КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
імені ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій
Кафедра прикладних інформаційних систем

122 «Комп'ютерні науки»
(шифр і назва спеціальності)

«Прикладне програмування»
(назва освітньої програми)

Кваліфікаційна робота бакалавра

на тему: «Мобільний застосунок гри»

Виконав ____________________
(Підпис)
Рибас Ілля Романович
(прізвище, ім'я, по батькові)

Керівник _Ващіліна О.В._
(прізвище, ім'я, по батькові)
__*до захисту*______________________________
(Резолюція «До захисту»)

Унікальність тексту 93,69 %
Автор __________ Рибас І.Р.
(Підпис) (Прізвище, ініціали)

Попередній захист:

__*До захисту в екзаменаційній комісії*_________________
(Висновок: "До захисту в екзаменаційній комісії ")
Завідувач кафедри __________ Плескач В.Л. 23.06.2022
(Підпис) (Прізвище, ініціали) (Дата)

Київ – 2022

Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій

Кафедра прикладних інформаційних систем

Назва теми: «___________Мобільний застосунок гри__________________»

Освітня програма: Прикладне програмування

Спеціальність: Комп’ютерні науки

ПІБ	Підпис
Рибас Ілля Романович	

Назва роботи українською та англійською мовами:

Мобільний застосунок гри
Mobile game application

Мета бакалаврської роботи: Створення кросплатформенного мобільного застосунку гри
План роботи: 1. Підходи до розробки сучасних мобільних ігор 2. Аналіз програмно-технологічних рішень мобільних ігор та технологічних можливостей платформ Android та iOS 3. Реалізація та впровадження мобільної гри мовою C#

ПІБ, ступінь, звання наукового керівника роботи:

Ващіліна О.В., к.ф.-м.н., доцент

КАЛЕНДАРНИЙ ПЛАН ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

№з/п	Назва етапів кваліфікаційної роботи бакалавра	Термін виконання етапів кваліфікаційної роботи бакалавра	Відмітка про виконання
1.	Вибір теми та наукового керівника кваліфікаційної роботи бакалавра	09.10.2021	Виконав
2.	Видача завдання кваліфікаційної роботи бакалавра	19.10.2021	Виконав
3.	Настановча групова співбесіда з питань кваліфікаційної роботи бакалавра	21.10.2021	Виконав
4.	Затвердження плану кваліфікаційної роботи бакалавра	25.10.2022	Виконав
5.	Підбір та вивчення літературних та інших джерел з теми дослідження	01.11.2022	Виконав
6.	Підготовка і подання науковому керівнику першого варіанту І розділу роботи	21.12.2022	Виконав
7.	Підготовка і подання науковому керівнику першого варіанту ІІ розділу роботи	31.01.2022	Виконав
8.	Підготовка і подання науковому керівнику першого варіанту ІІІ розділу роботи	30.03.2022	Виконав
9.	Подання роботи у першому варіанті	28.04.2022	Виконав
10.	Оформлення пояснювальної записки кваліфікаційної роботи бакалавра	03.05.2022	Виконав
11.	Подання кваліфікаційної роботи бакалавра на попередній захист	**23.05.2022**	Виконав
12.	Врахування зауважень керівника і подання роботи в остаточному варіанті (з відповідним висновком про допуск) на кафедру	27.05.2022	Виконав
13.	Затвердження роботи в цілому (підготовка письмового відгуку керівника, письмова рецензія на бакалаврської роботу)	10.06.2022	Виконав
14.	Захист кваліфікаційної роботи бакалавра	22.06.2022 23.06.2022 24.06.2022	

Здобувач вищої освіти __ ______ Керівник ____________
(підпис) (підпис)

ВІДОМІСТЬ КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

Складові частини дипломної роботи	Обсяг, арк.
Титульний аркуш	1
Календарний план дипломної роботи	1
Відомість дипломної роботи	1
Анотація	1
Анотація (іноземною мовою-англійською)	1
Зміст	1
Перелік скорочень, умовних позначень, термінів	1
Вступ	2
Розділ 1	6
Розділ 2	8
Розділ 3	34
Висновки	1
Перелік використаних джерел	3
Додатки	98

<table>
<tr><td></td><td></td><td></td><td></td><td colspan="3" rowspan="2">ДП XXXX 00.000.00</td></tr>
<tr><td></td><td>ПІБ</td><td>Підп.</td><td>Дата</td></tr>
<tr><td>Ризробн.</td><td>Рибас І.Р.</td><td></td><td></td><td rowspan="4">Відомість дипломної роботи</td><td>Лист</td><td>Листів</td></tr>
<tr><td>Керівн.</td><td>Ващіліна О.В.</td><td></td><td></td><td></td><td>159</td></tr>
<tr><td>Н/контр.</td><td>Базилюк А.М.</td><td></td><td></td><td colspan="2" rowspan="2"></td></tr>
<tr><td>Зав.каф.</td><td>Плескач В.Л.</td><td></td><td></td></tr>
</table>

АНОТАЦІЯ

Бакалаврська робота на тему “Мобільний застосунок гри”.

Робота містить: 60 сторінок, 3 розділів, 20 підрозділів, 2 таблиці, 24 ілюстрації, 30 використаних джерел, 3 додатки.

Ключові слова: Unity, C#, навігаційна сітка, полігональна сітка, ріг.

Актуальністю роботи є створення додатку для проведення вільного часу та покращення аналітичних навичок та реакції людей.

Метою роботи є розвиваюча гра-екшн на базі мобільного застосунку.

Перший розділ присвячений загальному огляду ринку мобільних застосунків та аналізу проблеми. Також в цьому розділу проведене обґрунтування мети вирішення проблеми та поставлена задача.

Другий розділ присвячений аналізу існуючих технологічних рішень та аналізу конкурентів.

Третій розділ присвячений функціоналу додатка, принципам побудови кожної окремої системи та висновком, як ці системи вплинули на розвиток та розробку додатка.

У результаті роботи був створений мобільний додаток для платформи Android та iOS та кілька універсальних інструментів для подальших розробок, які мають потенціал стати незалежними SDK.

Практичним значенням роботи є створення мобільної гри у жанрі екшн та подальше використання інструментів, створених у процесі розробки додатка.

Роботу виконав: Рибас Ілля Романович

Спеціальність: 122 Комп’ютерні науки

ABSTRACT

Bachelor's thesis on "Mobile Game Application".

The paper contains: 60 pages, 3 sections, 20 subsections, 2 tables, 24 illustrations, 30 sources used, 3 appendices.

Keywords: Unity, C#, navigation mesh, polygonal mesh, rig.

The relevance of the work is to create an application for free time and improve people's analytical skills and reactions.

The aim of the work is an educational action game based on a mobile application.

The first section is devoted to a general overview of the mobile app market and an analysis of the problem. Also in this section is a justification of the problem solving objective and the problem.

The second section is devoted to the analysis of existing technological solutions and the analysis of competitors.

The third section is devoted to the functionality of the application, the construction principles of each individual system and the conclusion of how these systems have influenced the development and design of the application.

As a result of the work was created a mobile application for the Android and iOS platforms, and several universal tools for further development, which can become an independent SDK.

The practical value of the work is the creation of a mobile game in the action genre and further use of the tools created during the development of the application.

Written by: Illia Rybas

Specialization: 122 Computer Science

ЗМІСТ

ВІДОМІСТЬ КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА 4
АНОТАЦІЯ 5
ABSTRACT 6
ЗМІСТ 6
ВСТУП 8
Розділ 1. ТЕОРЕТИЧНІ ОСНОВИ ПОБУДОВИ КОМП'ЮТЕРНОЇ ГРИ 9
1.1 Огляд і аналіз програмних систем, принципів і засобів створення мобільної гри 9
1.2 Обґрунтування мети вирішення поставленої задачі 11
1.3 Постановка задачі 13
1.4 Висновок 14
Розділ 2. АНАЛІЗ ПРОГРАМНИХ РІШЕНЬ ДЛЯ РОЗРОБКИ МОБІЛЬНИХ ІГОР 15
2.1 Аналіз існуючих рушіїв для побудови відеоігор 15
2.2 Аналіз інструментів та технології, які використовуються у рушії Unity 17
2.3 Аналіз існуючих ігор у жанрі Hyper Casual 20
2.4 Висновок 22
Розділ 3. ПРОЕКТУВАННЯ, РОЗРОБЛЕННЯ, РЕАЛІЗАЦІЯ МОБІЛЬНОЇ ГРИ 23
3.1 Функціональні можливості проектованої системи 23
3.2 Принципи організації ігрового процесу 29
3.3 Принципи побудови системи зони зору супротивників 31
3.4 Принципи побудови системи сплайнів та маршрутів пересування супротивників на рівні 35
3.5 Принципи побудови системи системи встановлення бомб 41
3.6 Принципи побудови системи програшу та Ragdoll 43
3.7 Принципи побудови механіки руйнування будівлі 48
3.7.1 Принципи побудови інструменту об'єднання дочірніх мешів у один меш 49
3.7.2 Принципи побудови системи інструменту процедурного розрізання полігональної сітки 51
3.7.3 Принципи побудови системи цілісності уламків 54
3.8 Результати тестування мобільної гри 56
3.9 Висновок 57
ВИСНОВОК 58
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 59
ДОДАТКИ 62
Додаток А 62
Source Code проекту 62

ВСТУП

На сьогоднішній день відомо багато досліджень з приводу користі ігор, не тільки настільних, але й відеоігор. В першу чергу гра є чудовим полігоном для вивчення нових навичок. Так відомо дуже багато ігор-симуляторів, які допомагають людям різних професій практикувати їх майстерність - це військові, пілоти, гонщики, будівельники. Також нейробіологи відзначають кілька якостей які допомагають людям у повсякденному житті. Такими якостями є вміння контролювати увагу, думки і почуття, здатність мотивувати себе і проявляти силу волі, співчуття і рішучість.

Актуальність теми цієї дипломної роботи лежить у ідеї створення ще одного способу проведення вільного часу, який допоможе людям покращити їх аналітичні навички та реакцію за рахунок жанру гри - екшн.

Метою дипломної роботи розвиваюча гра-екшн на базі мобільного застосунку.

Об’єктом дослідження дипломної роботи є процеси забезпечення гри.

Предметом дослідження є програмно-технічні рішення та засоби, підходи, принципи створення мобільного кросплатформеного застосунку.

Методи дослідження:

- Спостереження. Для того, щоб обрати тематику та жанр мобільної гри, що буде розробляється, необхідно проаналізувати поточний ринок та сучасні тренди.
- Аналіз та класифікація. Отримані результати спостереження необхідно класифікувати та обрати провідні тренди сучасного ринку.
- Методи математичної обробки результатів дослідження: мобільна гра була розроблена та представлена за допомогою мови програмування C# та Unity.

Практичним значенням створеного додатку є подальше використання інструментів, що були створені протягом розробки цієї мобільної гри.

Під час створення додатку були використані унікальні практики оптимізації застосунку, які показали дуже добрі результати. Це розширило спектр пристроїв, на яких дана гра працює з високою кількістю кадрів за секунду.

Розділ 1. ТЕОРЕТИЧНІ ОСНОВИ ПОБУДОВИ КОМП'ЮТЕРНОЇ ГРИ

1.1 Огляд і аналіз програмних систем, принципів і засобів створення мобільної гри

Одним із способів проведення часу в дорозі є мобільні відеоігри. Такі типи додатків є одним з найпотужніших провайдерів трафіку мобільної реклами.

Виходячи з досліджень компанії Newzoo за 2021 рік, мобільні ігри займають близько 45% усього ринку відеоігор.

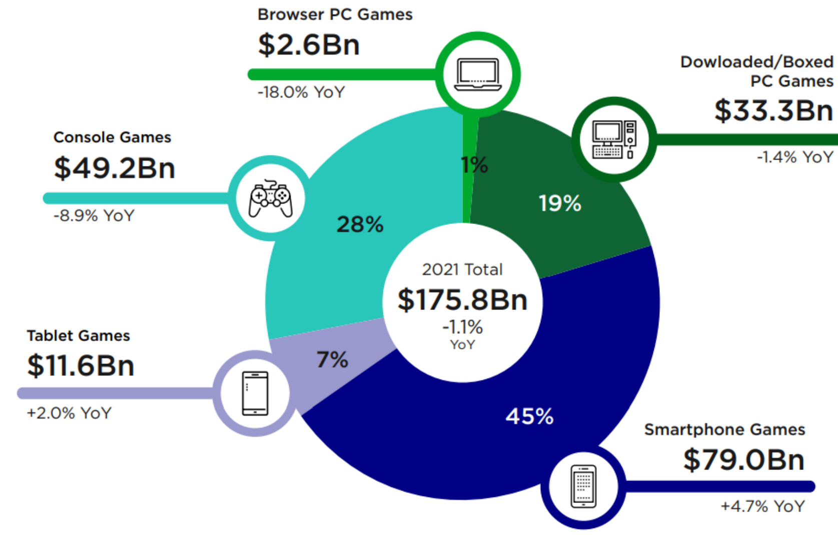

Рисунок 1.1 - Розподіл жанрів відеоігор за даними 2021 року

Це означає, що найбільший попит серед користувачів саме в цьому сегменті. Таке переважання зумовлене низьким порогом входу, тобто людині не потрібен попередній ігровий досвід для того, щоб розібратися в додатку та почати грати. Також середня ігрова сесія складає близько 500 секунд, що є не дуже великою кількістю часу та людині набагато простіше психологічно приділити 10 хвилин грі, ніж приділяти годину або більше більш “серйозним” продуктам. Це означає, що на такі ігри завжди є і буде попит, незважаючи на те, що ринок сильно перенасичений.

Для аналізу поточних трендів на ринку мобільних відеоігор було взято такі ресурси:

- Sensortower;
- Appannie

Для аналізу були обрані відеоігри компанії Voodoo, Supersonic, Ketchapp, які є безумовними лідерами у мобільному сегменті відеоігор.

Функціонал найбільш популярних відеоігор із рейтингу було вивчено та проаналізовано. Ринок гіперказуальних ігор - це ринок, що змінюється кожного тижня та на якому нові потужні тренди з'являються кожного місяця. На момент аналізу, ринок сегментується таким чином за жанрами:

Рисунок 1.2 - Розподіл жанрів на ринку у секторі Hyper Casual

Як можна бачити з діаграми, найбільшу частину ринку займає жанр Action, тому, оскільки в цьому жанрі існує найбільший попит, гру саме такого жанру ми будемо розробляти.

Усі ігри орієнтовані на коротку ігрову сесію та найбільш простий ігровий процес, який вимагає від користувача найпростіші дії: тапи та свайпи.

