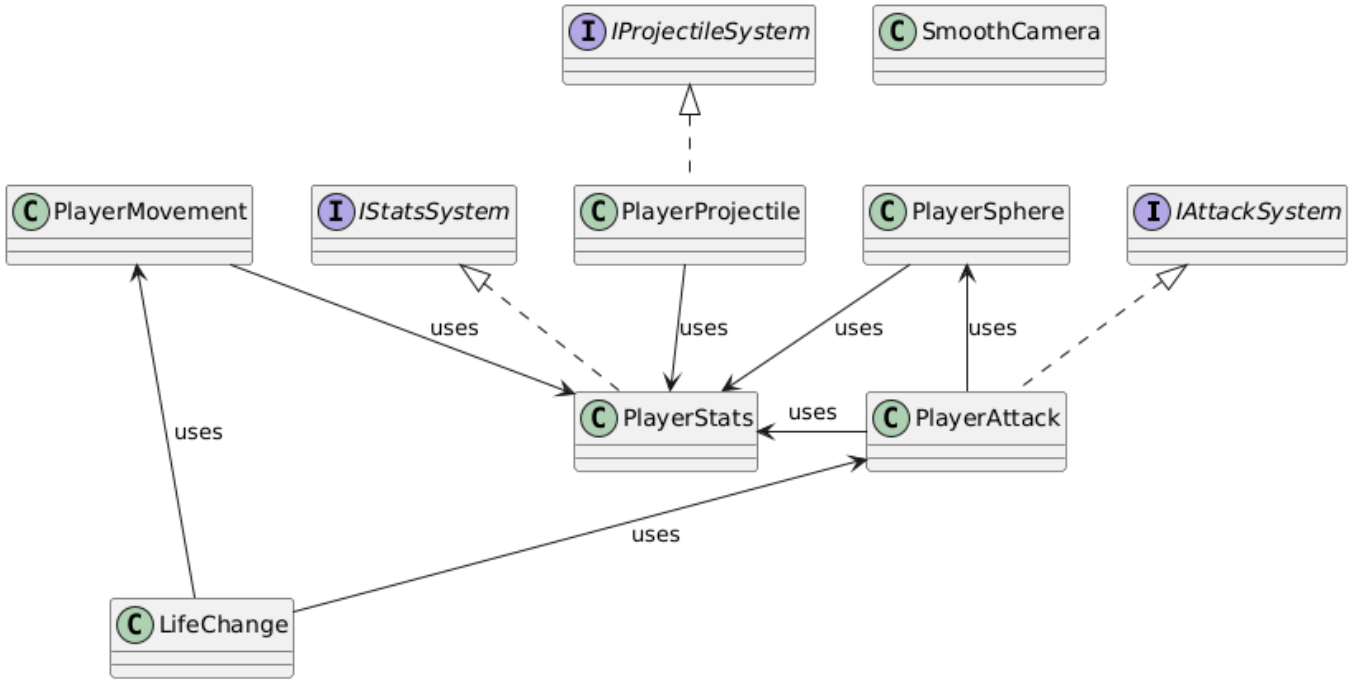


Рисунок 2.22 – Діаграма класів Player

Розглянемо клас PlayerSphere, що є класом для оберту зарядів енергії довкола орбіти гравця, та відповідає за логіку реалізації цих зарядів: застосування бар'єру, застосування відновлення здоров'я та вистріл зарядом в напрямку курсору.

Рисунок 2.23 – Клас PlayerSphere

Клас PlayerStats містить в собі інформацію про характеристики гравця, такі як от здоров'я, швидкість руху, значення бар'єру і т.д. Також цей клас додає

додаткові ефекти станів гравцю, такі як сповільнення, знерухомлення або розброєння. Також додатково розглянемо пов’язаний клас StatusEffectData.

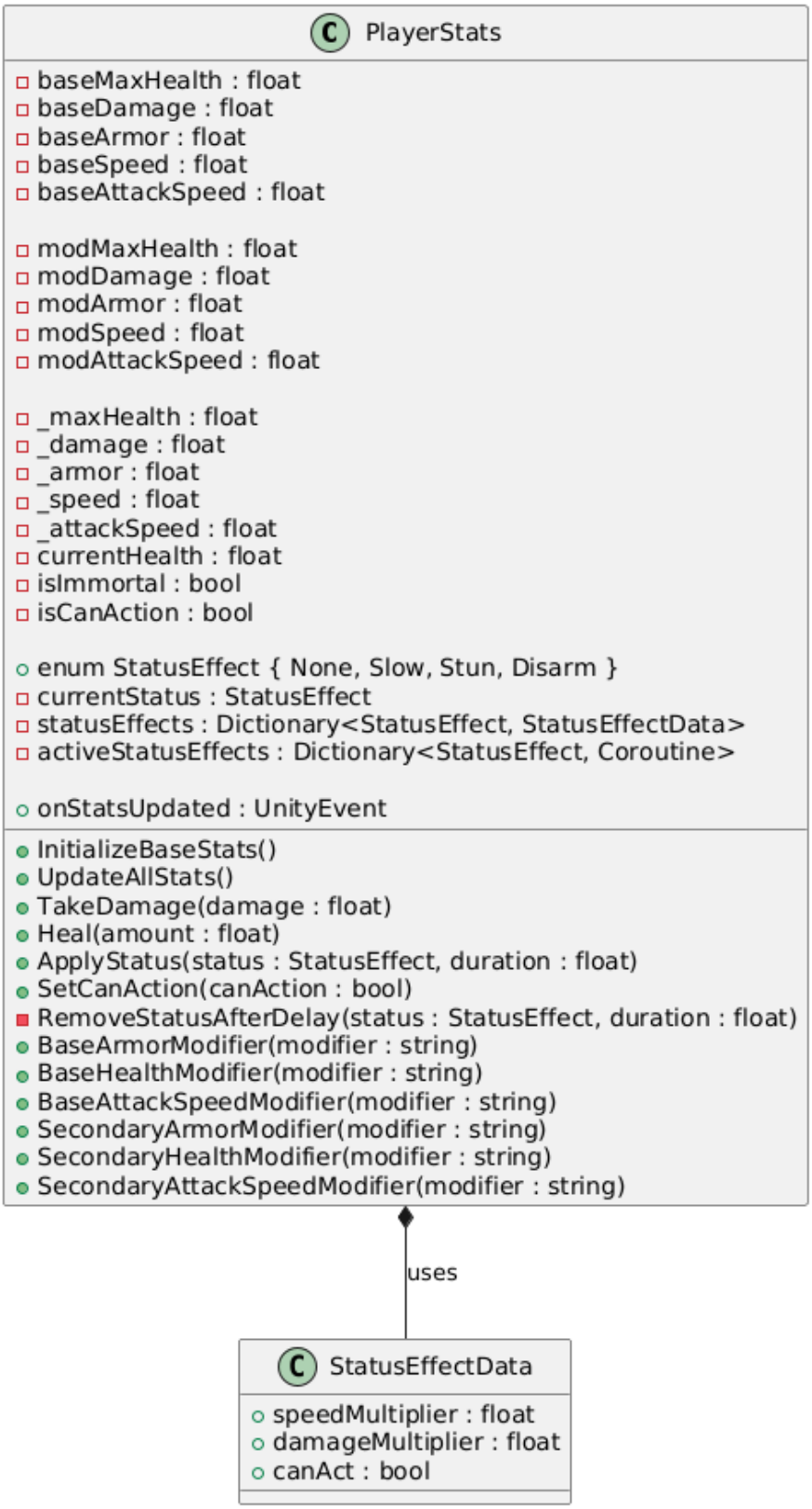


Рисунок 2.24 – Класи PlayerStats та StatusEffectData

На останок в теці Player розглянемо клас PlayerMovement котрий відповідає за переміщення гравця. Цей клас також дає змогу використовувати ривок чи пришвидшувати переміщення гравця.