1.2 Обґрунтування мети вирішення поставленої задачі

Після проведення аналізу поточного мобільного ринку, наявних програмних рішень, а також враховуючи його переваги й недоліки, було встановлено такі висновки:

- Ринок відеоігор з кожним роком стає все більшим і більшим. Це пов’язано з тим, що ігри становляться доступними для все більш широкого кругу

людей. Якщо раніше ігри випускалися в основному для комп'ютерів та консолей, то на теперішній день більшість ігор випускається для мобільних пристроїв, які є у кожного;

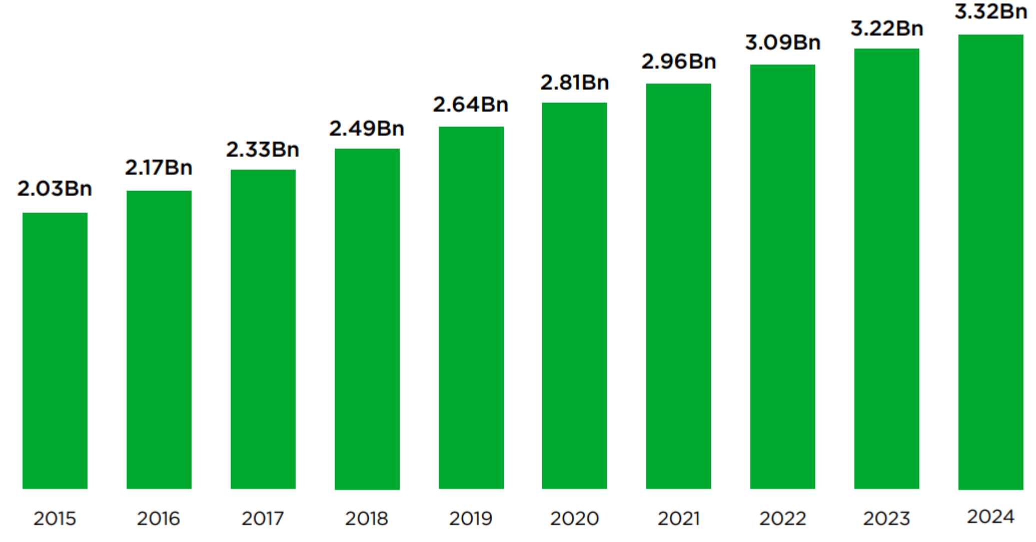

Рисунок 1.2 - Графік зростання кількості гравців відеоігор за даними 2021 року

- За останній рік відбувся значний перерозподіл створюванних ігор під різні платформи. Якщо раніше більшість програм створювалися для платформи iOS, то тепер спочатку більшість ігор випускається під платформу Android. Це пов'язано з тим, що компанія Apple у оновленні iOS 14 почала показувати спливаюче вікно, яке запитує у користувача дозвіл збирати аналітику та формувати його унікальний рекламний портрет в мережі Інтернет. Оскільки таке повідомлення викликає у користувача почуття стежки, багато користувачів забороняє збирати аналітику, що призводить до того, що розробники не можуть отримувати звіти про те, наскільки цікавий для користувачів додаток та де є проблемні місця;

- У багатьох ігор проблеми з залученням уваги, що виражається у високому CPI (cost per install - ціна за установку). Причиною може бути неприваблива графіка або погана якість рекламних відеороликів;

- Оскільки для того, щоб протестувати “цікавість” програмного забезпечення для користувача, необхідно отримати його дозвіл через спливаюче вікно на платформі iOS, то ми в першу чергу будемо створювати додаток для платформи Android. А після цього з усіма отриманими результатами аналітики та покращеним додатком будемо публікувати відео ігру на платформі App Store.

Створення мобільного застосунку комп’ютерної гри є основною метою цієї роботи. Додаток є кросплатформним, що дає змогу використовувати її на різних операційних системах.

1.3 Постановка задачі

Основним завданням дипломної роботи є створення мобільної гри для платформи Android та iOS.

Для виконання поставленої мети необхідно вирішити такі задачі:

- Проаналізувати вже існуючі мобільні відеоігри у жанрі “гіперказульні ігри”;
- На основі результатів аналізу знайти переваги та недоліки мобільних ігор, визначити доповнення до функціоналу мобільної гри;

- Створення усіх необхідних інструментів та розширень редактору для зручної подальшої роботи;
- Створення простого та зрозумілого ігрового процесу;
- Додати заявлений функціонал у додаток;
- Інтеграція сторонніх плагінів та SDK для підключення рекламної мережі та сигналів аналітики;
- Виконати тестування функціоналу;
- Перевірити працездатність програми в декількох операційних системах;
- Перевірити надходження сигналів аналітики до особистого кабінету додатку в AppsFlyer.

1.4 Висновок

На сьогоднішній день ринок мобільних ігор зростає все більше і більше з кожним роком. Це пов’язано з тим, що мобільні пристрої досягають високих потужностей, а це в свою чергу надає можливість створювати більш цікаві додатки для користувачів.

Найбільше на сьогоднішній день виділяється ринок мобільних гіпер казуальних ігор, який займає більшу частину сегменту мобільного геймінгу. Причиною цього є те, що мобільні пристрої розповсюджуються серед людей, які не мають глибокого досвіду гравця та які хочуть дуже легко зрозуміти правила гри та не витрачати на це багато часу.

Існує 3 основних показника, серед яких можна визначити “успішність” створеної гри:

- CPI (Cost Per Install) - ціна за “гравця”. Тобто скільки ми витратили грошей на рекламу для того, щоб залучити користувача;
- Play Time - час гри. Тобто скільки секунд на день користувач проводить в нашому додатку;
- 7-Day Retention - кількість користувачів, які повертаються в додаток на 7 день після становлення.

Саме на ці 3 основних показника ми будемо орієнтуватися при створенні мобільної гри.

Розділ 2. АНАЛІЗ ПРОГРАМНИХ РІШЕНЬ ДЛЯ РОЗРОБКИ МОБІЛЬНИХ ІГОР

2.1 Аналіз існуючих рушіїв для побудови відеоігор

Unity - рушій для розробки комп'ютерних ігор, що дозволяє створювати додатки, що працюють на різних платформах, що включають ПК, ігрові консолі (XBOX, PlayStation, Nintendo Switch і т.д.), мобільні пристрої (Android, iOS), інтернет-додатки та інші.

Основними перевагами є наявність візуального середовища розробки, міжплатформенності і модульної системи компонентів. До недоліків відносять появу складнощів при роботі з багатокомпонентними схемами.

Редактор Unity має простий інтерфейс, який легко налаштовувати, завдяки чому можна проводити налагодження гри прямо в редакторі. Двіжок використовує мову програмування C# для написання скриптів. Раніше підтримувалися також Boo і модифікація JavaScript, відома як UnityScript. Розрахунки фізики виробляє фізичний движок PhysX від NVIDIA. Графічний API - DirectX.

Великою перевагою всіх готових движків є вже готова реалізація системи рендерінгу, фізики та освітлення. Оскільки це дуже складні математичні задачі, реалізація їх з нуля зайняла би дуже багато часу.

Також перевагою Unity є величезне співтовариство програмістів, художників і геймдизайнерів з усього світу, що з одного боку, підвищує шанси знайти команду з такою ж спеціалізацією, а з іншого боку - створює такий формат відносин як Unity Asset Store, в якому розробники діляться готовими рішеннями конкретних завдань та викладають моделі або малюнки, які можуть використовувати інші розробники. Така гнучкість є особливо важливою в розробці мобільних ігор, де головне - це швидкість видання нових додатків.

Unity отримав свою популярність через декілька причин:

1. Низький поріг входу. Оскільки існує попит серед людей, які не знайомі або поверхнево знайомі з програмуванням, Unity зробив ставку на максимально просту інтеграцію в розробку ігор. Тому користувач, який вперше відкрив рушій, може майже одразу почати створювати ігри.
2. Розширюваність. Те, що на Unity створюються лише прості ігри - міф. Насправді багато великих AAA мобільних ігор створені на рушії Unity:

Call of Duty: Mobile, Rainbow Six Siege: Magnum і т.д. Це стало можливим через те, що Unity є легко розирюваним, тому кожна команда розробки може дуже просто створити та інтегрувати всі необхідні інструменти.

3. Потужність. Звісно, Unity не може конкурувати в потужності з такими рушіями як Unreal Engine, проте зі своїми задачами він повністю справляється, особливо беручи до уваги поточний тренд в індустрії розробки ігор - ECS, який багатократно підвищує продуктивність додатку.

2.2 Аналіз інструментів та технології, які використовуються у рушії Unity

Для розробки мобільної гри, було відібрано та використано технології для створення мобільних кросплатформенних застосунків: Unity, C#, ShaderLab, HLSL. Дані технології та їх інструментарій дозволяють в повному обсязі відтворити вимоги технічного завдання.

C# — ООП мова програмування зі статичною типізацією для .NET. Синтаксис C# схожий до C++ і Java. Мова підтримує поліморфізм, перевантаження операторів, вказівники на функції-члени класів, атрибути, події, властивості, винятки, коментарі у форматі XML. C# стандартизований в ECMA та ISO.

Ключові особливості мови C#:

- Орієнтованість на компоненти
- Система типів і їх безпечність
- Автоматична робота за пам'яттю
- Використання CLR

.NET Framework — це програмна технологія, надана Microsoft як платформа для створення звичайних і веб-додатків. Це багато в чому є продовженням ідей і принципів, закладених у технології Java. Однією з філософій .NET є сумісність служб, написаних різними мовами.

.NET ділиться на дві основні частини - середовище виконання та засоби розробки.

Common Language Runtime (CLR), компонент Microsoft .NET Framework, є віртуальною машиною, яка запускає всі мови .NET Framework. CLR перекладає вихідний код у байт-код у IL, компіляції, реалізованої Microsoft під назвою MSIL, і надає програмам MSIL доступ до .NET Framework або так званої .NET FCL (Framework Class Library).

Код і ресурси IL, включаючи растрові зображення та рядки, зберігаються в збірках, як правило, з розширенням .dll. Колекція містить маніфест з інформацією про тип, версію, мову та місцевість колекції.

Середовище CLR є реалізацією специфікації CLI (Common Language Infrastructure).

CLI (Common Language Infrastructure) - специфікація загальномовної інфраструктури. Специфікація CLI визначає архітектуру виконавчої системи .NET - CLR і сервіси, що надаються CLR виконуваних програм, класи, що надаються бібліотекою BCL, синтаксис і мнемоніку загального проміжного мови (CIL).

PhysX - сполучне програмне забезпечення, багатоплатформний фізичний движок для симуляції ряду фізичних явищ, а також комплект засобів розробки (SDK) на його основі. Підпрограмне забезпечення PhysX SDK дозволяє розробникам ігор уникати написання власного програмного коду для обробки складних фізичних взаємодій в сучасних комп'ютерних іграх

DirectX — це набір функцій API, призначених для легкого й ефективного вирішення завдань програмування ігор та відео.

ShaderLab - це декларативна мова, яку ви використовуєте у шейдерних вихідних файлів. Він використовує синтаксис вкладених дужок для опису об'єкта шейдера.

У ShaderLab можна визначити безліч параметрів, але найпоширенішими є такі:

- Визначення загальної структури об'єкта шейдера;
- Використання блоків коду для додавання шейдерних програм, написаних мовою HLSL;
- Використання команд для встановлення стану рендерингу GPU перед виконанням шейдерної програми або виконання операції, пов'язаної з іншим проходом;

- Розкриття властивостей коду шейдера, щоб їх можна було редагувати в інспекторі матеріалів та зберегти як частину матеріального активу;
- Вказувати вимоги до пакету для субшейдерів та проходів. Це дозволяє Unity запускати певні SubShaders та Passes лише у тому випадку, якщо у проекті Unity встановлені певні пакети;
- Визначення поведінки відкату для випадків, коли Unity не може запустити жодного з SubShaders з об'єктом Shader на поточному обладнанні;

HLSL (High Level Shader Language) - C-подібна мова високого рівня для програмування шейдерів. Був створений корпорацією Microsoft і включений в пакет DirectX 9.0.

HLSL дозволяє створювати шейдери шести типів:

- Піксельні шейдери;
- Вершинні шейдери;
- Геометричні шейдери;
- Обчислювальні шейдери;
- Шейдери теселяції;

2.3 Аналіз існуючих ігор у жанрі Hyper Casual

Розглянемо гру у жанрі Action, схожу за ідеєю на гру, яку ми плануємо розробити - Mr Spy.

У корні проекту лежить проста механіка - тап та утримання, що ж доволі гарним рішенням, оскільки дуже зручно грати однією рукою десь у транспорті.

Також проект є дуже привабливим та пропрацьованим візуально.

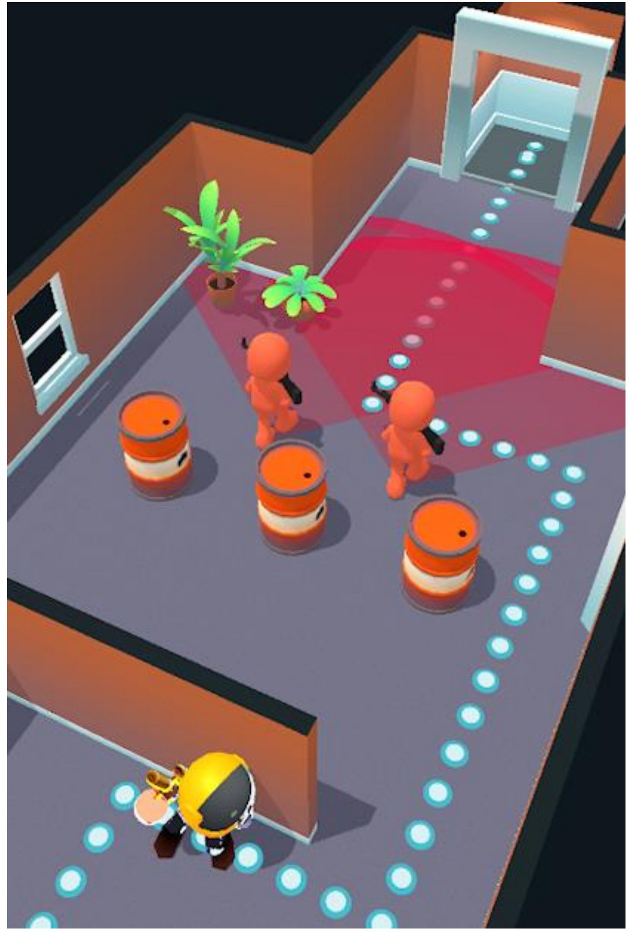

Рисунок 2.1 - ігровий процес Mr Spy

Ідеєю для покращення проекту можуть бути додаткові завдання на рівні та більша свобода рухів, що може підвищити залученість гравця.

Також варіантом покращення гри може бути надання можливості гравцю зробити приємну дію після закінчення рівня, наприклад, показати результати його дій на рівні, як це прийнято в більшості сучасних гіпер казуальних ігор.

Також у цій грі є супротивники, що реалізують механіку “обмеженої баченості”. Це є дуже гарною механікою, що часто показує високі результати залученості гравця та підвищує інтерес до гри, надаючи можливість гравцю проявити свої навички та реакцію.

Рисунок 2.2 - ігровий процес Mr Spy

Також в грі є можливість програти, що сигналізує гравці про неправильні дії з його боку. У наш проект буде необхідно буде додати таку саме функціональність.

2.4 Висновок

Отже, в результаті проведення другої частини дослідження були проаналізовані існуючі рушії та інструменти для створення мобільних відео ігор. Було прийняте рішення використовувати наступні інструменти:

- Рушій Unity;
- Мова програмування C#;
- Мови програмування шейдерів ShaderLab, HLSL;

Були проаналізовані наявні конкуренти у жанрі екшн. Серед них були обрані найкращі підходи до взаємодії з користувачем.

Розділ 3. ПРОЕКТУВАННЯ, РОЗРОБЛЕННЯ, РЕАЛІЗАЦІЯ МОБІЛЬНОЇ ГРИ

3.1 Функціональні можливості проектованої системи

Для реалізації поставлених цілей проектованій системі, було розглянуто та проаналізовано якісні ігри, які затвердилися на ринку та виділено функціонал, який найкраще підходить до даної мобільної відеогри та подобається гравцям.

Таблиця 2.1 — Функціонал мобільної гри “Plant the Bomb”

Назва функціоналу	Опис функціоналу

Механіка пересування головного героя	Основна механіка ігри. За допомогою джойстика на екрані дозволяє пересувати головного героя у заданих межах ігрової мапи.
Система джойстику	Дозволяє створити ще одну систему вводу з боку користувача. Зображує екранний джойстик та змінює його форму в залежності від вводу користувача.
Механіка пересування ворогів	Дозволяє пересувати ворогів на рівні за заданою траєкторією.
Механіка установки бомби	Дозволяє встановити бомбу у заданій точці.
Механіка прогресу встановлення бомби	Візуальне відображення прогресу поточної бомби.
Механіка стрілянин	Дозволяє керувати частотою створення куль з вказаної зброї.
Механіка куль	Дозволяє створювати тверде тіло, яке взаємодіє з системою здоров’я та корегує її значення.
Система здоров’я	Дозволяє створити систему здоров’я певного об’єкта чи персонажу з вихідними сигналами про те, в якому стані на поточний момент знаходиться об’єкт.
Система зору	Дозволяє створити поле зору та перешкод для зору персонажу. Якщо певний об’єкт потрапляє у задане поле та на шляху до нього не лежить перешкода, то персонаж реагує певним заданим чином.
Система станів ворога	Дозволяє легко створювати нові стани та переходи між ними для ворога.
Система патрулювання	Дозволяє створити шлях патрулювання та назначити дії, які виконує персонаж при патрулювання, включаючи очікування на точці та час

	пересування між ними.
Система підозри	Дозволяє створити стан підозри для персонажа, у якому він буде шукати ворожий об’єкт у разі потрапляння у поле зору.
Система агресії	Дозволяє створити систему агресії для персонажу у разі виходу зі стану підозри. У ції системі можна задати будь-які дії та реакції у напрямку супротивника. У даному додатку як реакція використовується система стрілянини.
Система екіпірування	За допомогою цієї системи можливо екіпірувати персонажа будь-яким об’єктом та підлаштувати систему анімації, Rig та IK відповідним чином до об’єкта. У наведеному додатку в якості об’єктів для екіпірування використовуються зброя (у випадку ворогів) та бомба (у випадку протагоніста).
Механіка збереження прогресу	Дозволяє зберегти прогрес гравця (останній пройдений рівень). При наступному вході в гру підгрузиться рівень, на якому зупинився гравець. Прогрес зберігається автоматично та не потребує додаткових дій від гравця.
Система нарізання 3D об’єкта	Дозволяє нарізати будь-який об’єкт на менші об’єкти в залежності від вхідних параметрів.
Система важких тіл	Дозволяє надати нарізаним об’єктам фізичним властивостей в залежності від вхідних параметрів та типу матеріалу.
Система твердих тіл	Дозволяє надати об’єктам можливість стикатися та реагувати на інші тверді тіла.
Система колайдерів	Дозволяє створити для об’єкта його

	фізичну модель для фізичної частини движка.
Система фізичних бомб	Дозволяє створити фізично реалістичні бомби для впливу да наднння імпульсної сили об’єктам із системи важких тіл.
Система цілісності об’єкту	Дозволяє задати параметри цілісності для важкого тіла, після чого тіло буде вважатися пошкодженим та буде реагувати в залежності від заданих параметрів чином. У наведеному додатку існує кілька стадій пошкодженості об’єкта.
Система підриву	Дозволяє налаштувати епізод відеогри “Підрив”, починаючи від системи зворотнього зв’язку від підриву до налаштування фізичної реакції на підрив.
Система гвинтокрила	Дозволяє налаштувати маршрут та реакції протагоніста на появу у полі зору гвинтокрилу.
Система фізичного оточення	Дозволяє налаштувати вагу та хрупкість предметів оточення. Що в результаті створює повністю інтерактивне оточення, яке можна пересувати.
Система об’єднання мешів	Дозволяє створити єдиний меш з кількох менших мешів для того, щоб у подальшому передати цей меш системі нарізання об’єктів.
Система глобальної відладки	Система глобальних подій, яка дозволяє тестувати основні частини гри з інспектора.
Завантаження рівнів	Завантажує рівні у заданій послідовності після завершення попереднього рівня.
Універсальна система проектування рівнів	Розроблювана система має містити можливість створення ігрових рівнів

	універсальним способом, який не буде відрізнятися від рівня до рівня.
Тактильна віддача (Haptics)	Кожна задана дизайнером дія гравця має супроводжуватися тактильною віддачею - вібрацією різної сили та тривалості.
Система частинок	Кожна задана дизайнером дія гравця має супроводжуватися візуальним ефектом різної тривалості.
Система камер (Cinemachine)	Різні ракурси в середині гри, а також плавний перехід між камерами всередині гри.
Система тіней	Гра має містити тіні з заданими дизайнером насиченістю, довжиною та кутом.
Система матеріалів та шейдерів	Розроблювана система повинна мати можливість налаштовувати шейдер та матеріал кожного окремого ігрового об’єкта.
Система сплайнів	Дозволяє малювати та спроектувати схему маршруту ворогів на рівні.
Система сцен та епізодів	Дозволяє розбити ігровий процес на окремі частини та працювати з ними окремо, а після завершення розробки з’єднати їх у єдину послідовність.
Система меню	Дозволяє створювати різні вікна користувальницького інтерфейсу за допомогою кодогенерації.
Система уповільненого часу	Дозволяє уповільнювати час у залежності від вхідних параметрів
Система Post-Process	Дозволяє впливати повністю на всі наявні камери та створювати для них ефекти пост-обробки та також керувати цими ефектами.
Система зворотнього зв’язку	Комплексна система, що дозволяє налаштовувати усі наявні системи

	зворотнього зв'язку через інспектор движка Unity. Ця система включає в себе систему частинок, анімації, систему твинів, систему вібрацій та багато іншого.
Система твинів	Дозволяє створювати окремі одиниці алгоритмів типу Lerp та об'єднувати їх у ланцюги. Тип Lerp дозволяє керувати будь-якими типами даних.
Система анімацій	Дозволяє створювати стани анімацій та переходити між ними за допомогою вказаних сигналів.
Система Rig	Дозволяє розподілити модель персонажу на гуманоїдні частини тіла та працювати з кожною частиною як з окремим фізичним тілом.
Система IK	Дозволяє візуалізувати реалістичні ланцюги гуманоїдного тіла. За допомогою алгоритма інвертивної кінематики є можливість створювати реалістичні анімації рухів та при пересуванні лише однієї частини тіла (наприклад ступні) не задумуватися про пересування інших кісток, які пов'язані біологічно зі ступнею. Алгоритм виконає необхідні дії.
Система Blend Shapes	Дозволяє змішувати різні анімації та кадри анімацій за допомогою алгоритма Lerp.
Система Ragdoll	Дозволяє відключити усі внутрішні сили рігу гуманоїда та передати контроль над частинами тіла зовнішнім силам (наприклад, гравітації).
Система аналітики	Дозволяє відправляти сигнали у особистий кабінет аналітики додатку за допомогою спрощеного інтерфейсу.
Система прогресу рівню	Дозволяє блокувати та розблокувати

	можливість перейти на наступний рівень в залежності від того, чи виконав гравець усі поставленні перед ним завдання

3.2 Принципи організації ігрового процесу

Усі основні частини гри розподілені на сцени. Основний розподіл такий:

- Game - основна сцена, з якої відбувається запуск додатку. Саме тут ініціалізуються усі сервіси та контролюючі елементи логіки;
- Level_N - конкретний рівень гри, на якому знаходяться унікальні для рівня речі: оточення та супротивники;
- BuildingDemolition - сцена руйнування будівлі. Оскільки ця частина гри є окремим логічним елементом та ніяк не пов’язана з іншими частинами, то було прийнято рішення винести її в окрему сцену.

Послідовність рівнів була побудована за допомогою типу даних ScriptableObject. У даному додатку було створено 5 унікальних рівнів, які повторюються після завершення останнього рівня.

ScriptableObject — це контейнер даних, який можна використовувати для збереження великих обсягів даних, незалежно від екземплярів класу.

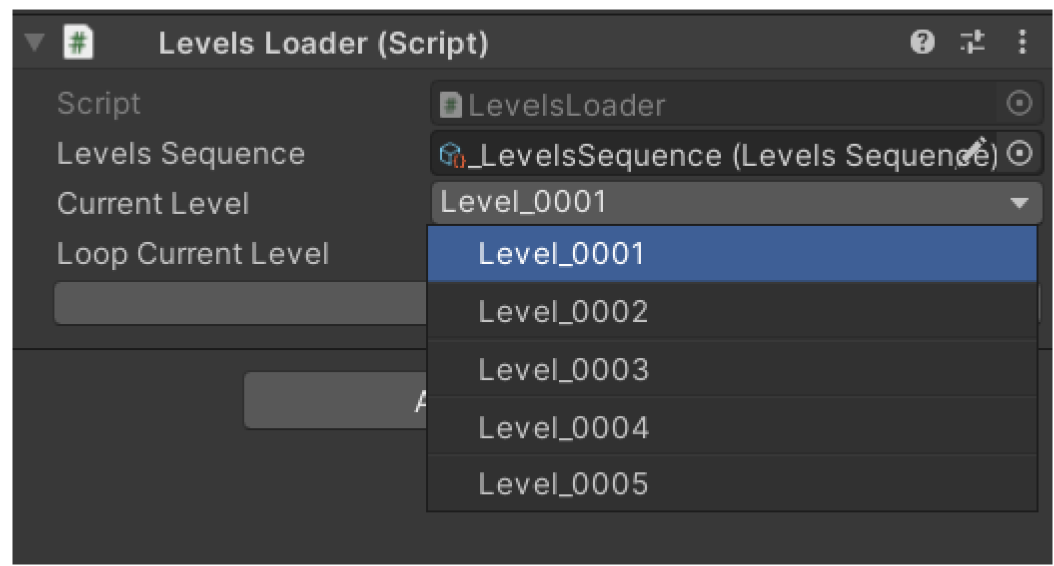

Рисунок 3.1 - Інструмент Levels Loader в інспекторі рушія Unity

Більшість логічних сутоностей (наприклад, логіка головного героя) було організовано за допомогою паттерна MVC з вводом у Controller.

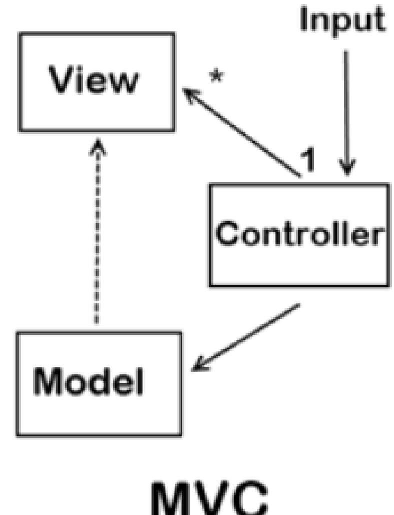

Рисунок 3.2 - Схема паттерну MVC

У рамках шаблону MVC програма поділяється на три окремі, але взаємопов'язані частини з розподілом функцій між компонентами. Model відповідає за зберігання даних та їх структуру. View відповідальний за представлення цих даних користувачеві, тобто інтерфейс програми. Controller керує компонентами, отримує сигнали у вигляді реакції на дії користувача і передає дані у модель.

- Модель є центральним компонентом шаблону MVC і відображає поведінку застосунку, незалежну від інтерфейсу користувача. Модель стосується прямого керування даними, логікою та правилами застосунку.
- Вигляд може являти собою будь-яке представлення інформації, одержане на виході, наприклад графік чи діаграму.
- Контролер одержує вхідні дані й перетворює їх на команди для моделі чи вигляду.

3.3 Принципи побудови системи зони зору супротивників

Змінні, які використовує система зору супротивника:

- View Radius - радіус, в якому супротивник перевіряє наявність своєї цілі;
- View Angle - кут, який є кутом огляду супротивника, щоб симулювати людський зір. Оскільки людина не може дивитися потилицею, то ми симулюємо зір “з очей” за допомогою цього кута;
- Target Mask - шар, який шукає система зору. Якщо об’єкт з таким шаром потрапляє у поле зору, то система надсилає сигнал;
- Obstacle Mask - шар, який є перешкодою для зони зору супротивника. Якщо система знайшла об’єкт з Target Mask, але на шляху до цього об’єкта лежить об’єкт за шаром Obstacle Mask, то система не надішле сигнал.

Система зору побудована наступним чином:

1. Супротивник кожен кадр перевіряє об’єкти навколо себе за допомогою технології Trigger Collider.
2. Якщо такий об’єкт потрапляє в зону навколо супротивника, то система перевіряє, чи кут між “очима” супротивника та його ціллю менше ніж View Angle, то система йде далі.
3. Наступним кроком система перевіряє, чи не лежить перешкода між супротивником та та його ціллю за допомогою Raycast. Якщо супротивник не знайшов перешкоду, то система надсилає сигнал про виявлення цілі.