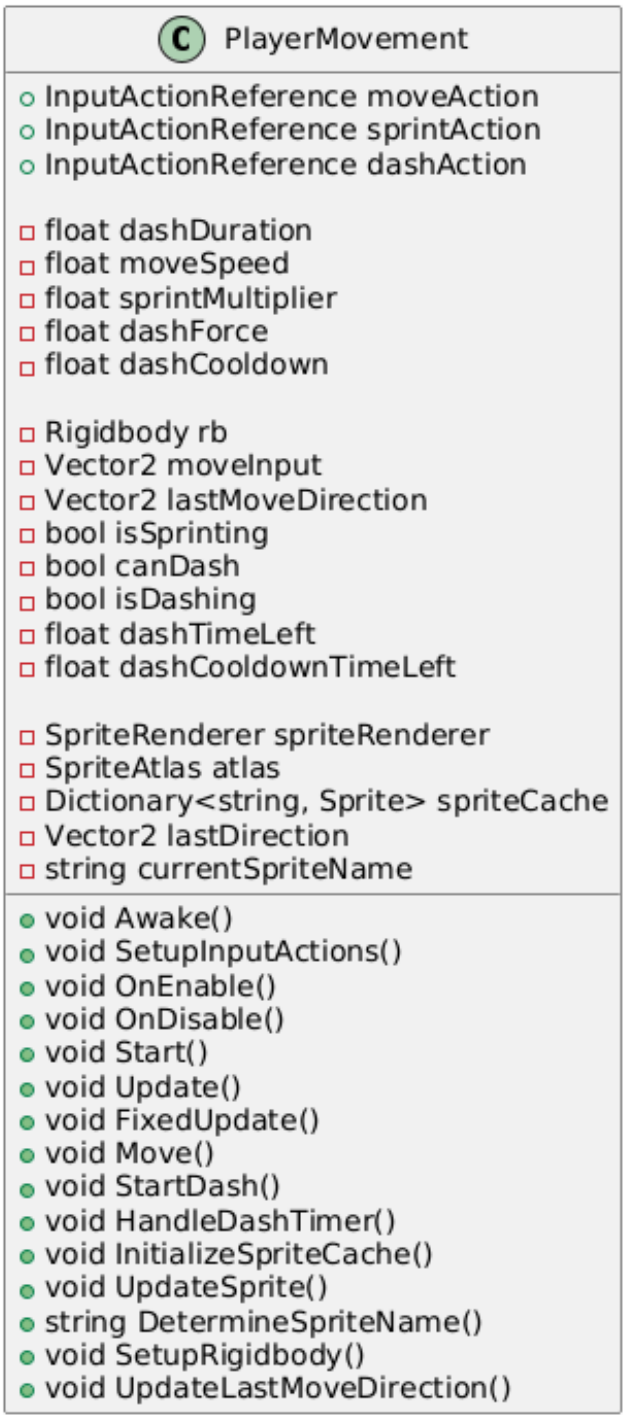


Рисунок 2.25 – Клас PlayerMovement

Розглянемо діаграму класів Enemy, та детальніше подивимось на два класи: EnemyStats та EnemyAI.

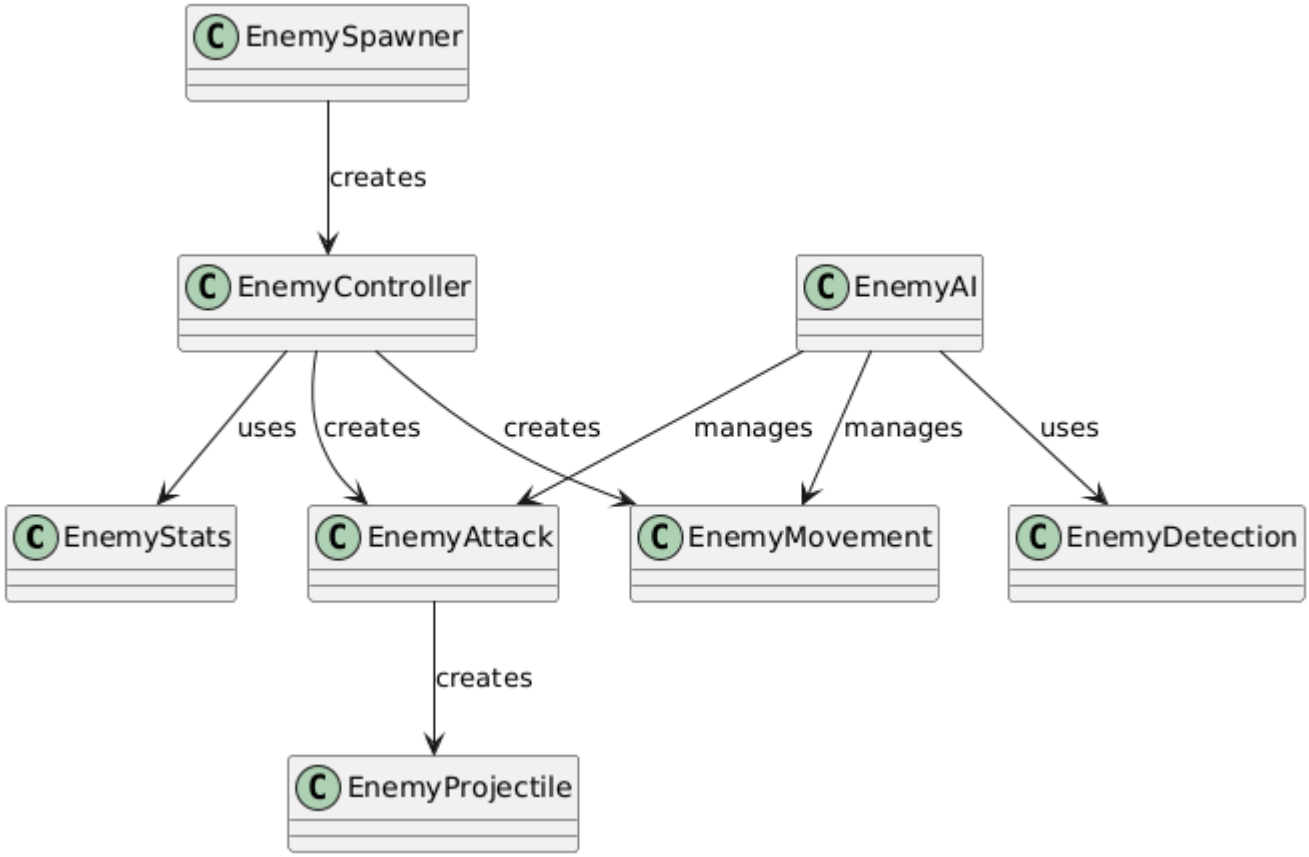


Рисунок 2.26 – Діаграма класів Enemy

Тепер можна розглянути клас EnemyStats. Він може здатись схожим до класу PlayerStats, проте він має суттєву різницю – відсутність додаткових модифікаторів. Також клас характеристик ворога, як і клас характеристик гравця має в собі клас ефектів станів.

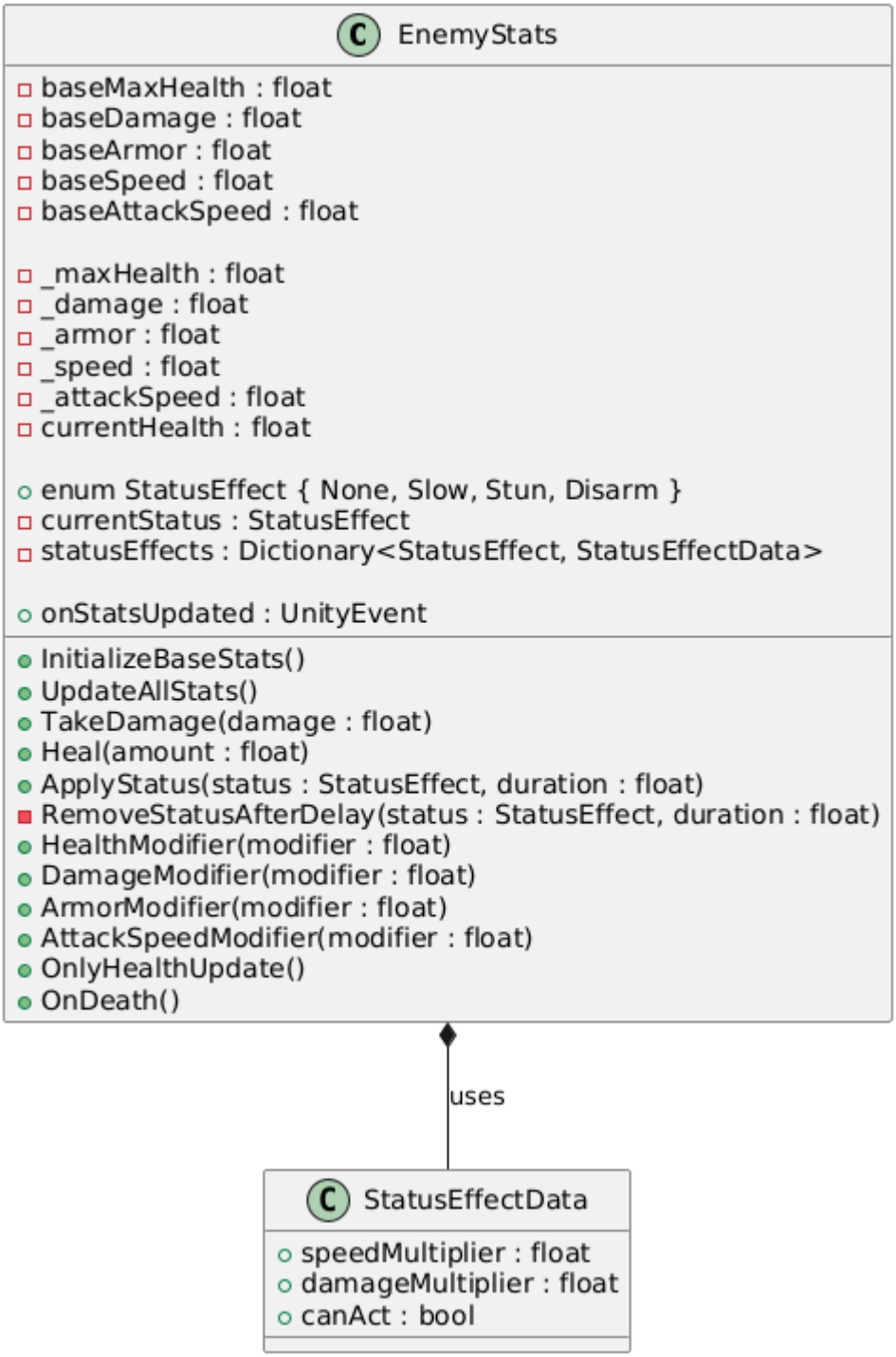


Рисунок 2.27 – Класи EnemyStats та StatusEffectData

Розглядаючи клас EnemyAI можна зрозуміти наскільки важко робиться примітивна логіка ворогів і наскільки це важкий процес створення ворога що буде мати логіку та поведінку. В цьому випадку ворог має чотири типи поведінки та три типи.