Вигляд створеної системи в редакторі Unity:

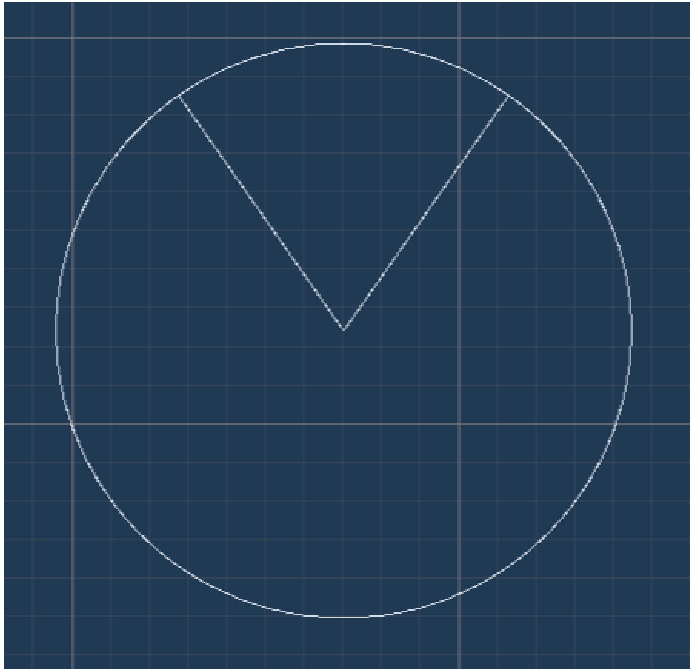

Рисунок 3.3 - Вигляд інструменту Field Of View у вікні Scene

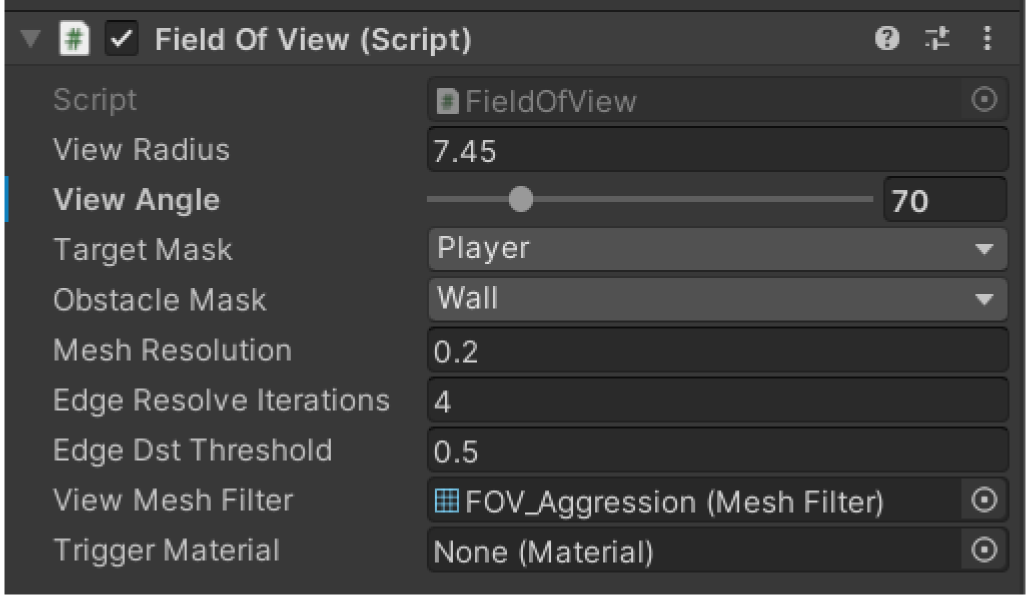

Рисунок 3.4 - Вигляд інструменту Field Of View у вікні Inspector

Кастинг променів (raycasting) - це найпростіший з багатьох алгоритмів рендеринга комп'ютерної графіки, в яких використовується геометричний алгоритм трасування променів. Спрощено, принцип роботи можна виразити так: "З точки А в точку Б випускається невидимий промінь. Якщо він перетинається з коллайдером об'єкта, система сповіщає про це користувача".

В движку система ділиться на дві зони: візуальна та фізична. Візуальна відповідає за те, як відображаються об’єкти на екрані. Сюди входять шейдери, матеріали та будь-яка робота з текстурами.

Фізична ж частина відповідає за те, як взаємодіють між собою об’єкти згідно з фізичними законами. Сюди входить логіка зіткнень, гравітація, прикладення будь-яких сил та інше.

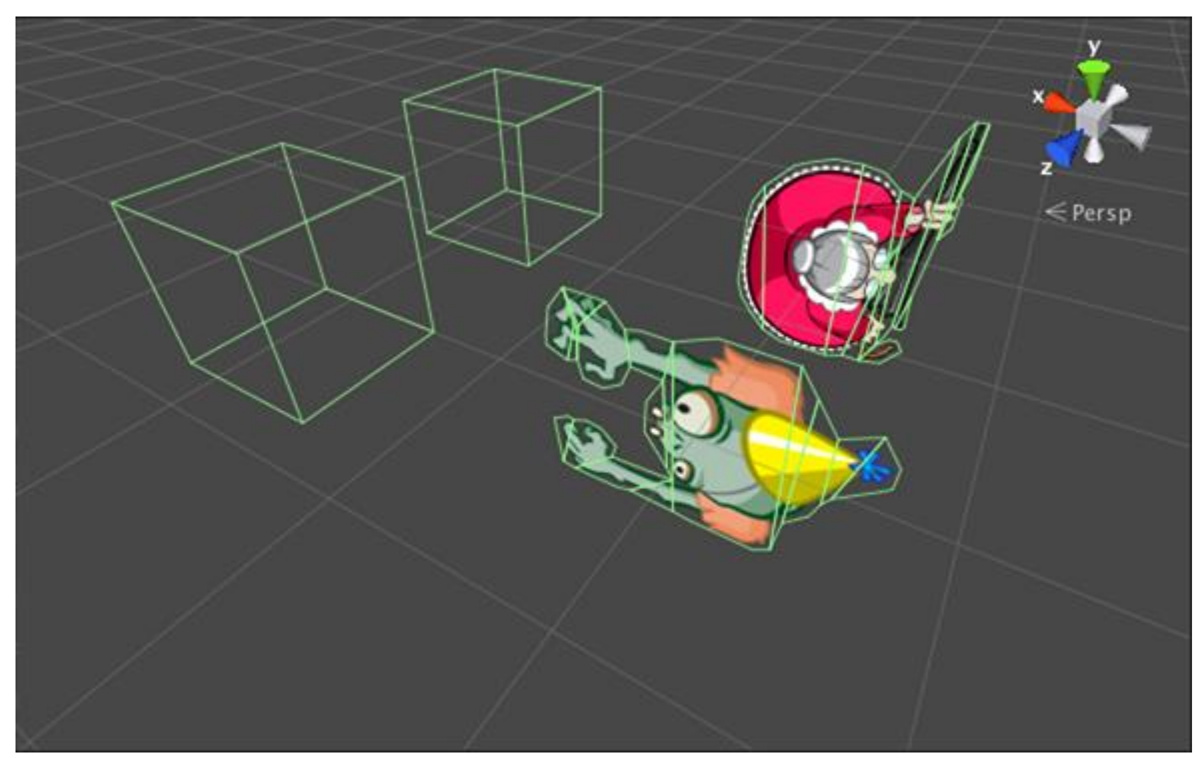

Рисунок 3.5 - Коллайдери

Коллайдер - це представлення об’єкта у фізичній частині гри. Це невидимая фігура, яку фізичний движок буде враховувати при обчисленні будь-якої взаємодії цього об’єкта з іншими фізичними частинами гри. Тому візуальне відображення об’єкта та його фізичне представлення можуть суттєво відрізнятися.

Однією з найскладніших частин побудови системи зору є візуальна складова, оскільки необхідно будувати меш власноруч в залежності від вхідних параметрів радіусу та кута.

Задля того, щоб ця частина системи не забирала дуже багато ресурсів GPU було прийнято ввести декілька додаткових параметрів для візуальної складової.

- Mesh Resolution - зі скількох трикутників буде складатися меш зони зору. Чим більше число, тим більш плавна виходить картинка;
- Edge Resolve Iteration - з якою частотою буде вираховуватися та будуватися візуальна складова;
- Edge Dst Threshold - допустима похибка для вимірювання дистанції між кутами. Необхідно для того, щоб вирішити візуальну помилку, коли супротивник намагається "дивитися" між двох стін.

Результат роботи системи:

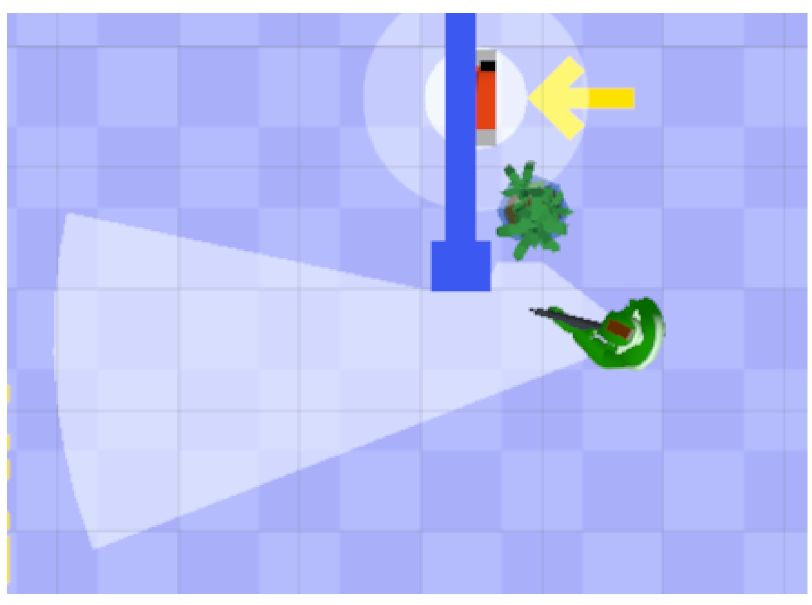

Рисунок 3.6 - Результат роботи Field Of View

Як видно з малюнку, біла область - це зона зору супротивника. Вона підлаштовується під оточуюче середовище. У даному випадку супротивник не буде бачити боковим зором, що відбувається за стіною справа від нього.

3.4 Принципи побудови системи сплайнів та маршрутів пересування супротивників на рівні

Для того, щоб побудувати достатньо реалістичну модель пересування супротивника по мапі, були використані 2 технології: NavMesh та система сплайнів.

Навігаційна сітка — це абстрактна структура даних, яка використовується в програмах штучного інтелекту, щоб допомогти агентам знайти шлях у складних просторах.

Це набір двовимірних опуклих багатокутників, які використовуються для визначення того, які частини середовища є прохідними. Іншими словами, персонажі в грі можуть вільно ходити в цих зонах, не заважаючи їм дерева, лавки чи інші перешкоди, які є частиною навколишнього середовища.

Суміжні багатокутники з’єднуються в один граф. Визначення контуру в одному з цих багатокутників можна просто виконати по прямій, оскільки багатокутник є опуклим і може проходити наскрізь. Призначення шляху між багатокутниками в сітці можна виконати за допомогою одного з кількох алгоритмів пошуку графів, таких як A*. Агенти в navmesh можуть уникнути дорогих обчислень тестів для виявлення зіткнень з перешкодами, які є частиною середовища.

Для того, щоб використовувати навігаційну сітку в Unity необхідно створити та налаштувати агента та запекти оточуюче середовище під характеристики цього агента. Характеристиками можуть бути висота, ширина, максимальна висота, на яку може забратися агент, а також максимальний схил, по якому може рухатися агент.

Ось так виглядає запечена навігаційна сітка в одному з рівнів гри. Світла синя область - це зона, якою може рухатися агент.

Рисунок 3.7 - Навігаційна сітка

Для власної системи побудови сплайнів була використана крива Без'є.

Крива Без'є — це параметрична крива, яка використовується в комп'ютерній графіці та суміжних областях. Набір дискретних «контрольних точок» визначає плавну безперервну криву за допомогою формули. Зазвичай крива призначена для апроксимації реальної форми, яка в іншому випадку не має математичного представлення або представлення якої невідоме або занадто складне.

Крива Без'є — параметрична крива, вигляду

$$\mathbf{B}(t) = \sum_{i=0}^{n} \mathbf{b}_{i,n}(t)\mathbf{P}_i, \qquad t \in [0,1]$$

де

Pi - опорні вершини,

$$\mathbf{b}_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i}$$

- поліноми Бернштейна, вони є базисними функціями кривої Без’є.

Властивості кривої Без’є:

- Безперервність заповнення сегментів між початковою та кінцевою точками;
- Крива завжди лежить у межах графіка, утвореного лініями, що з'єднують контрольні точки;
- Якщо контрольних точок лише дві, відрізок прямої;
- Пряма утворюється лише тоді, коли контрольні точки лежать на прямій;
- Крива Без'є симетрична, тобто обмін положенням початкової і кінцевої точки (зміна напрямку траєкторії) не впливає на форму кривої;
- Масштабування та зміна масштабу кривої Без'є не призведе до її дестабілізації, оскільки вона математично «афінно інваріантна»;
- Зміна координат хоча б однієї точки призводить до зміни форми всієї кривої Без'є;
- Будь-яка частина кривої Без'є також є кривою Без'є;
- Ступінь кривої завжди на один менший за кількість контрольних точок. Наприклад, у трьох контрольних точках форма кривої є параболою;
- Коло не можна описати параметричним рівнянням кривої Без'є;

Результат роботи алгоритму після реалізації:

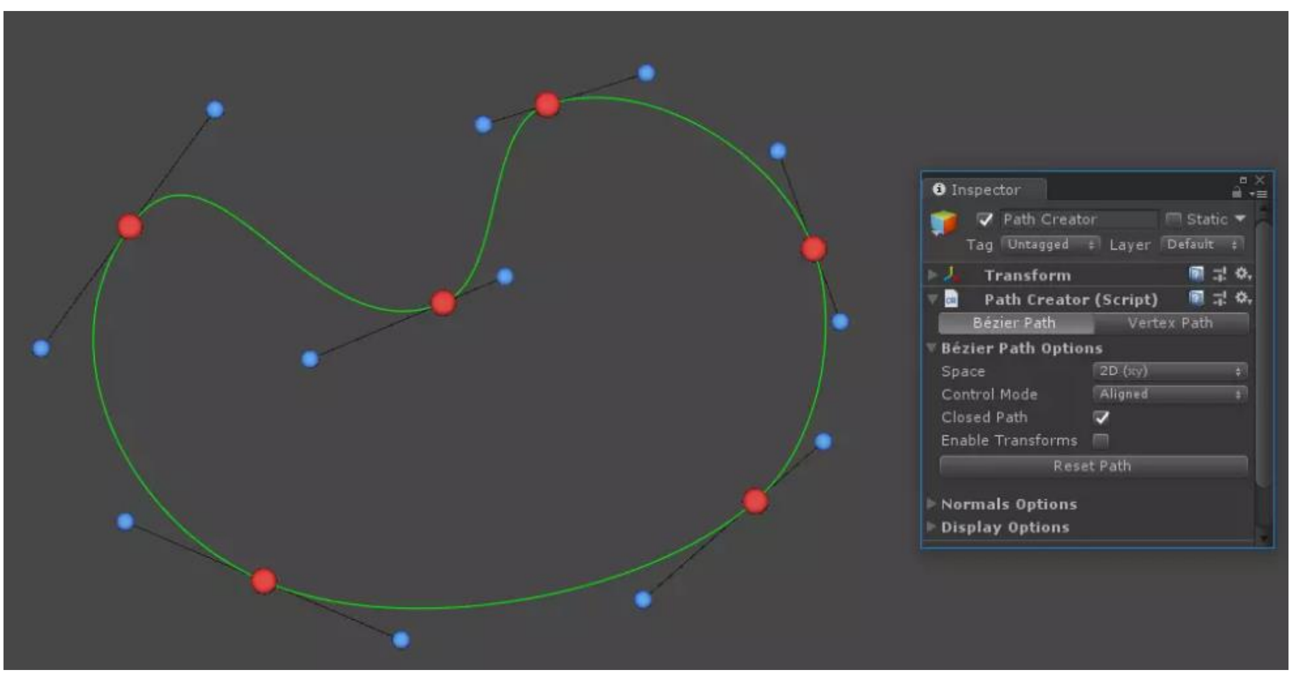

Рисунок 3.8 - Результат роботи кривих Без’є у вікні Scene

Як видно з малюнка, крива виходить дуже плавна та будується сама в залежності від заданих червоних точок та вздовж прямих з блакитними точками.