- Типи:
    - Melee – ближній тип.
    - Ranged – дальній тип.
    - Both – мішаний тип.
- Поведінка:
    - Idle – очікування.
    - Pursuing – переслідування.
    - Evading – ухилення, про те це вказує що ворог буде мати вибір: Заблокувати або ухилитись.
    - Attacking – атака.

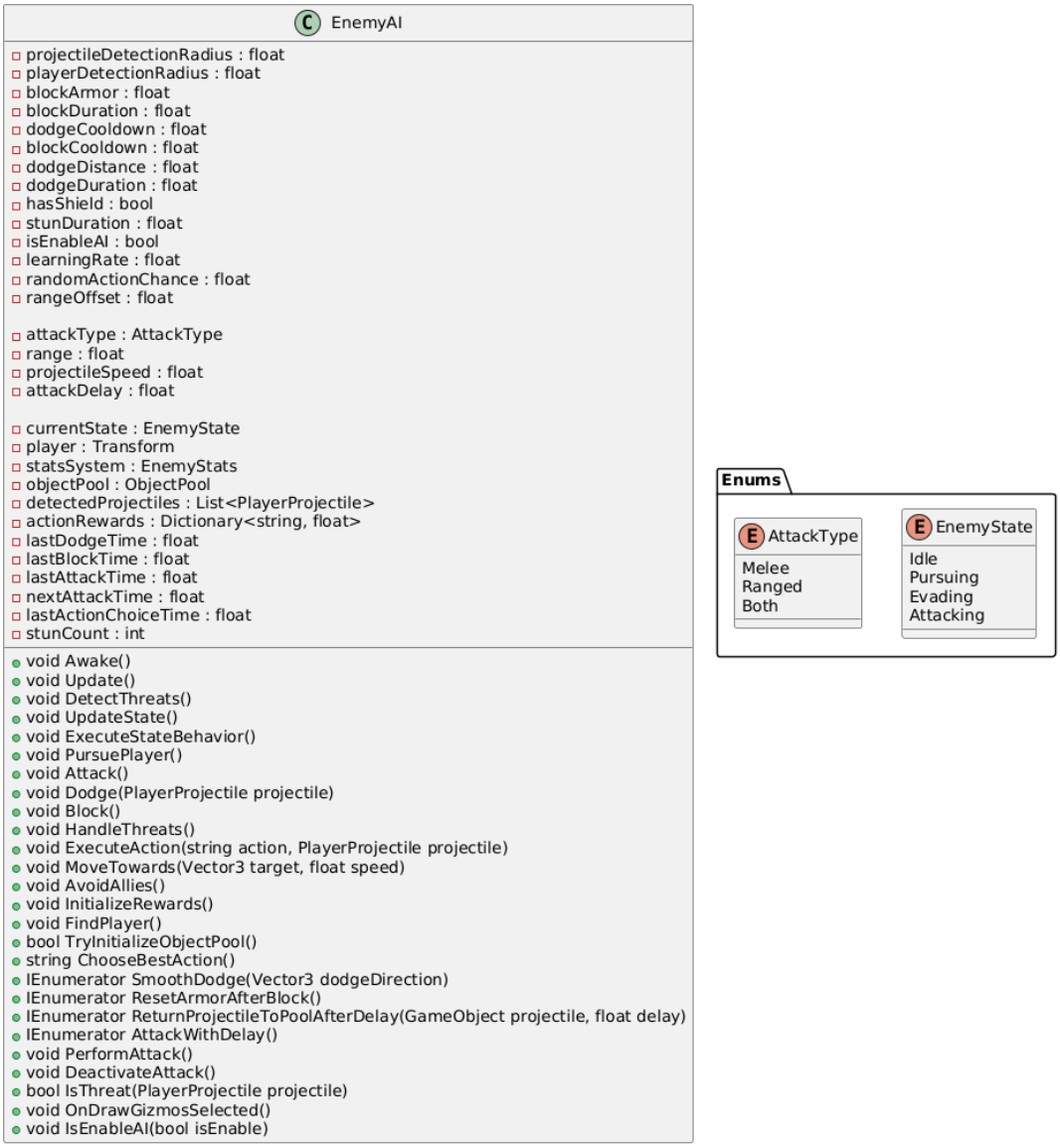


Рисунок 2.28 – Клас EnemyAI

На діаграмах класів котрі були ілюстровані відображають взаємодію між основними компонентами відеогри, які є важливими для її функціонування. Кожен клас виконує задану йому роль, реалізовуючи ігрові механіки та функції.

Перевірка взаємозв'язків та організації класів демонструє, що проєкт містить усі необхідні компоненти для повноцінної гри. Злагоджена взаємодія між класами забезпечує цілісне та ефективне функціонування системи. Така чітка архітектурна структура суттєво спрощує майбутнє розширення функціоналу та технічну підтримку, роблячи процес оновлення гри більш зручним та керованим.

Таким чином, розроблені діаграми класів підтверджують, що спроєктована відеогра включає всі критичні архітектурні елементи, необхідні для розробки стабільного та якісного продукту, який задовольняє сучасні стандарти ігрової індустрії.

РОЗДІЛ 3. ТЕСТУВАННЯ ВІДЕОГРИ

3.1 Тестування основних функціональних вимог відеогри

Тестування є одним із важливих етапів розробки, адже це гарантує надійність, стабільність та виконання функціональних вимог.

У відеоіграх тестування займає особливе значення, оскільки гра є інтерактивною системою із безліччю складових – графіка, механіки, візуальними та аудіо ефектами. Все це повинно працювати як один великий організм, злагоджено та без проблем, аби надати гравцю зануритись в ігровий світ та не вибиватись із нього.

Так як гра працює лише на системах Windows і є одно-користувацькою, тестування буде відбуватись на персональному комп'ютері. Насамперед потрібно запустити ігровий режим в Unity, або збудувати ігровий проєкт, далі перевірити чи весь функціонал працює відповідно.

Для початку необхідно перевірити систему діалогових вікон гри.