Для того, щоб створити достатньо реалістичну поведінку патрулювання було прийнято рішення створити три типи точок, які буде обходити супротивник:

- No Action - точка, яку супротивник просто проходить та не затримується, необхідна лише для того, щоб задати шлях патрулювання
- Stoppable - точка, на якій супротивник затримується на кілька секунд, імітуючи поведнку обстеження території
- Reverse Order - точка, у якій супротивник розвертається та починає слідувати в протилежному напрямку

Усі ці параметри налаштовуються в інспекторі. Наприклад, отак налаштовується тип точки патрулювання:

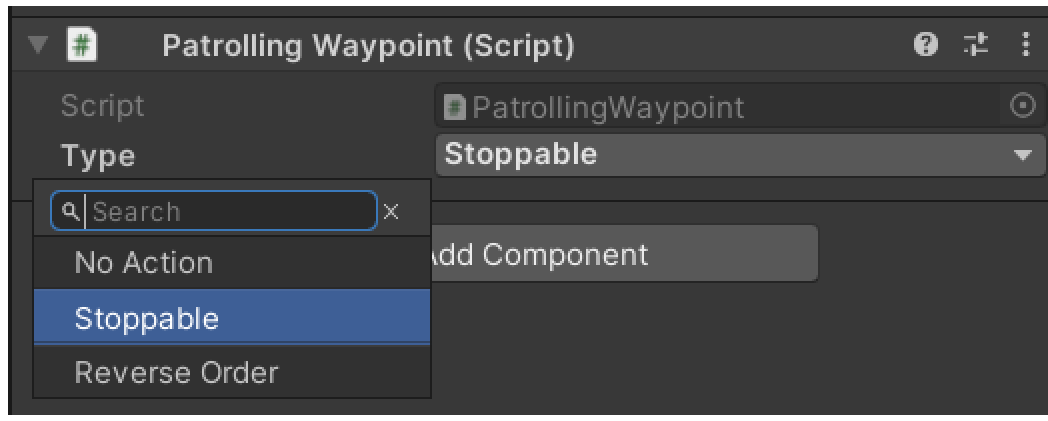

Рисунок 3.8 - Компонент Patrolling Waypoint

Ось так виглядає побудований шлях патрулювання на одному з рівнів. Червона лінія - це шлях, яким слідує супротивник.

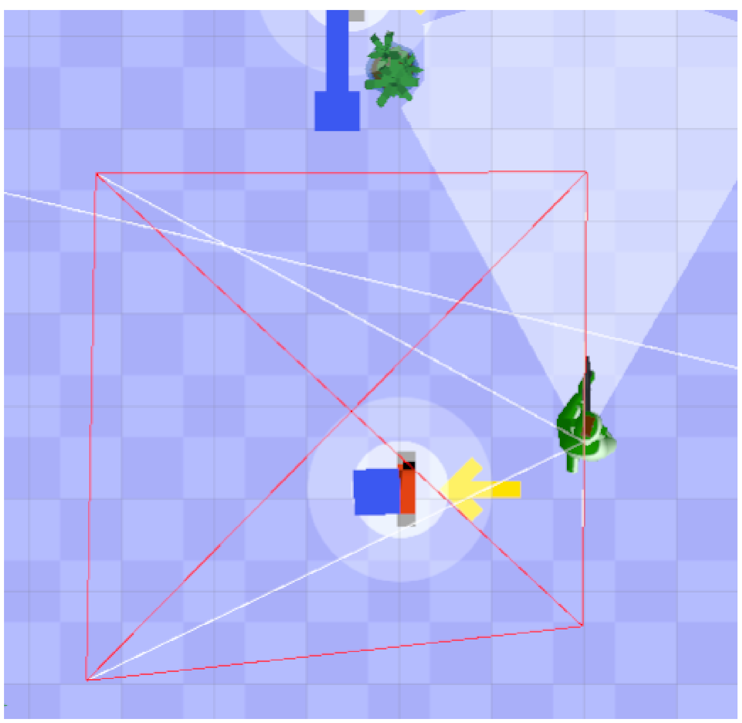

Рисунок 3.9 - Результуюча крива маршруту для супротивника

В результаті розробки була створена досить гнучка система пересування супротивників, за допомогою якої були створені усі супротивники у грі.

3.5 Принципи побудови системи системи встановлення бомб

Основним елементом ігрового процесу та ключовим завданням для гравця є встановлення бомб у відмічені на мапі точки.

Не зважаючи на те, що це є основним функціоналом у грі, її реалізація не є настільки складною, тому були використані базові техніки розробки програмного забезпечення на движку Unity.

В корені системи лежить робота з колайдерами та користувацьким інтерфейсом для зручного орієнтування у ігровому просторі.

У своїй суті кожна точку для установки бомби являє собою Trigger Collider, який посилає сигнал у разі потрапляння у нього об’єкта з маркером “Гравець”.

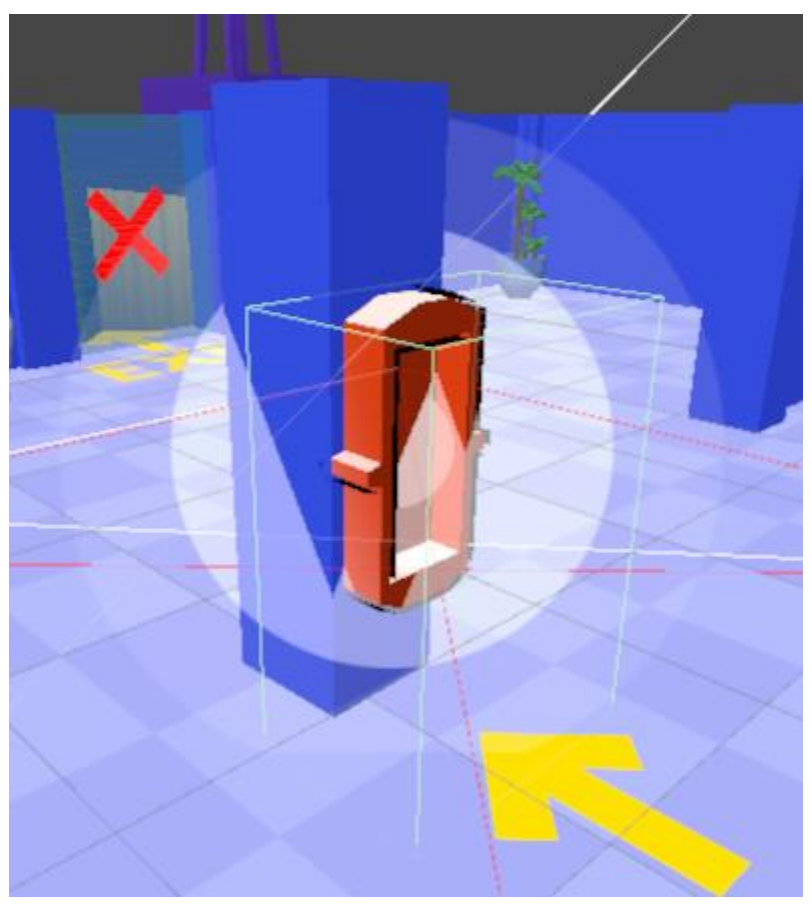

Рисунок 3.10 - Коллайдер для взаємодії з гравцем

Якщо гравець потрапляє у зону взаємодії з точкою для встановлення бомби, з’являється відповідний інтерфейс про прогресс встановлення бомби.

Рисунок 3.11 - Користувальницький інтерфейс закладання бомби

Встановлення займає певний час, тому над головою ігрового персонажу з'являється круговий сектор, який поступово заповнюється червоною лінією в залежності від того, скільки ігровий персонаж стоїть біля точки встановлення бомби.

Рисунок 3.12 - Користувальницький інтерфейс успішного закладання бомби

Після завершення встановлення бомби з’являється відповідний користувальницький інтерфейс, що повідомляє про успішну операцію.

Для того, щоб завершити рівень, гравець повинен встановити усі бомби, про що говорить інтерфейс.

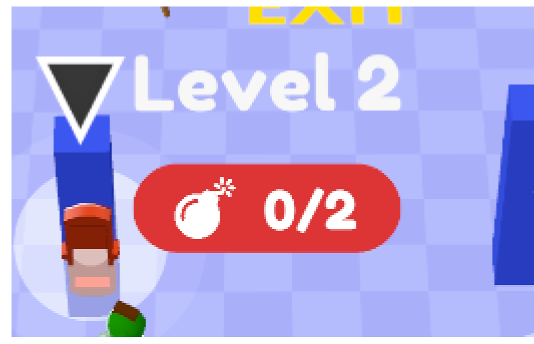

Рисунок 3.13 - Користувальницький інтерфейс цілей на рівні

Коли всі завдання виконані, то відкривається вихід з рівня та гравець може перейти до наступної стадії гри.

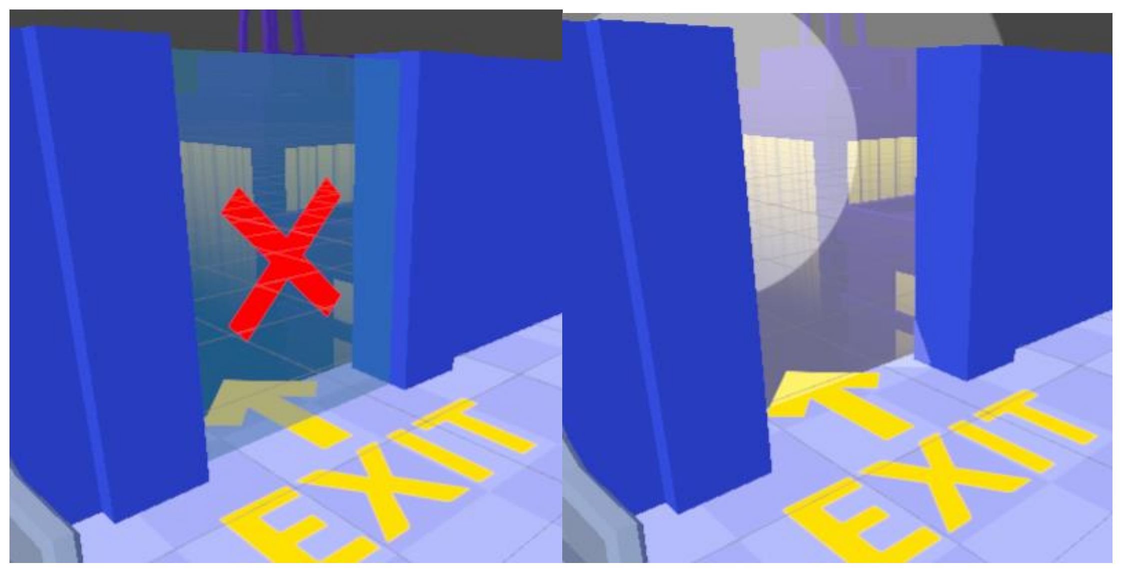

Рисунок 3.14 - Заблокований та розблокований вихід з рівня

3.6 Принципи побудови системи програшу та Ragdoll

Оскільки у кожній грі повинні бути виграш і програш, щоб сповіщати гравця про його успіхи, у цій грі програш також був реалізований. В даному проекті програш зав’язаний на системі здоров’я протагоніста. Рівень здоров’я може змінюватися через стрілянину, що виконують супротивники, які можуть

входити у режим агресії. Якщо рівень здоров'я падає нижче нуля, це означає, що гравець програв. Після цього з'являється вікно програшу та надається можливість почати рівень спочатку.

Програш було вирішено зробити таким чином - якщо протагоніст отримує критичне пошкодження від кулі, його тіло переходить у режим Ragdoll, екран стає сірим і після цього з'являється екран програшу.

Фізика Ragdoll - вид процедурної анімації, якою замінили статичну анімацію.

Оскільки прорахунок фізичних параметрів у реальному часі віднімає значну частину обчислювальної потужності процесора, багато ігор використовують спрощену структуру скелету «ганчіркової ляльки». Зокрема:

- Кістки кінцівок (пальці) зазвичай не анімуються;
- Замість реальних кінцівок людського тіла використовуються спрощені суглоби;
- Спрощена модель зіткнень набагато краще визначає взаємодія з іншими твердими тілами, ніж зіткнення з геометрією.

Основною перевагою ragdoll є значно правильніша взаємодія з навколишнім середовищем. Створення анімації на кожний можливий ігровий випадок займе дуже багато часу, а завдяки фізиці «ганчіркової ляльки» ігровий рушій може генерувати точні сцени в реальному часі.

Рисунок 3.15 - Екран програшу

Для того, щоб увімкнути можливість активувати Ragdoll, необхідно вимкнути параметр isKinematic на всіх частинах тіла гуманоїдного ріга та надати силу, у якому напрямку ми хочемо штовхнути тіло.

Rigidbody - вбудований компонент рушій Unity, що надає тілу фізичні властивості, та дозволяє йому реагувати на зовнішні сили, такі як сила тяжіння. Увімкнений параметр isKinematic означає, що тіло не буде реагувати на жодну зовнішню силу.

У компоненті rigidbody доступні такі наступні властивості:

Таблиця 2.2 — Параметри компонента Rigidbody

Mass	Маса об'єкта (у кілограмах за замовчуванням)
Drag	Який опір повітря впливає на об'єкт під час руху від сил. 0 означає відсутність опору повітря, а нескінченність змушує об’єкт негайно зупинятися.

Angular Drag	Який опір повітря впливає на об'єкт при обертанні від крутного моменту. 0 означає відсутність опору повітря.
Use Gravity	Якщо ввімкнено, на об’єкт впливає сила тяжіння.
Is Kinematic	Якщо ввімкнено, об’єкт не буде управлятися фізичним механізмом, і ним можна керувати лише за допомогою його трансформації.
Collision Detection	Використовується для запобігання проходження об’єкта через інший у випадку, якщо вони швидко рухаються.

Для того, щоб керувати часом, використовується параметр time та deltaTime класу Time.

time - змінна, що відображає час на початку поточного кадру; deltaTime - змінна, що відображає різницю часу між двома послідовними кадрами.

Таким чином синхронно зменшуючи значення змінних time та deltaTime ми можемо досягти ефекту уповільнення.

Також для того, щоб зробити екран рівномірно сірим, була використана техніка Post-processing. Треба зауважити, що ця техніка є досить ресурсозатратною, тому її треба використовувати з обережністю.

Post-processing працює таким чином, що вбудовується у Render Pipeline, саме тому Post-processing впливає на все зображення в цілому.

Для того, щоб досягти ефекту вицвітання було використане налаштування постпроцессінгу Volume з параметрами Color Curves.