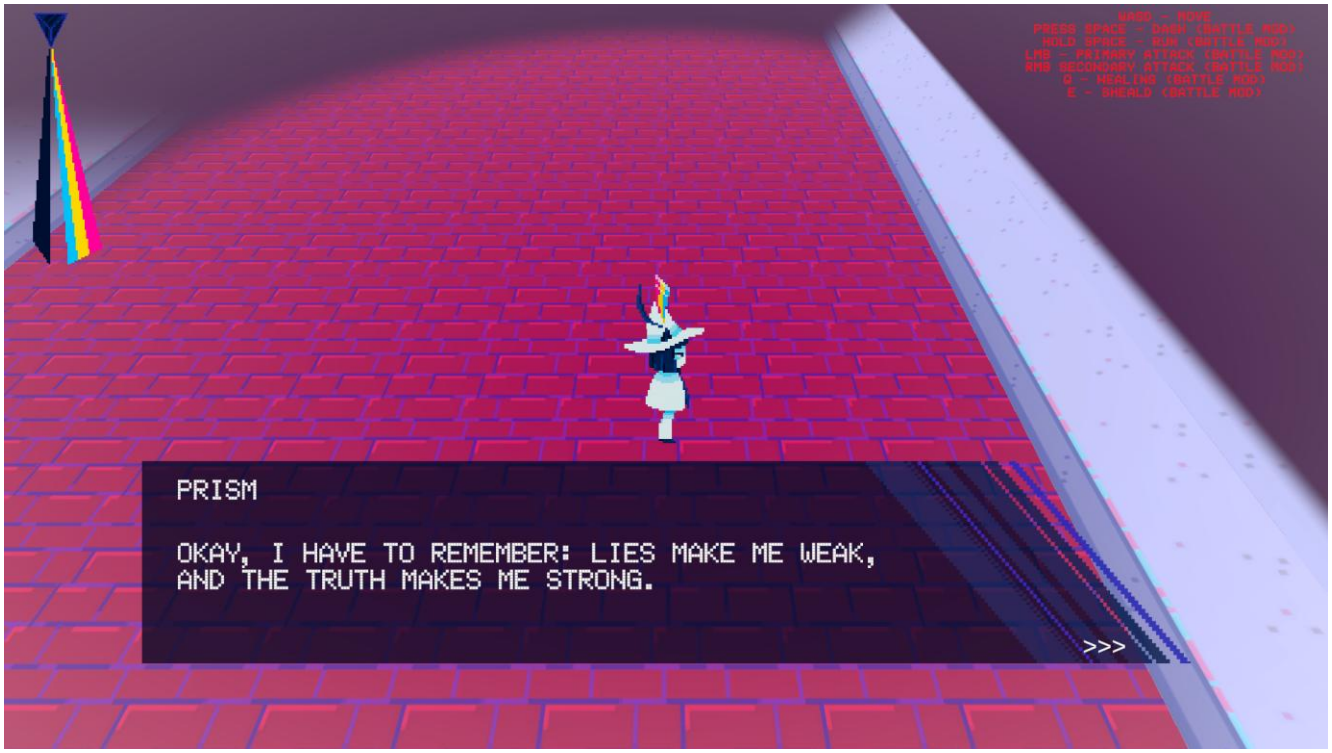


Рисунок 3.1 – Тестування системи діалогових вікон гри

Як було показано на рисунку 3.1, на початку гри вивело діалогове вікно з можливістю перейти до наступного повідомлення в цьому ж вікні за допомогою кнопки на користувацькому інтерфейсі, чи клавішою “space”.

Далі відбудеться перевірка як працює взаємодія із НІПами та підбором предметів із землі.

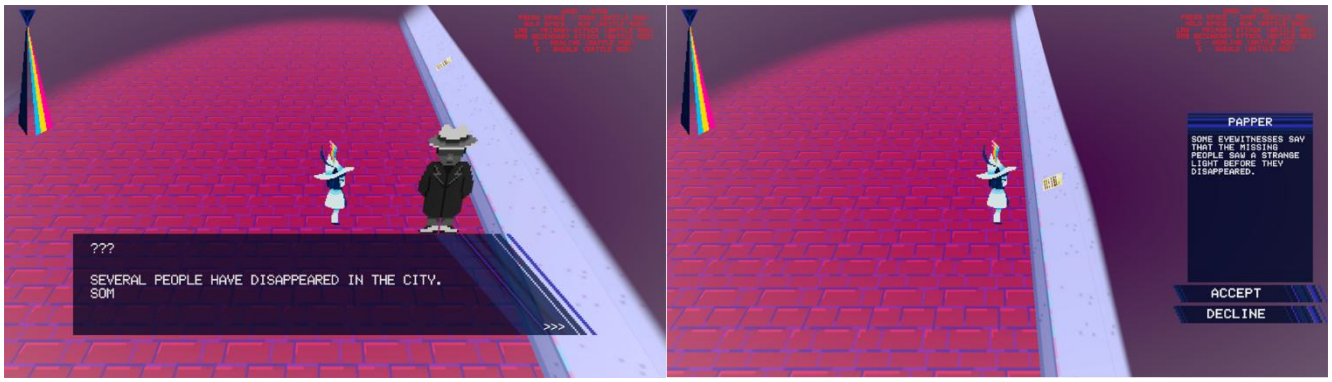


Рисунок 3.2 – Тестування взаємодії із НІПами та предметами

На рисунку 3.2 можна побачити, як при наближені до НІПа виводиться діалогове вікно, на якому показується його ім’я та текст що він нам говорить, а при наближені до предмету виводиться вікно з можливістю відмовитись від предмету, прийняти його, опис предмету та його ім’я. Тож розглянемо детальніше як працює система предметів.

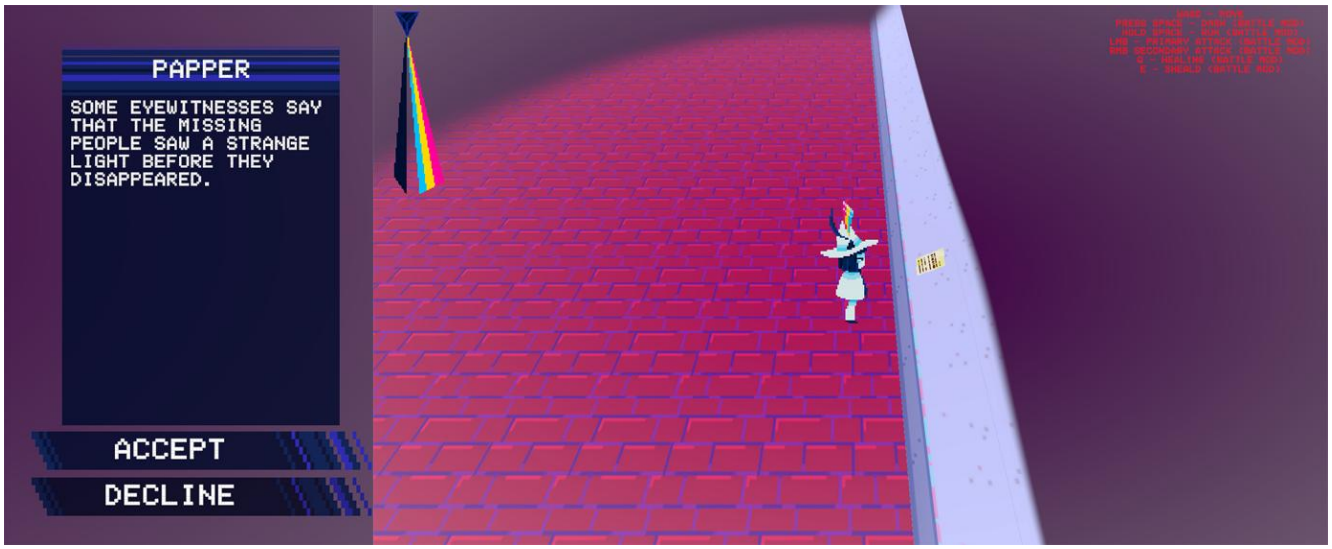


Рисунок 3.3 – Тестування натискання кнопки “Decline”

При натисненні кнопки “Decline” предмет залишиться висіти на стіні, чи лежати на землі, а діалогове вікно зникне. Якщо повторно підійти до предмету діалогове вікно знову з’явиться.

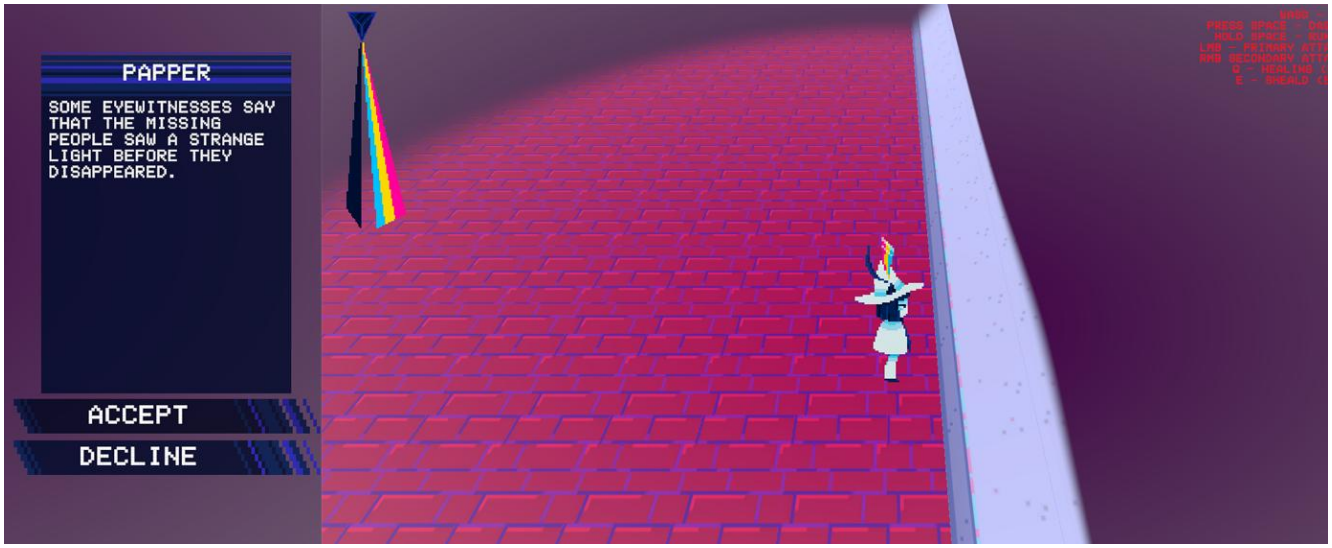


Рисунок 3.4 – Тестування натискання кнопки “Accept”

При натисканні кнопки “Accept”, предмет зникає із стіни чи землі, а його ефект додається в список ефектів, звичайно якщо предмет його мав.