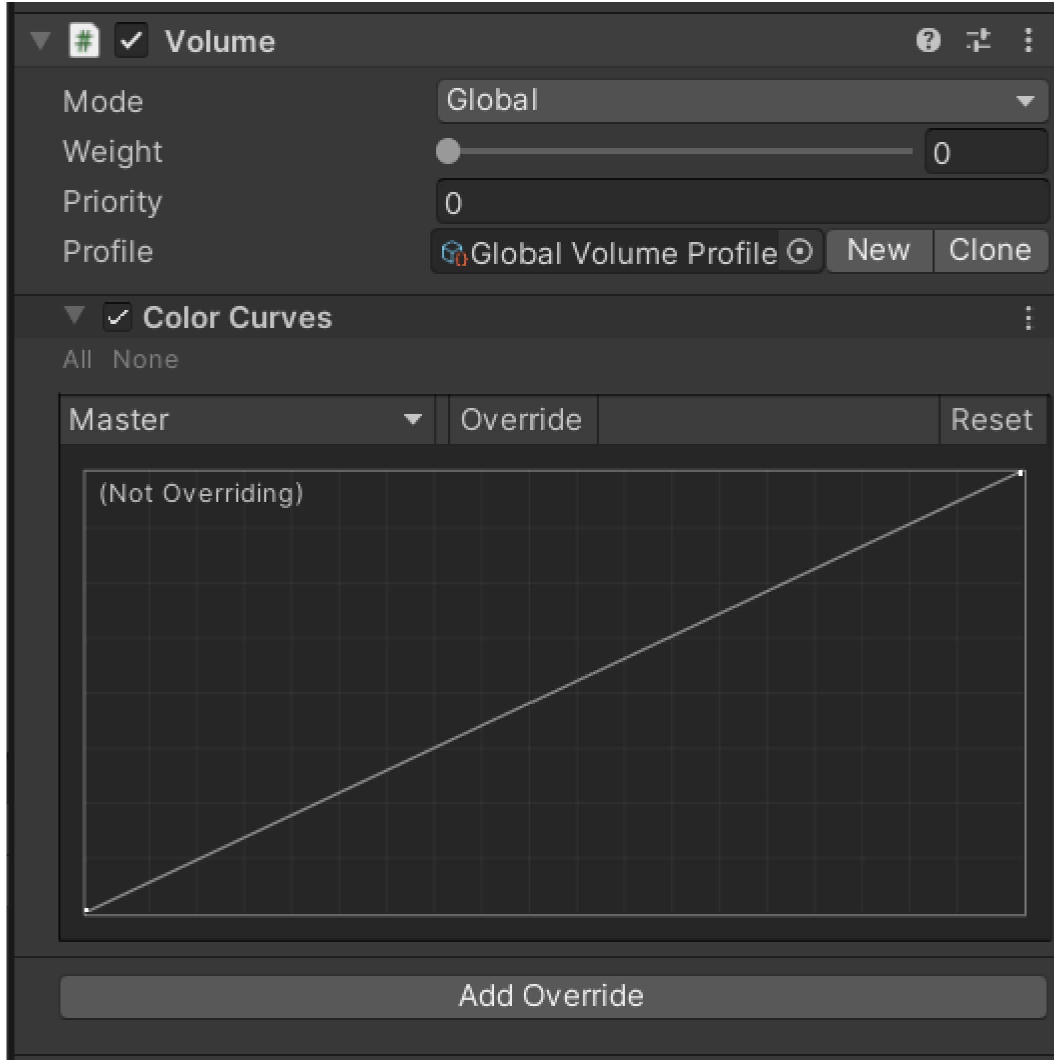

Рисунок 3.16 - Компонент Volume

Таким чином, змінюючи параметр Weight від 0 до 1 ми можемо керувати інтенсивністю вицвітання кольорів, де 0 - максимальна яскравість та 1 - максимальне вицвітання.

3.7 Принципи побудови механіки руйнування будівлі

Універсальний інструмент для руйнування будь-якої 3D моделі є однією з найскладніших частин гри. У ході розробки цієї комплексної системи було розроблено такі інструменти:

- Інструмент об’єднання дочірніх мешів у один великий меш;
- Інструмент процедурного розрізання меша на мілкі дочірні підмеши;
- Система важких тіл - власне рішення для фізичного тіла для того, щоб досягти більш реалістичного руху зруйнованих частин будівлі. Ця система дає можливість обрати тип матеріалу, в залежності від якого буде змінюватися швидкість падіння, вага, тертя і тд;
- Система фізичних бомб - дозволяє створювати фізично реалістичні бомби та за допомогою них руйнувати будівлю на менші уламки;
- Система цілісності уламків - система, яка надає кожному об’єкту параметр цілісності, якщо цей параметр падає до нуля, то об’єкт розпадається на менші уламки, які також отримують цей параметр.

3.7.1 Принципи побудови інструменту об’єднання дочірніх мешів у один меш

Полігональна сітка — це набір вершин, ребер і граней, які використовуються для опису форми гранених об’єктів у 3D-графіці та твердотільному моделюванні. Грані зазвичай складаються з трикутників (трикутних сіток), чотирикутників або інших опуклих багатокутників, що спрощує їх відтворення, хоча можна використовувати більш загальні неопуклі багатокутники або багатокутники з отворами.

Матеріал - поняття, яке існує лише в середовищі рушія Unity. Це оболонка для шейдеру ShaderLab, що дозволяє налаштовувати параметри шейдера прямо в інспекторі.

Алгоритм об’єднання кількох полігональних сіток працює так:

1. Алгоритм проходить по всім об’єктам з однаковим матеріалом та групує їх в окрему групу.
2. Проходить по кожній групі та на основі даних про положення вершин та ребер створює новий єдиний меш. Якщо вершини лежать в одному положенні в просторі, алгоритм залишає лише одну точку з двох, а іншу видаляє.

Таким чином після групування ми отримуємо декілька груп, що сегментовані за однаковим матеріалом, що вже само по собі дає приріст в швидкості обробки зображення, оскільки замість того, щоб обробляти кожен об’єкт окремо, Render Pipeline тепер вимальовує їх як одну єдину модель.

Також це дає приріст в тому, що ми змогли скоротити кількість вершин об’єкта. Тому GPU тепер треба менше часу, щоб відобразити не екрані зобрадження.

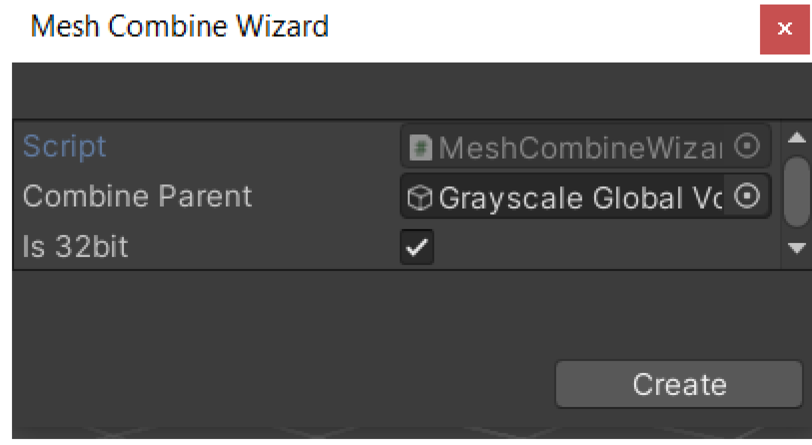

Рисунок 3.17 - Вікно інструмента для комбінування полігональних сіток

Таким чином виглядає вікно комбінування декількох полігональних сіток. У першому параметрі можна вказати кореневий об’єкт, з якого будуть діставатися дочірні полігональні сітки для об’єднання.

У другому параметрі вказуємо у якому форматі хочемо зберігати полігональну сітку. Він вказує на те, скільки буферу ми хочемо виділити для зберігання 3D моделі. 16-бітний формат дозволяє зберігати модель з максимальною кількістю 65535 вершин. 32-бітний формат дозволяє зберігати модель до 4 мільйонів вершин.

3.7.2 Принципи побудови системи інструменту процедурного розрізання полігональної сітки

Алгоритм процедурного розрізання полігональної сітки працює подібно до алгоритму об’єднання кількох полігональних сіток, проте операції виконуються зворотньо. Замість того, щоб шукати спільні точки та об’єднувати їх ми обираємо точки, в яких хочемо розрізати полігональну сітку, та в цьому місці будуємо площину, а саму полігональну сітку ділимо на дві незалежних сітки.

Порядок роботи алгоритму:

1. Обрати точки на полігональній сітці, у якій буде проведений розріз;
2. Обрати сполучення точок для того, щоб обрати незалежні полігональні сітки;
3. На місцях сполучення точок побудувати площини, та зберегти нові полігональні сітки як нові файли.

Для того, щоб обрати точки на полігональний сітці та провести лінії сполучення був використана діаграма Вороного.

Діаграма Вороного — це спеціальний поділ метричного простору, що визначається відстанню до даної дискретної точки в цьому просторі.

У найпростішому випадку ми маємо набір точок у площині S, які називаються вершинами діаграми Вороного. Кожна вершина s має клітинку Вороного, також звану клітинкою Діріхле, V(s), яка утворена всіма точками, ближчими до s, ніж будь-яка інша вершина. Межами в графі Вороного є всі точки на площині, рівновіддалені від двох найближчих вершин. Вузли Вороного — точки, рівновіддалені від трьох і більше вершин.

Алгоритм заснований на використанні лінійних розгорток. Лінії розгортки є допоміжними об’єктами для вертикальних ліній. На кожному кроці алгоритму діаграма Вороного формується у вигляді набору згорнутих ліній, що вказують ліворуч від неї. При цьому межа між областю Вороного прямої і областю точки складається з параболічних відрізків (оскільки геометричне положення точки,

рівновіддаленої від даної точки і прямої, є параболою). Лінія рухається зліва направо. Кожного разу, коли проходить іншу точку, ця точка додається до побудованої частини діаграми. Складність додавання точки до графіка за допомогою двійкового дерева пошуку дорівнює O(log n), загальна кількість точок дорівнює n, а сортування точок за координатою x може досягти O(nlog n), тому обчислювальна складність алгоритм Fortune - O (nlog n).

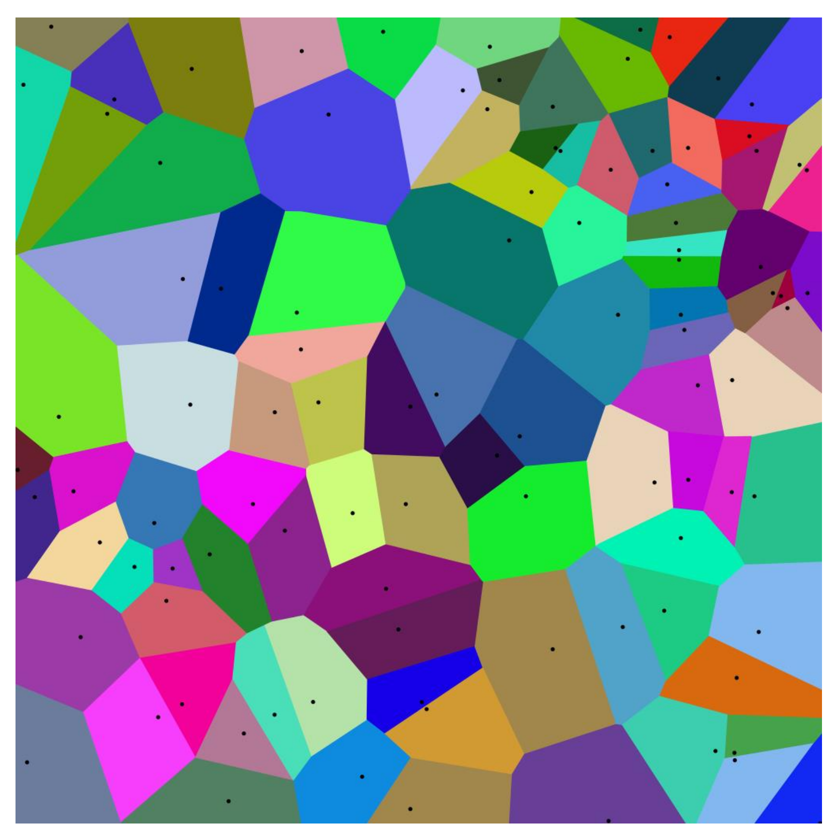

Рисунок 3.18 - Приклад діаграми Вороного

Таким чином виглядає діаграма Вороного для випадкової множини точок. Усі точки лежать всередині зображення.

За допомогою цього алгоритму ми можемо створити реалістичну симуляцію руйнування будівлі, оскільки відомо, що дуже багато природних явищ мають математичну будову.

Рисунок 3.19 - 3D модель стіни до розрізання

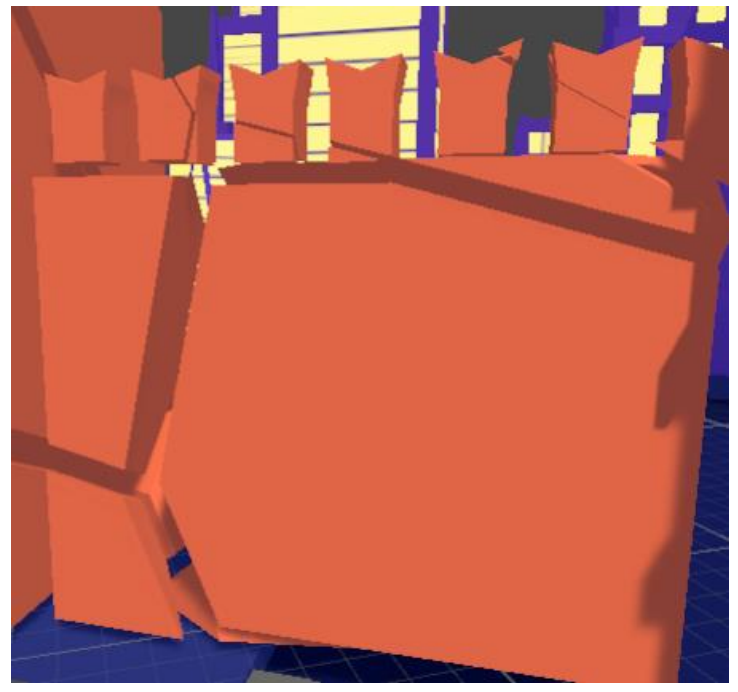

Рисунок 3.20 - 3D модель стіни після розрізання

Як видно, результат є досить реалістичний та схожий на те, як руйнується будівля. Його можна ще покращити, але це вже буде коштувати додаткових ресурсів пристрою. Оскільки цільові платформи - Android та iOS, було прийнято рішення зупинитися на такому результаті.

3.7.3 Принципи побудови системи цілісності уламків

Система цілісності уламків дуже схожа на систему здоров'я персонажу. Кожен об'єкт також має максимальне значення цілісності, воно може змінюватися через зовнішні умови. Відмінність в тому, що при досягненні нульового значення, уламок розпадається на декілька уламків з таким самим компонентом цілісності.

В результаті це додає реалізму до системи руйнування будівлі, оскільки, логічно, що матеріал повинен розпадатися у декілька етапів: від вибуху, від зіткнення з іншими уламками та від удару о землю під дією сили тяжіння.