То ж на рисунках 3.3 та 3.4 можна побачити як працює система діалогових вікон гри.

Рисунок 3.5 – Тестування логіки ворогів

На рисунку 3.5 видно як деякі вороги пробують оточити гравця, а деякі атакують на відстані.

Наступне що потрібно буде протестувати, це систему енергії, та регенерації здоров’я, чи барєру.

Рисунок 3.6 – Тестування системи енергії

На рисунку 3.6 видно дві сині сфери довкола персонажа, що означає два повноцінних заряди енергії котрі можна тратити на: постріл енергією, лікування здоров’я, створення бар’єру. Зліва в верхньому куті видно дві смуги: синя та кольорова. Синя смуга відповідає за накопичення заряду енергії, сама енергія накопичується з кожною атакою по ворогу. Кольорова смужка – здоров’я персонажа.

Рисунок 3.7 – Тестування створення бар’єру

При натисканні англійської клавіші “E”, української “У” – сфери енергії збільшуються в розмірі та пришвидшують своє обертання, як це зображено на рисунку 3.7. Використання зарядів енергії для бар’єрів витрачає їх всі.

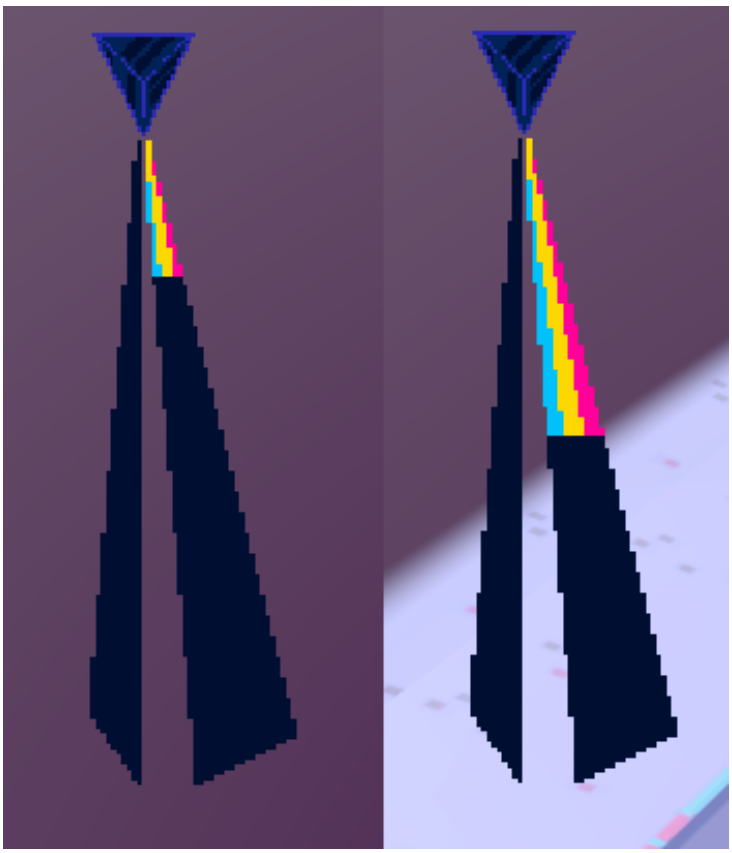


Рисунок 3.8 – Тестування відновлення здоров’я

Подивившись на рисунок 3.8 можна побачити як працює відновлення здоров’я за заряди енергії. Для відновлення такої кількості здоров’я було потрачено два заряди.

3.2 Розвиток відеогри та висновки

Провівши тестування відеогри, вдалося отримати повне уявлення про її функціональність, та роботу. Перевірка основних компонентів, таких як: система діалогових вікон, система взаємодії із предметами, система енергії, лікування та бар’єри за допомогою енергії – показала, що гра відповідає своїм основним вимогам, визначеним підчас проєктування.

Проте, окрім наявності стабільного функціонування, відеогра потребує розвитку в подальшому часі. В майбутньому, для подальшого розвитку проєкту, як

власний незалежний продукт, потрібно буде виправити певну кількість багів та проблем, а саме такі:

- При загибелі гравця, його повинно повертати на останню точку збереження.
- При загибелі гравця вороги повинні перестати атакувати його.
- При відкритті меню паузи, гравець не має мати здатності застосувати: бар’єр/відновлення здоров’я/атаку.

Також грі не вистачає цільного сюжету та покращень механік, тому в подальшому розвитку відеогри, як окремого продукту, має бути покращено та додано наступне:

- Покращення системи діалогів та діалогових вікон.
- Покращення системи предметів.
- Додавання системи навичок.
- Покращення системи предметів.
- Покращення логіки ворогів.
- Створення повноцінних великих рівнів, в яких буде відбуватись розвиток, як головного персонажа так і другорядних героїв.
- Покращення візуального стилю.
- Покращення інтерфейсу.
- Покращення логіки ворогів.
- Додавання більшої різноманітності ворогів.
- Додавання системи збереження.
- Додавання вибору мов.

Беручи підсумки можна сказати наступне – гра все ще сирий продукт, який планується підтримуватись та розроблятись в майбутньому, що в кінцевому результаті має привести до захоплення великої кількості аудиторії. І покращення та доповнення що описані вище мають допомогти із кінцевою метою – виходу гри “Pagefinder” на світовий ринок.

РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ

4.1 Допомога при теплових і сонячних ударах

Теплові та сонячні удари є серйозною небезпекою для працівників, зайнятих на відкритому повітрі або в умовах високих температур. Найбільший ризик мають пожежники, працівники металургійних підприємств, будівельники, шахтарі, аграрії, працівники транспорту та ті, хто виконує фізичну роботу в умовах спеки або прямого сонячного випромінювання.

Тепловий удар виникає внаслідок перегрівання організму під дією зовнішніх теплових факторів.

Характерні ознаки. Головний біль, запаморочення, нудота, блювота, шум у вухах, спрага, шкірні покриви бліді, холодні, пульс та дихання прискорені. Буває носова кровотеча. У важких випадках підвищується температура тіла, втрата пам'яті, поява судоми [18].

Механізм надання домедичної допомоги при теплових і сонячних ударах

Даний алгоритм встановлює процедуру надання невідкладної допомоги особами без медичної освіти, які за своїми професійними обов'язками зобов'язані надавати домедичну допомогу постраждалим від теплових ударів.

Визначення теплового удару

Тепловий удар являє собою критичний стан організму, спричинений впливом підвищеної температури оточуючого середовища, що призводить до комплексних порушень функціонування систем організму.

Клінічні прояви теплового удару: підвищення температури тіла до критичних показників (може сягати 41°C), зміна кольору та стану шкірних покривів (червоний колір, підвищена температура, сухість), порушення поведінки та свідомості (агресивність, дезорієнтація, непритомність), зміни дихальних функцій (часте, неглибоке дихання).

Алгоритм надання домедичної допомоги при тепловому ударі:

1. Оцінка безпеки ситуації для всіх учасників події перед початком надання допомоги.
2. Психологічна підтримка постраждалого та інформування про подальші дії.
3. Негайний контакт зі службою екстреної медичної допомоги з дотриманням інструкцій оператора.
4. Усунення теплового впливу шляхом переміщення постраждалого до прохолодного місця.
5. Контроль температури тіла за допомогою відповідного обладнання (за наявності кваліфікації та приладів).
6. Застосування методів зниження температури тіла: при температурі понад 40°C - повне занурення у прохолодну воду (18-26°C) до зниження температури нижче 39°C. Альтернативне охолодження: використання холодових пакетів, вентиляторів із зволоженими серветками, забезпечення безпеки дихальних шляхів під час процедур.
7. Безперервний контроль стану постраждалого до прибуття медичної бригади.
8. Забезпечення рідиною при збереженні свідомості.
9. Повторний виклик екстреної допомоги при погіршенні стану.
10. Збір анамнестичних даних для передачі медичним працівникам.

Особливості сонячного удару

Сонячний удар розвивається внаслідок безпосереднього впливу сонячних променів на область голови.

Симптоматика сонячного удару: цефалгія, вестибулярні розлади, диспептичні явища, гіперемія шкіри, гіпергідроз, можлива носова кровотеча, тахікардія, тахіпное, гіпертермія, у складних випадках - втрата свідомості, судомний синдром.

Невідкладна допомога при сонячному ударі: переміщення постраждалого в затінене місце, застосування холодового компресу на область голови, у важких випадках - обгортання зволоженою тканиною.

Критичні ситуації

У разі втрати свідомості постраждалим до прибуття медичної бригади необхідно керуватися протоколами надання домедичної допомоги при зупинці кровообігу відповідно до віку постраждалого, затвердженими Міністерством охорони здоров'я України [19].

4.2 Заходи щодо автоматизації виробничих процесів, які сприяють покращенню умов праці.

У сучасному виробництві автоматизація виконує ключову роль не лише як інструмент підвищення продуктивності, а насамперед як засіб забезпечення безпечних умов праці. Її застосування особливо важливе в галузях, де працівники щодня піддаються ризику через екстремальні або небезпечні виробничі умови: у гірничій справі, металургії, хімічній промисловості, пожежно-рятувальних службах, атомній енергетиці, сільському господарстві, будівництві та інших галузях.

Основні функції автоматизації:

1. Заміна важкої фізичної праці — особливо актуально на ділянках, де виконання завдань пов'язане з постійним фізичним навантаженням або одноманітними, повторюваними операціями. Роботизовані комплекси, конвеєри, маніпулятори дають змогу усунути людську участь з найвиснажливіших процесів.
2. Виконання робіт у непридатних або небезпечних для людини умовах – автоматизовані системи дозволяють керувати процесами дистанційно або повністю без участі людини в середовищах із високим тиском, температурою, токсичними чи вибухонебезпечними речовинами, в умовах ризику обвалів чи пожеж.

Переваги автоматизації з точки зору охорони праці:

1. Мінімізація ризику для життя і здоров’я працівників: дистанційне керування гірничими комбайнами у вугільних шахтах із високим ризиком обвалів або метанового вибуху, автоматизовані системи нейтралізації хімічних викидів на підприємствах хімічної галузі.
2. Зниження фізичного та психоемоційного навантаження – Автоматизовані системи зменшують втому працівників, дозволяючи їм зосереджуватись на управлінні та контролі процесів, а не на важких або монотонних діях. Наприклад, у логістиці – сортувальні комплекси, що замінюють ручну працю.
3. Зменшення впливу шкідливих виробничих факторів – шум, вібрація, пил, отруйні речовини, перевантаження, екстремальні температури, автоматизоване обладнання дозволяє значно зменшити контакт працівника із цими загрозами, або взагалі його уникнути.
4. Підвищення точності і контрольованості технологічних процесів – Автоматика дозволяє виключити людський фактор у критичних виробничих операціях.
5. Скорочення кількості аварій та виробничого травматизму – У зонах підвищеної небезпеки (робота на висоті, в замкнених просторах, біля рухомих механізмів) використання дистанційних систем і роботів істотно знижує ймовірність нещасних випадків.

Приклади застосування автоматизації у небезпечних виробництвах:

1. Шахтарська промисловість: дистанційно керовані прохідницькі машини, автоматичні вентиляційні системи, системи моніторингу рівня метану.
2. Металургія: автоматизовані ливарні установки, роботи для завантаження/вивантаження гарячих заготовок, тепловізійні системи контролю.
3. Хімічна галузь: автоматичні системи подачі та змішування реагентів, датчики витоку, аварійні зупинки.

4. Пожежно-рятувальні служби: роботи-розвідники для роботи в зоні загоряння, повітряні роботи для повітряного спостереження лісових пожеж.
5. Сільське господарство: автономні машини для обприскування та збирання врожаю, системи автоматичного поливу, контроль клімату в теплицях.
6. Будівництво: автономні крани, системи контролю стійкості риштувань.

Відповідно до Закону України «Про охорону праці» [20], роботодавець зобов’язаний вживати технічних заходів щодо усунення небезпечних виробничих факторів. Одним з таких є впровадження автоматизованих систем управління технологічними процесами.

Автоматизація виробничих процесів – це сучасний інструмент забезпечення промислової безпеки, що забезпечує організацію робочих середовищ таким чином, щоб виключити або звести до мінімуму участь людини в небезпечних процесах. Це сприяє збереженню здоров’я працівників у найризикованіших професіях.

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи, мною було досліджено та розроблено 2.5D гру з використанням SRP фреймворку та рушія Unity.

Результатом є відеогра, що відповідає базовим потребам та вміщує в собі Souls-Like та Adventure жанри. Використовує URP для забезпечення продуктивності та збереження якості зображення за допомогою шейдерів.

Вибір Unity відбувся завдяки його перевагам: зручність інтерфейсу та можливість його налаштувати під себе, майже безмежна база матеріалів для навчання, та його продуктивна й товариська спільнота, легкість освоєння.

Було створено гру: із декількома сутичками із НІПом, три коридори, три типи предметів та різні їх вплив на середовище (на гравця та ворогів), бойова арена із декількома ворогами та три варіанта розвитку подій. Було додано туман війни для обмеження поля зору гравцеві, до того ж це набагато продуктивніше, аніж використовувати об’єкти. Також було створено шейдер-сітку, для заповнення об’єкту землі – це допомогло зменшити кількість використовуваних об’єктів землі, та можливість використовувати більш динамічні текстури для об’єкту землі за допомогою маски. Маска була поділена на три кольори, для трьох текстур: червоний, зелений, синій.

Проведена робота відображає розробку відеоігор за допомогою сучасних інструментів розробки. Використання Unity та SRP фреймворку надало змогу створити гру, що не лише виглядає незвично, але й з непомітним використанням трьох вимірних об’єктів. Данна розробка доводить важливість поєднання творчих та технічних навичок разом із сучасними приладдями розробки для досягнення результатів в галузі геймдеву.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Pac-Man. URL: https://en.wikipedia.org/wiki/Pac-Man.
2. Unity. URL: https://unity.com/.
3. Godot. URL: https://godotengine.org/.
4. Unreal Engine. URL: https://www.unrealengine.com/en-US.
5. Source 2. URL: https://developer.valvesoftware.com/wiki/Source_2.
6. S&box. URL: https://store.steampowered.com/app/590830/sbox.
7. What is a Souls-like? Game devs break down FromSoftware's accidental genre. URL: https://www.gamesradar.com/what-is-a-souls-like-developers-explain/.
8. Know Your Genres: Adventure Games. URL: https://news.xbox.com/en-us/2015/07/17/games-know-your-genres-adventure/.
9. Demon`s Souls. URL: https://store.playstation.com/uk-ua/concept/10000368.
10. Dark Souls. URL: https://store.steampowered.com/app/570940/DARK_SOULS_REMASTERED.
11. The Last of Us. URL: https://store.steampowered.com/app/1888930/The_Last_of_Us_Part_I/.
12. What is a 2.5D Game? Definition, Features, and Examples. URL: https://medium.com/@expertappdevs/what-is-a-2-5d-game-definition-features-and-examples-6cca2546493b .
13. Star Ocean. URL: https://www.rockpapershotgun.com/star-ocean-the-second-story-r-will-bring-the-90s-playstation-rpg-to-pc-with-a-hot-new-25d-look.
14. Review on GPU Architecture. URL: https://web.archive.org/web/20240906010254/https://www.irjet.net/archives/V11/i7/IRJET-V11I7124.pdf.
15. Global Game Jame. URL: https://globalgamejam.org/.
16. Pixel-Perfect Fog Of War. URL: https://assetstore.unity.com/packages/vfx/shaders/fullscreen-camera-effects/pixel-

perfect-fog-of-war-229484?srsltid=AfmBOoq77bb2cpcbzQZ4L4BvrKLFehnjY7kFEYM_23PYz3F4shftx7U7.

17. Unity hub. URL: https://unity.com/unity-hub.

18. ТЕМА 9. Медицина катастроф. Долікарська допомога при теплових і сонячних ударах. URL: https://dl.tntu.edu.ua/content.php?cid=299872.

19. НАКАЗ Про затвердження порядків надання домедичної допомоги особам при невідкладних станах 28 березня 2022 р. за № 356/37692 URL: https://zakon.rada.gov.ua/laws/show/z0356-22#n769.

20. ЗАКОН УКРАЇНИ Про охорону праці. URL: https://zakon.rada.gov.ua/laws/main/2694-12.

ДОДАТКИ

Додаток А – Тези конференції

VIII Міжнародна студентська науково - технічна конференція
"ПРИРОДНИЧІ ТА ГУМАНІТАРНІ НАУКИ. АКТУАЛЬНІ ПИТАННЯ"

УДК 004.4
Тітков В. –ст. гр. СП-43
Тернопільський національний технічний університет імені Івана Пулюя

РОЗРОБКА 3D ГРИ З ВИКОРИСТАННЯМ UNITY 3 ТА SRP ФРЕЙМВОРКУ

Науковий керівник: к.т.н., доц. Стоянов Ю.М.

Titkov V.
Ternopil Ivan Puluj National Technical University

3D GAME DEVELOPMENT USING UNITY 3 AND SRP FRAMEWORK

Supervisor: PhD Yurii Stoianov

Ключові слова: інженерія програмного забезпечення, Unity, Unity 3, SRP, 3D, гра
Key words: software engineering, Unity, Unity 3, SRP, 3D, game

Створення ігрових 3D світів, гнучка інтеграція графічних компонентів і адаптивна побудова сцени — все це стає можливим завдяки сучасним технологіям Unity 3, які забезпечують фундамент для розробки ігор нового покоління. Упевнено нарощує потужності SRP фреймворк, який дозволяє програмно контролювати рендеринг, оптимізувати продуктивність і виводити візуальні ефекти на рівень AAA-продуктів. Стабільно підтримується розвиток індивідуальних рендер-пайплайнів, кастомізація URP та HDRP, впровадження світлових технологій для мобільних платформ і фотореалізму одночасно. Ми впевнені, що впровадження SRP (Scriptable Render Pipeline) — життєво необхідне для конкурентної розробки.