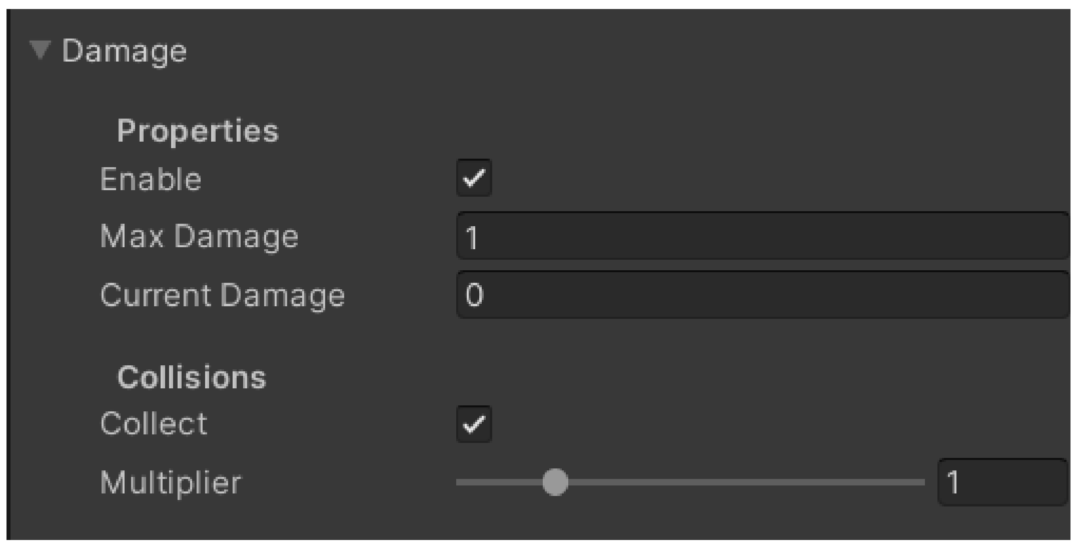

Рисунок 3.21 - Вікно системи цілісності уламків

У результаті повної роботи всіх систем руйнування, будівля виглядає таким чином:

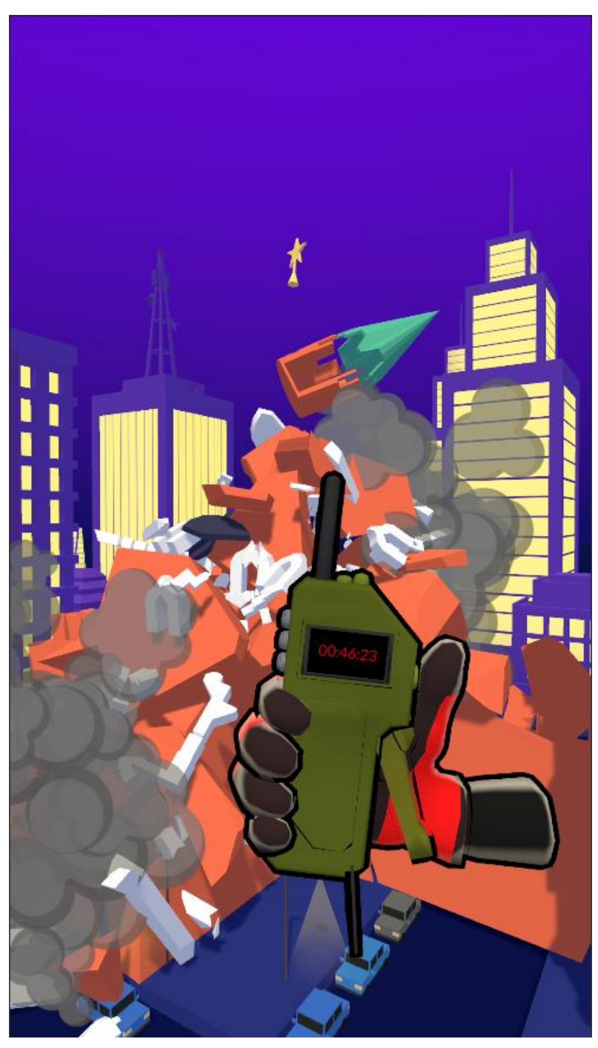

Рисунок 3.22 - Будівля до та після руйнування

3.8 Результати тестування мобільної гри

Для тестування було використано мобільний телефон з операційною системою Android - Google Pixel 2XL з версією Android 11 та мобільний телефон з операційною системою iOS - iPhone XR з версією iOS 14.5. Було проведено декілька тестувань, якими перевірялися різні аспекти гри.

Перше тестування націлене на перевірку кросплатформенних властивостей проектованої гри та коректний рендеринг об'єктів та кольорів у зв'язку з різними системами рендерингу для різних операційних систем.

Дане тестування довело, що гра дійсно є кросплатформним додатком та усі елементи гри рендеряться однаково на різних операційних системах.

Друге тестування було спрямовано на визначення кількості кадрів на секунду (FPS – Frames Per Second), при якому працює гра на обох платформах.

Дане тестування показало такі результати:

- Android. Гра працює при стабільних 60 кадрах на секунду та при часу обробки кадру на процесорі 15-16 мс;
- iOS. Гра працює при стабільних 60 кадрах на секунду та при часу обробки кадру на процесорі 15-16 мс.

Результати даного тестування свідчать про те, що гра є досить оптимізованою та готова до випуску у магазин після підключення сервісу аналітики.

Таким чином, в результаті виконання третього розділу кваліфікаційної роботи було проведено детальне проектування, планування, реалізація та тестування програмної системи. Результатом є мобільна гіпер казуальна гра у жанрі екшн.

3.9 Висновок

У третій частині були розглянуті ключові моменти розробки мобільної гри у жанрі екшн. Спочатку були сформовані функціональні проектованого додатку та розроблені наступні системи:

- Інструмент для організації порядку рівнів;
- Система обмеженого зору супротивника;
- Система розрізання полігональної сітки за діаграмою Вороного;
- Система об'єднання кількох полігональних сіток у одну;
- Система важких тіл;
- Система цілісності уламків;
- Система побудови пересування супротивників по ігровому рівню.

Розроблена гра проходить усі тести продуктивності та навантаженості на пристрій. Отже, додаток готовий до публікації в інтернет-магазині.

ВИСНОВОК

У процесі дипломної роботи було спроектовано та створено мобільний застосунок комп’ютерної гри “Plant The Bomb”.

Під час написання було проведено декілька досліджень, якими було визначено найкраща тема та жанр розробленого проекту, влучні технологічні рішення для створення необхідного функціоналу з задовільною складністю розробки та підтримки системи. Для дослідження теоретичних засад було взято існуючі ігри та документації про роботу з фізикою та графікою.

Протягом розробки даного додатку були створені наступні інструменти:

- Система обмеженого зору супротивника;
- Система розрізання полігональної сітки за діаграмою Вороного;
- Система об’єднання кількох полігональних сіток у одну;
- Система важких тіл;
- Система цілісності уламків;
- Система побудови пересування супротивників по ігровому рівню.

Програмна система реалізована на Unity, C# та HLSL. Для проектування логіки програми було використано програмне забезпечення компанії JetBrains Rider.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Документація Microsoft .NET [Електронний ресурс] – Режим доступу: https://docs.microsoft.com/en-us/dotnet/ (переглянуто 13 березня 2022 року);
2. Джеффрі Ріхтер. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#.
3. Ray-tracing / Пересечение луча и треугольника [Електронний ресурс] – Режим доступу: http://ray-tracing.ru/articles213.html (переглянуто 26 березня 2022 року)
4. 3D Collision Detection [Електронний ресурс] - Режим доступу: https://developer.mozilla.org/en-US/docs/Games/Techniques/3D_collision_detection (переглянуто 22 березня 2022 року)
5. Pattern Command [Електронний ресурс] - Режим доступу: https://refactoring.guru/ru/design-patterns/command/csharp/example (переглянуто 2 квітня 2022 року)
6. Sensortower [Електронний ресурс] - Режим доступу: https://sensortower.com/ (переглянуто 14 березня 2022 року)
7. NewZoo 2021 Global Games Market Review [Електронний ресурс] - Режим доступу: https://newzoo.com/insights/trend-reports/newzoo-global-games-market-report-2021-free-version/
8. Unity Learn [Електронний ресурс] – Режим доступу : https://learn.unity.com/ (переглянуто 14 березня 2022 року)
9. 100 поєднань кольорів, які приваблюють потрібну аудиторію. [Електронний ресурс] – Режим доступу до ресурсу: https://www.canva.com/ru_ru/obuchenie/100-cvetovyx-sochetnij/ (переглянуто 15 березня 2022 року)
10. Unity Documentation [Електронний ресурс] – Режим доступу: https://docs.unity3d.com/ScriptReference/Gizmos.html (переглянуто 16 березня 2022 року)

11. Unity Shader Graph Documentation [Електронний ресурс] – Режим доступу: https://docs.unity3d.com/Packages/com.unity.shadergraph@6.9/manual/Triplanar-Node.html (переглянуто 16 березня 2022 року)

12. Unity ShaderLab Documentation [Електронний ресурс] - Режим доступу: https://docs.unity3d.com/2019.4/Documentation/Manual/SL-Material.html (переглянуто 7 травня 2022 року)

13. Unity Cameras Documentation [Електронний ресурс] - Режим доступу: https://docs.unity3d.com/2019.4/Documentation/Manual/CamerasOverview.html (переглянуто 7 травня 2022 року)

14. Unity Render Pipeline Documentation [Електронний ресурс] - Режим доступу: https://docs.unity3d.com/2019.4/Documentation/Manual/render-pipelines.html (переглянуто 8 травня 2022 року)

15. Unity Lighting Documentation [Електронний ресурс] - Режим доступу: https://docs.unity3d.com/2019.4/Documentation/Manual/LightingOverview.html (переглянуто 8 травня 2022 року)

16. Unity Models Documentation [Електронний ресурс] - Режим доступу: https://docs.unity3d.com/2019.4/Documentation/Manual/models.html (переглянуто 8 травня 2022 року)

17. Unity Meshes Documentation [Електронний ресурс] - Режим доступу: https://docs.unity3d.com/2019.4/Documentation/Manual/mesh.html (переглянуто 8 травня 2022 року)

18. Unity Textures Documentation [Електронний ресурс] - Режим доступу: https://docs.unity3d.com/2019.4/Documentation/Manual/Textures.html (переглянуто 8 травня 2022 року)

19. Unity Materials Documentation [Електронний ресурс] - Режим доступу: https://docs.unity3d.com/2019.4/Documentation/Manual/Materials.html (переглянуто 8 травня 2022 року)

20.Unity Visual Effects Documentation [Електронний ресурс] - Режим доступу: https://docs.unity3d.com/2019.4/Documentation/Manual/visual-effects.html (переглянуто 8 травня 2022 року)

21. Unity C# [Електронний ресурс]. // itProger. - Режим доступу: https://itproger.com/course/unity-csharp

22. Сім етапів створення гри: від концепту до релізу [Електронний ресурс]. // habr. - Режим доступу: https: //habr.com/company/miip/blog/308286

23. Джессі Шелл. Геймдізайн. Як створити гру, в яку гратимуть усі. Альпіна паблішер, 2019. 640 с.