Накопичення практик оптимізації, SRP Batcher і модульних шейдерів забезпечило значне зменшення навантаження на CPU та GPU, і як наслідок — збільшення швидкості рендерингу, зокрема в проектах з великою кількістю однакових об'єктів. Ми прагнемо інтегрувати рендер-сценарії в синергетичному поєднанні з ігровою логікою, а також забезпечити розробників точними інструментами шейдингу, освітлення, камери й буферів глибини. Завдяки кваліфікованим кадрам, відкритій архітектурі SRP та офіційній документації Unity [1], індустрія отримує зручний інтерфейс для створення кастомних пайплайнів, які відображають візуальні вимоги конкретної гри або студії.

Упевнене масштабування, технічна вправність, SRP Debugger і профайлер забезпечили гнучке управління ресурсами та провідну роль Unity серед рушіїв для кросплатформеної розробки. Одна з найбільших технологічних платформ надає повний стек інструментів, включаючи Universal Render Pipeline для середнього класу пристроїв і High Definition Render Pipeline — для найвибагливіших проєктів. Наша мета проста: забезпечення якісного візуального середовища, адаптованого під вимоги платформи, з акцентом на продуктивність, візуальну естетику та реалістичність. Ми використовуємо потужність Unity Shader Graph, Forward та Deferred рендерингу, а також фреймворк URP як універсальний інструмент розробника для реалізації ігрових світів, процедурних ефектів та адаптивного освітлення [2].

Література:
1. Unity Documentation. URL: https://docs.unity3d.com/2022.3/Documentation/Manual/index.html.
2. SRP: Scriptable Render Pipeline. URL: https://unity.com/features/srp.

Додаток Б – Лістинг коду Grid шейдера

Лістинг коду Grid.shader

```
Shader "Exavi/GridURP"
{
    Properties
    {
        _Color ("Color", Color) = (1,1,1,1)
        [Space(10)]
        [Header(Texture Settings)]
        [Space(5)]

        _MainTex ("Texture 1", 2D) = "white" {}
        _SecondTex ("Texture 2", 2D) = "white" {}
        _ThirdTex ("Texture 3", 2D) = "white" {}
        _MaskTex ("Mask Texture", 2D) = "white" {}
        [Space(10)]
        [Header(Rotation Settings)]
        _EnableRandomRotation ("Enable Random Rotation", Float) = 0
        [Space(10)]
        [Header(Grid Settings)]
        _TileSize ("Tile Size", Float) = 1.0
        _GridScale ("Grid Scale", Range(0.1, 10)) = 1.0
        _Seed ("Random Seed", Float) = 1.0
        }

        SubShader
        {
        Tags {
            "RenderType" = "Opaque"
            "RenderPipeline" = "UniversalPipeline"
            "Queue" = "Geometry"
        }

        Pass
        {
            Name "ForwardLit"
            Tags { "LightMode" = "UniversalForward" }

            HLSLPROGRAM
            #pragma vertex vert
            #pragma fragment frag
            #pragma multi_compile _ _MAIN_LIGHT_SHADOWS
            #pragma multi_compile _ _MAIN_LIGHT_SHADOWS_CASCADE
                  #pragma  multi_compile  _  _ADDITIONAL_LIGHTS_VERTEX
_ADDITIONAL_LIGHTS
            #pragma multi_compile _ _SHADOWS_SOFT
```

```
                                    #include    "Packages/com.unity.render-
pipelines.universal/ShaderLibrary/Core.hlsl"
                                    #include    "Packages/com.unity.render-
pipelines.universal/ShaderLibrary/Lighting.hlsl"

                struct Attributes
                {
                float4 positionOS : POSITION;
                float2 uv : TEXCOORD0;
                float3 normalOS : NORMAL;
                };

                struct Varyings
                {
                float4 positionCS : SV_POSITION;
                float2 uv : TEXCOORD0;
                float3 positionWS : TEXCOORD1;
                float3 normalWS : TEXCOORD2;
                };

                TEXTURE2D(_MainTex);
                TEXTURE2D(_SecondTex);
                TEXTURE2D(_ThirdTex);
                TEXTURE2D(_MaskTex);
                SAMPLER(sampler_MainTex);

                CBUFFER_START(UnityPerMaterial)
                float4 _Color;
                float _TileSize;
                float _GridScale;
                float _EnableRandomRotation;
                float _Seed;
                CBUFFER_END

                float hash(float2 p)
                {
               float3 p3 = frac(float3(p.xyx) * float3(.1031, .1030, .0973));
                p3 += dot(p3, p3.yzx + 33.33);
                return frac((p3.x + p3.y) * p3.z);
                }

                float2 RotateUV(float2 uv, float rotation)
                {
                float s = sin(rotation);
                float c = cos(rotation);
                float2x2 rotationMatrix = float2x2(c, -s, s, c);
                uv -= 0.5;
                uv = mul(rotationMatrix, uv);
                uv += 0.5;
                return uv;
```

```
                }

                Varyings vert(Attributes IN)
                {
                Varyings OUT;
                OUT.positionCS = TransformObjectToHClip(IN.positionOS.xyz);
                OUT.positionWS = TransformObjectToWorld(IN.positionOS.xyz);
                OUT.normalWS = TransformObjectToWorldNormal(IN.normalOS);
                OUT.uv = IN.uv;
                return OUT;
                }

                half4 frag(Varyings IN) : SV_Target
                {
                float2 tilePosition = floor(IN.positionWS.xz * _GridScale /
_TileSize);
                    float2 tileUV = frac(IN.positionWS.xz * _GridScale /
_TileSize);

                float randomValue = hash(tilePosition + _Seed);

                if (_EnableRandomRotation > 0.5)
                {
                    float rotation = randomValue * 6.28318530718;
                    tileUV = RotateUV(tileUV, rotation);
                }

                  half4 mask = SAMPLE_TEXTURE2D(_MaskTex, sampler_MainTex,
IN.uv);
                half4 color;

                if (mask.r > 0.5)
                       color = SAMPLE_TEXTURE2D(_MainTex, sampler_MainTex,
tileUV);
                else if (mask.g > 0.5)
                     color = SAMPLE_TEXTURE2D(_SecondTex, sampler_MainTex,
tileUV);
                else if (mask.b > 0.5)
                      color = SAMPLE_TEXTURE2D(_ThirdTex, sampler_MainTex,
tileUV);
                else
                    color = half4(0, 0, 0, 1);

                                                Light      mainLight      =
GetMainLight(TransformWorldToShadowCoord(IN.positionWS));
                float3 normalWS = normalize(IN.normalWS);
                float NdotL = saturate(dot(normalWS, mainLight.direction));
                float3 ambient = SampleSH(normalWS);

                  color.rgb *= _Color.rgb * (mainLight.shadowAttenuation *
NdotL * mainLight.color + ambient);
```

```
                return color;
                }
                ENDHLSL
            }

            Pass
            {
                Name "ShadowCaster"
                Tags
                {
                    "LightMode" = "ShadowCaster"
                }

                ZWrite On
                ZTest LEqual
                ColorMask 0
                Cull[_Cull]

                HLSLPROGRAM
                #pragma target 2.0

                #pragma vertex ShadowPassVertex
                #pragma fragment ShadowPassFragment

                #pragma shader_feature_local _ALPHATEST_ON
                                    #pragma    shader_feature_local_fragment
_SMOOTHNESS_TEXTURE_ALBEDO_CHANNEL_A

                #pragma multi_compile_instancing
                        #include_with_pragmas   "Packages/com.unity.render-
pipelines.universal/ShaderLibrary/DOTS.hlsl"

                #pragma multi_compile _ LOD_FADE_CROSSFADE

               #pragma multi_compile_vertex _ _CASTING_PUNCTUAL_LIGHT_SHADOW

                                    #include    "Packages/com.unity.render-
pipelines.universal/Shaders/LitInput.hlsl"
                                    #include    "Packages/com.unity.render-
pipelines.universal/Shaders/ShadowCasterPass.hlsl"
                ENDHLSL
            }

            Pass
            {
                Name "GBuffer"
                Tags
                {
                    "LightMode" = "UniversalGBuffer"
                }
```

```
                ZWrite[_ZWrite]
                ZTest LEqual
                Cull[_Cull]

                HLSLPROGRAM
                #pragma target 4.5

                #pragma exclude_renderers gles3 glcore

                #pragma vertex LitGBufferPassVertex
                #pragma fragment LitGBufferPassFragment

                #pragma shader_feature_local _NORMALMAP
                #pragma shader_feature_local_fragment _ALPHATEST_ON
                #pragma shader_feature_local_fragment _EMISSION
                #pragma shader_feature_local_fragment _METALLICSPECGLOSSMAP
                                        #pragma    shader_feature_local_fragment
_SMOOTHNESS_TEXTURE_ALBEDO_CHANNEL_A
                #pragma shader_feature_local_fragment _OCCLUSIONMAP
                #pragma shader_feature_local _PARALLAXMAP
                #pragma shader_feature_local _ _DETAIL_MULX2 _DETAIL_SCALED

                                        #pragma    shader_feature_local_fragment
_SPECULARHIGHLIGHTS_OFF
                                        #pragma    shader_feature_local_fragment
_ENVIRONMENTREFLECTIONS_OFF
                #pragma shader_feature_local_fragment _SPECULAR_SETUP
                #pragma shader_feature_local _RECEIVE_SHADOWS_OFF

                                  #pragma   multi_compile   _   _MAIN_LIGHT_SHADOWS
_MAIN_LIGHT_SHADOWS_CASCADE _MAIN_LIGHT_SHADOWS_SCREEN
                #pragma multi_compile_fragment _ _REFLECTION_PROBE_BLENDING
                                         #pragma    multi_compile_fragment    _
_REFLECTION_PROBE_BOX_PROJECTION
                                #pragma   multi_compile_fragment   _   _SHADOWS_SOFT
_SHADOWS_SOFT_LOW _SHADOWS_SOFT_MEDIUM _SHADOWS_SOFT_HIGH
               #pragma multi_compile_fragment _ _DBUFFER_MRT1 _DBUFFER_MRT2
_DBUFFER_MRT3
                #pragma multi_compile_fragment _ _RENDER_PASS_ENABLED
                        #include_with_pragmas  "Packages/com.unity.render-
pipelines.universal/ShaderLibrary/RenderingLayers.hlsl"

                #pragma multi_compile _ LIGHTMAP_SHADOW_MIXING
                #pragma multi_compile _ SHADOWS_SHADOWMASK
                #pragma multi_compile _ DIRLIGHTMAP_COMBINED
                #pragma multi_compile _ LIGHTMAP_ON
                #pragma multi_compile _ DYNAMICLIGHTMAP_ON
                #pragma multi_compile _ USE_LEGACY_LIGHTMAPS
                #pragma multi_compile _ LOD_FADE_CROSSFADE
                #pragma multi_compile_fragment _ _GBUFFER_NORMALS_OCT
```

```
                        #include_with_pragmas  "Packages/com.unity.render-
pipelines.universal/ShaderLibrary/ProbeVolumeVariants.hlsl"

                #pragma multi_compile_instancing
                #pragma instancing_options renderinglayer
                        #include_with_pragmas  "Packages/com.unity.render-
pipelines.universal/ShaderLibrary/DOTS.hlsl"

                                    #include    "Packages/com.unity.render-
pipelines.universal/Shaders/LitInput.hlsl"
                                    #include    "Packages/com.unity.render-
pipelines.universal/Shaders/LitGBufferPass.hlsl"
                ENDHLSL
            }

            Pass
            {
                Name "DepthOnly"
                Tags
                {
                    "LightMode" = "DepthOnly"
                }

                ZWrite On
                ColorMask R
                Cull[_Cull]

                HLSLPROGRAM
                #pragma target 2.0

                #pragma vertex DepthOnlyVertex
                #pragma fragment DepthOnlyFragment

                #pragma shader_feature_local _ALPHATEST_ON
                                    #pragma    shader_feature_local_fragment
_SMOOTHNESS_TEXTURE_ALBEDO_CHANNEL_A

                #pragma multi_compile _ LOD_FADE_CROSSFADE

                #pragma multi_compile_instancing
                        #include_with_pragmas  "Packages/com.unity.render-
pipelines.universal/ShaderLibrary/DOTS.hlsl"

                                    #include    "Packages/com.unity.render-
pipelines.universal/Shaders/LitInput.hlsl"
                                    #include    "Packages/com.unity.render-
pipelines.universal/Shaders/DepthOnlyPass.hlsl"
                ENDHLSL
            }

            Pass
```

```
            {
                Name "DepthNormals"
                Tags
                {
                    "LightMode" = "DepthNormals"
                }

                ZWrite On
                Cull[_Cull]

                HLSLPROGRAM
                #pragma target 2.0

                #pragma vertex DepthNormalsVertex
                #pragma fragment DepthNormalsFragment

                #pragma shader_feature_local _NORMALMAP
                #pragma shader_feature_local _PARALLAXMAP
                #pragma shader_feature_local _ _DETAIL_MULX2 _DETAIL_SCALED
                #pragma shader_feature_local _ALPHATEST_ON
                                    #pragma    shader_feature_local_fragment
_SMOOTHNESS_TEXTURE_ALBEDO_CHANNEL_A

                #pragma multi_compile _ LOD_FADE_CROSSFADE

                            #include_with_pragmas  "Packages/com.unity.render-
pipelines.universal/ShaderLibrary/RenderingLayers.hlsl"

                #pragma multi_compile_instancing
                            #include_with_pragmas  "Packages/com.unity.render-
pipelines.universal/ShaderLibrary/DOTS.hlsl"

                                    #include    "Packages/com.unity.render-
pipelines.universal/Shaders/LitInput.hlsl"
                                    #include    "Packages/com.unity.render-
pipelines.universal/Shaders/LitDepthNormalsPass.hlsl"
                ENDHLSL
            }

            Pass
            {
                Name "Meta"
                Tags
                {
                    "LightMode" = "Meta"
                }

                Cull Off

                HLSLPROGRAM
                #pragma target 2.0
```

```
                #pragma vertex UniversalVertexMeta
                #pragma fragment UniversalFragmentMetaLit

                #pragma shader_feature_local_fragment _SPECULAR_SETUP
                #pragma shader_feature_local_fragment _EMISSION
                #pragma shader_feature_local_fragment _METALLICSPECGLOSSMAP
                #pragma shader_feature_local_fragment _ALPHATEST_ON
                                #pragma    shader_feature_local_fragment    _
_SMOOTHNESS_TEXTURE_ALBEDO_CHANNEL_A
                #pragma shader_feature_local _ _DETAIL_MULX2 _DETAIL_SCALED
                #pragma shader_feature_local_fragment _SPECGLOSSMAP
                #pragma shader_feature EDITOR_VISUALIZATION

                                    #include    "Packages/com.unity.render-
pipelines.universal/Shaders/LitInput.hlsl"
                                    #include    "Packages/com.unity.render-
pipelines.universal/Shaders/LitMetaPass.hlsl"

                ENDHLSL
            }

            Pass
            {
                Name "Universal2D"
                Tags
                {
                    "LightMode" = "Universal2D"
                }

                Blend[_SrcBlend][_DstBlend]
                ZWrite[_ZWrite]
                Cull[_Cull]

                HLSLPROGRAM
                #pragma target 2.0

                #pragma vertex vert
                #pragma fragment frag

                #pragma shader_feature_local_fragment _ALPHATEST_ON
                #pragma shader_feature_local_fragment _ALPHAPREMULTIPLY_ON

                        #include_with_pragmas  "Packages/com.unity.render-
pipelines.universal/ShaderLibrary/DOTS.hlsl"

                                    #include    "Packages/com.unity.render-
pipelines.universal/Shaders/LitInput.hlsl"
                                    #include    "Packages/com.unity.render-
pipelines.universal/Shaders/Utils/Universal2D.hlsl"
                ENDHLSL
```

```
            }

            Pass
            {
                Name "MotionVectors"
                Tags { "LightMode" = "MotionVectors" }
                ColorMask RG

                HLSLPROGRAM
                #pragma shader_feature_local _ALPHATEST_ON
                #pragma multi_compile _ LOD_FADE_CROSSFADE
                                    #pragma    shader_feature_local_vertex
_ADD_PRECOMPUTED_VELOCITY

                                   #include    "Packages/com.unity.render-
pipelines.universal/Shaders/LitInput.hlsl"
                           #include_with_pragmas   "Packages/com.unity.render-
pipelines.universal/ShaderLibrary/ObjectMotionVectors.hlsl"
                ENDHLSL
            }

            Pass
            {
                Name "XRMotionVectors"
                Tags { "LightMode" = "XRMotionVectors" }
                ColorMask RGBA

                Stencil
                {
                    WriteMask 1
                    Ref 1
                    Comp Always
                    Pass Replace
                }

                HLSLPROGRAM
                #pragma shader_feature_local _ALPHATEST_ON
                #pragma multi_compile _ LOD_FADE_CROSSFADE
                                    #pragma    shader_feature_local_vertex
_ADD_PRECOMPUTED_VELOCITY
                #define APLICATION_SPACE_WARP_MOTION 1

                                   #include    "Packages/com.unity.render-
pipelines.universal/Shaders/LitInput.hlsl"
                           #include_with_pragmas   "Packages/com.unity.render-
pipelines.universal/ShaderLibrary/ObjectMotionVectors.hlsl"
                ENDHLSL
            }
        }
    }
```

Додаток В – Грамота GameJam

Додаток Г – Диск з роботою