24. Joris Dormans. Game Mechanics: Advanced Game Design. Addison-Wesley Professional, 2008. 464 с.

25. Robert Martin. Clean Code. Pearson, 2008, 464 с.

26. Steve Swink. Game Feel. CRC Press, 2008, 376 с.

27. Raph Koster. Theory of Fun. O'Reilly Media, 2013. 300 с.

28. Robert Nystrom. Game Programming Patterns, 2014. 354 с.

29. Jesse Schell. The Art of Game Design. A K Peters/CRC Press, 2014. 600 с.

30. Tracy Fullerton. Game Design Workshop. A K Peters/CRC Press, 2014. 535 с

ДОДАТКИ

Додаток А

Source Code проекту

```
namespace PlantTheBomb
{
  public enum EGameState
  {
     Menu,
     Game,
     Success,
     Fail,
     BuildingDemolition
  }

  public enum ECharacters
  {
     PlayerCharacter,
  }

  public class Constants
  {

  }
}using System;
using System.Threading;
using System.Threading.Tasks;
using SuperTools.CameraManager;
using SuperTools.Chronos;
using SuperTools.Menus;
using PlantTheBomb.Cameras;
using PlantTheBomb.Characters.Player;
using PlantTheBomb.Commands;
using PlantTheBomb.GameStates;
using PlantTheBomb.LevelSystem;
using UnityEngine;

namespace PlantTheBomb
{
   public class Dispatcher
   {
       public Dispatcher()
       {
           instance = this;
           InitEvents();
       }

       private static Dispatcher instance;
```

```
public static Dispatcher Instance
{
    get
    {
        if (instance == null)
        {
            instance = new Dispatcher();
        }

        return instance;
    }
}

public EGameState CurrentState { get; private set; }

public event Action loadLevel;
public event Action startPlayLevel;
public event Action successLevel;
public event Action failLevel;
public event Action endLevel;

private bool _isEnd;

private CancellationTokenSource _successWindowCts;

private void InitEvents()
{
    loadLevel += new InitLevelCommand().Execute;
    loadLevel += OnLoadLevel;
    startPlayLevel += OnStartPlayLevel;
    successLevel += OnSuccessLevel;
    failLevel += OnFailLevel;
    endLevel += OnEndLevel;
}

~Dispatcher()
{
    loadLevel -= new InitLevelCommand().Execute;
    loadLevel -= OnLoadLevel;
    startPlayLevel -= OnStartPlayLevel;
    successLevel -= OnSuccessLevel;
    failLevel -= OnFailLevel;
    endLevel -= OnEndLevel;
}

private async void OnLoadLevel()
{
    Chronos.TimeScale = 1f;

    CurrentState = EGameState.Menu;
```

```
            MenuManager.Instance.HideAllMenusImmediately();

            await Task.Yield();

            MenuManager.Instance.MenuList.MainMenu.Show();
        }

        private void OnStartPlayLevel()
        {
            MenuManager.Instance.HideAllMenusImmediately();

            MenuManager.Instance.MenuList.GameMenu.Show();
        }

        private void OnSuccessLevel()
        {
            _successWindowCts = new CancellationTokenSource();

MenuManager.Instance.MenuList.SuccessMenu.ShowWithDelay(_successWindowCts
.Token);

            LevelsLoader.Instance.IncreaseLevel();
        }

        private void OnFailLevel()
        {
            var failCamera =
CameraManager.Instance.GetManagedCamera("FailCamera") as FailCamera;
            var player =
GameObject.FindObjectOfType<PlayerCharacterController>();

            failCamera.CinemachineVirtualCameraBase.Follow =
player.Model.UnitModel.BodyCenter.transform;
            failCamera.CinemachineVirtualCameraBase.LookAt =
player.Model.UnitModel.BodyCenter.transform;

            Chronos.TimeScale = .3f;

            failCamera.Transition();
            failCamera.PlayGrayscale();

            MenuManager.Instance.MenuList.FailMenu.ShowWithDelay();
        }

        private void OnEndLevel()
        {

        }
```

```
public void LoadLevel()
{
    loadLevel?.Invoke();
}

public async void StartPlayLevel()
{
    await Task.Yield(); // Wait a frame to hide MainMenu

    CurrentState = EGameState.Game;

    startPlayLevel?.Invoke();
}

public void StartBuildingDemolitionState()
{
    CurrentState = EGameState.BuildingDemolition;

    new BuildingDemolitionState().PerformTransition();
}

public void SuccessLevel()
{
    if (_isEnd) return;

    _isEnd = true;

    CurrentState = EGameState.Success;

    successLevel?.Invoke();

    EndLevel();
}

public void FailLevel()
{
    if (_isEnd) return;

    _isEnd = true;

    CurrentState = EGameState.Fail;

    failLevel?.Invoke();

    EndLevel();
}

public void EndLevel()
{
    endLevel?.Invoke();
```

```
        }
    }
}using UnityEngine;

namespace PlantTheBomb
{
  [DefaultExecutionOrder(-10001)]
  public class Init : MonoBehaviour
  {
     private void Awake()
     {
        new Dispatcher();
     }

     private void Start()
     {
        Application.targetFrameRate = 90;

        Dispatcher.Instance.LoadLevel();
     }
  }
}namespace PlantTheBomb
{
  public enum EGameState
  {
     Menu,
     Game,
     Success,
     Fail,
     BuildingDemolition
  }

  public enum ECharacters
  {
     PlayerCharacter,
  }

  public class Constants
  {

  }
}using System;
using System.Threading;
using System.Threading.Tasks;
using SuperTools.CameraManager;
using SuperTools.Chronos;
using SuperTools.Menus;
using PlantTheBomb.Cameras;
using PlantTheBomb.Characters.Player;
using PlantTheBomb.Commands;
```

```
using PlantTheBomb.GameStates;
using PlantTheBomb.LevelSystem;
using UnityEngine;

namespace PlantTheBomb
{
   public class Dispatcher
   {
       public Dispatcher()
       {
           instance = this;
           InitEvents();
       }

       private static Dispatcher instance;
       public static Dispatcher Instance
       {
           get
           {
               if (instance == null)
               {
                   instance = new Dispatcher();
               }

               return instance;
           }
       }

       public EGameState CurrentState { get; private set; }

       public event Action loadLevel;
       public event Action startPlayLevel;
       public event Action successLevel;
       public event Action failLevel;
       public event Action endLevel;

       private bool _isEnd;

       private CancellationTokenSource _successWindowCts;

       private void InitEvents()
       {
           loadLevel += new InitLevelCommand().Execute;
           loadLevel += OnLoadLevel;
           startPlayLevel += OnStartPlayLevel;
           successLevel += OnSuccessLevel;
           failLevel += OnFailLevel;
           endLevel += OnEndLevel;
       }
```

```
        ~Dispatcher()
        {
            loadLevel -= new InitLevelCommand().Execute;
            loadLevel -= OnLoadLevel;
            startPlayLevel -= OnStartPlayLevel;
            successLevel -= OnSuccessLevel;
            failLevel -= OnFailLevel;
            endLevel -= OnEndLevel;
        }

        private async void OnLoadLevel()
        {
            Chronos.TimeScale = 1f;

            CurrentState = EGameState.Menu;

            MenuManager.Instance.HideAllMenusImmediately();

            await Task.Yield();

            MenuManager.Instance.MenuList.MainMenu.Show();
        }

        private void OnStartPlayLevel()
        {
            MenuManager.Instance.HideAllMenusImmediately();

            MenuManager.Instance.MenuList.GameMenu.Show();
        }

        private void OnSuccessLevel()
        {
            _successWindowCts = new CancellationTokenSource();

MenuManager.Instance.MenuList.SuccessMenu.ShowWithDelay(_successWindowCts
.Token);

            LevelsLoader.Instance.IncreaseLevel();
        }

        private void OnFailLevel()
        {
            var failCamera =
CameraManager.Instance.GetManagedCamera("FailCamera") as FailCamera;
            var player =
GameObject.FindObjectOfType<PlayerCharacterController>();

            failCamera.CinemachineVirtualCameraBase.Follow =
player.Model.UnitModel.BodyCenter.transform;
```

```
            failCamera.CinemachineVirtualCameraBase.LookAt =
player.Model.UnitModel.BodyCenter.transform;

            Chronos.TimeScale = .3f;

            failCamera.Transition();
            failCamera.PlayGrayscale();

            MenuManager.Instance.MenuList.FailMenu.ShowWithDelay();
        }

        private void OnEndLevel()
        {

        }

        public void LoadLevel()
        {
            loadLevel?.Invoke();
        }

        public async void StartPlayLevel()
        {
            await Task.Yield(); // Wait a frame to hide MainMenu

            CurrentState = EGameState.Game;

            startPlayLevel?.Invoke();
        }

        public void StartBuildingDemolitionState()
        {
            CurrentState = EGameState.BuildingDemolition;

            new BuildingDemolitionState().PerformTransition();
        }

        public void SuccessLevel()
        {
            if (_isEnd) return;

            _isEnd = true;

            CurrentState = EGameState.Success;

            successLevel?.Invoke();

            EndLevel();
        }
```

```
        public void FailLevel()
        {
            if (_isEnd) return;

            _isEnd = true;

            CurrentState = EGameState.Fail;

            failLevel?.Invoke();

            EndLevel();
        }

        public void EndLevel()
        {
            endLevel?.Invoke();
        }
    }
}using UnityEngine;

namespace PlantTheBomb
{
  [DefaultExecutionOrder(-10001)]
  public class Init : MonoBehaviour
  {
     private void Awake()
     {
        new Dispatcher();
     }

     private void Start()
     {
        Application.targetFrameRate = 90;

        Dispatcher.Instance.LoadLevel();
     }
  }
}namespace PlantTheBomb
{
  public enum Eusing UnityEngine;

namespace PlantTheBomb.Analytics
{
  public class LevelAnalytics : MonoBehaviour
  {
  }
}using DG.Tweening;
using SuperTools.CameraManager;
using UnityEngine;
using UnityEngine.Rendering;
```

```
namespace PlantTheBomb.Cameras
{
  public class FailCamera : ManagedCamera
  {
     [SerializeField] private Volume _grayscaleVolume;

     public void PlayGrayscale()
     {
        DOTween.To(() => _grayscaleVolume.weight, x =>
_grayscaleVolume.weight = x,
                   1, 1);
     }
  }
}using SuperTools.CameraManager;
using UnityEngine;

namespace PlantTheBomb.Cameras
{
  public class FPSCamera : ManagedCamera
  {
     [SerializeField] private Shake _explosionShake;

     public void ShakeExplosion()
     {
        _explosionShake.Play();
     }
  }
}using CoolTools.StateMachine;
using PlantTheBomb.Characters.States;
using PlantTheBomb.Characters.Units.Abstractions;
using PlantTheBomb.Damage.Health;
using UnityEngine;

namespace PlantTheBomb.Characters.Abstractions
{
  public abstract class BaseCharacterController : MonoBehaviour
  {
     public BaseCharacterModel Model { get; private set; }
     protected BaseCharacterView View { get; set; }

     protected BaseUnitController UnitController { get; set; }
     protected HealthComponent HealthComponent { get; set; }

     protected StateMachine StateMachine { get; set; }
     protected IdleState IdleState { get; set; }
     protected MoveState MoveState { get; set; }
     protected DeadState DeadState { get; set; }

     public void Init(BaseCharacterModel model, BaseCharacterView view)
```

```
      {
         Model = model;
         View = view;

         InitComponents();

         InitStateMachine();
      }

      protected virtual void InitComponents()
      {
         InitUnit();

         HealthComponent = Model.HealthComponent;
         HealthComponent.takeDamage += View.TakeDamage;
         HealthComponent.healthChanged += DisplayHealth;
         DisplayHealth();
      }

      private void InitUnit()
      {
         UnitController =
Model.UnitModel.gameObject.GetComponent<BaseUnitController>();
         UnitController.Init(Model.UnitModel, View.UnitView);
      }

      private void DisplayHealth()
      {
         View.DisplayHealth(HealthComponent.CurrentHealth,
HealthComponent.MaxHealth);
      }

      protected virtual void InitStateMachine()
      {
         StateMachine = Model.StateMachine;

         InitStates();

         InitStateMachineTransitions();

         StateMachine.SetState(IdleState);
      }

      protected virtual void InitStates()
      {
         // TODO: Move init states to another place.
         // Init states shouldn't place in a controller, but they need a
view.
         // The view shouldn't in the model. So InitStates() shouldn't
place in either the model or the controller.
```

```
            // So we have to create "place".

            IdleState = Model.IdleState;
            IdleState.Init(Model.UnitModel.Rb, View);

            MoveState = Model.MoveState;
            MoveState.Init(Model.UnitModel.Movement, Model.Input, View);

            DeadState = Model.DeadState;
            DeadState.Init(Model.UnitModel.Movement, View);
        }

        protected virtual void InitStateMachineTransitions()
        {
            StateMachine.AddTransition(IdleState, DeadState, () =>
HealthComponent.IsDead);
            StateMachine.AddTransition(MoveState, DeadState, () =>
HealthComponent.IsDead);
        }

        private void OnDestroy()
        {
            HealthComponent.takeDamage -= View.TakeDamage;
            HealthComponent.healthChanged -= DisplayHealth;
        }

        private void Update()
        {
            StateMachine.Tick();
        }

        private void FixedUpdate()
        {
            StateMachine.FixedTick();
        }
    }
}using PlantTheBomb.Characters.Units.Abstractions;
using PlantTheBomb.Damage.Health;
using PlantTheBomb.Input;
using UnityEngine;

namespace PlantTheBomb.Characters.Abstractions
{
    public abstract class BaseCharacterInitializer<TController> :
MonoBehaviour
        where TController : BaseCharacterController
    {
        private void Start()
        {
            Init();
```

```
    }

    private void Init()
    {
      var model = CreateModel(gameObject);
      var view = CreateView(gameObject);

      var controller = gameObject.GetComponent<TController>();
      controller.Init(model, view);
    }

    private BaseCharacterModel CreateModel(GameObject characterGO)
    {
      var input = characterGO.GetComponentInChildren<IBaseInput>();
      input.Init();

      var unitModel =
characterGO.GetComponentInChildren<BaseUnitModel>();
      unitModel.Init();

      var health =
characterGO.GetComponentInChildren<HealthComponent>();

      var characterModel =
characterGO.GetComponent<BaseCharacterModel>();
      characterModel.Init(input, unitModel, health);

      return characterModel;
    }

    private BaseCharacterView CreateView(GameObject characterGO)
    {
      var characterView =
characterGO.GetComponent<BaseCharacterView>();

      return characterView;
    }
  }
}using CoolTools.StateMachine;
using PlantTheBomb.Characters.States;
using PlantTheBomb.Characters.Units.Abstractions;
using PlantTheBomb.Damage;
using PlantTheBomb.Damage.Health;
using PlantTheBomb.Input;
using UnityEngine;

namespace PlantTheBomb.Characters.Abstractions
{
  public abstract class BaseCharacterModel : MonoBehaviour,
IDamageReceiver
```

```
    {
        public IBaseInput Input { get; private set; }
        public BaseUnitModel UnitModel { get; private set; }
        public HealthComponent HealthComponent { get; private set; }

        public StateMachine StateMachine { get; private set; }
        public IdleState IdleState { get; private set; }
        public MoveState MoveState { get; private set; }
        public DeadState DeadState { get; private set; }

        public virtual void Init(IBaseInput input, BaseUnitModel unitModel,
HealthComponent healthComponent)
        {
            Input = input;
            UnitModel = unitModel;

            HealthComponent = healthComponent;

            InitComponents();

            InitStateMachine();
        }

        protected virtual void InitComponents()
        {
        }

        protected virtual void InitStateMachine()
        {
            StateMachine = new StateMachine();

            IdleState = new IdleState();
            MoveState = new MoveState();
            DeadState = new DeadState();
        }

        public void ReceiveDamage(float damage, Vector3 direction)
        {
            HealthComponent.TryTakeDamage(damage);

            if (HealthComponent.IsDead)
                DeadState.SetHitDirection(direction);
        }
    }
}using PlantTheBomb.Characters.Units.Abstractions;
using UnityEngine;

namespace PlantTheBomb.Characters.Abstractions
{
  public abstract class BaseCharacterView : MonoBehaviour
```

```
  {
     public BaseUnitView UnitView { get; private set; }

     private void Awake()
     {
        UnitView = GetComponentInChildren<BaseUnitView>();
     }

     public virtual void TakeDamage(float value)
     {
        UnitView.TakeDamage();
     }

     public virtual void DisplayHealth(float health, float maxHealth){}

     public void SetIdle()
     {
        UnitView.SetIdle();
     }

     public void SetWalk(float moveSpeed, float maxMoveSpeed)
     {
        UnitView.SetMove(moveSpeed, maxMoveSpeed);
     }

     public void SetAttackActive(bool value)
     {
        UnitView.SetAttackActive(value);
     }

     public virtual void SetDeath(Vector3 hitDirection)
     {
        UnitView.SetDeath(hitDirection);
     }
  }
}using PlantTheBomb.Characters.Abstractions;
using PlantTheBomb.Damage.Health;
using PlantTheBomb.FieldOfViewSystem;
using UnityEngine;

namespace PlantTheBomb.Characters.AI
{
  public abstract class AICharacterController : BaseCharacterController
  {
     public new AICharacterModel Model => base.Model as AICharacterModel;
     public new AICharacterView View => base.View as AICharacterView;

     protected FieldOfView AggressionFieldOfView { get; set; }
     private float _lastTimeTargetWasVisible = -1;
```

```
    protected WorriesState WorriesState { get; set; }
    protected AggressionState AggressionState { get; set; }

    protected override void InitComponents()
    {
       base.InitComponents();

       AggressionFieldOfView = Model.AggressionFieldOfView;
    }

    protected override void InitStates()
    {
       base.InitStates();

       WorriesState = Model.WorriesState;

       AggressionState = Model.AggressionState;
       AggressionState.Init(Model.TargetFocuser, AggressionFieldOfView,
Model.Weapon, View);
    }

    protected override void InitStateMachineTransitions()
    {
       base.InitStateMachineTransitions();

       StateMachine.AddTransition(WorriesState, AggressionState, () =>
IsTargetInFieldOfView(AggressionFieldOfView) &&
IsTimeUntilAggressionPassed());

       StateMachine.AddTransition(AggressionState, IdleState, () =>
!IsTargetInFieldOfView(AggressionFieldOfView));
    }

    protected bool IsTargetInFieldOfView(FieldOfView field)
    {
       if (_lastTimeTargetWasVisible > 0 && Time.time -
_lastTimeTargetWasVisible < Model.ExitAggressionDelay)
          return true;

       if (field.visibleTargets.Count > 0)
       {
          var targetHealth =
field.visibleTargets[0].GetComponentInParent<HealthComponent>();

          if (targetHealth && !targetHealth.IsDead)
          {
             _lastTimeTargetWasVisible = Time.time;
             return true;
          }
       }
```

```
            return false;
        }

        private bool IsTimeUntilAggressionPassed() =>
            Time.time - WorriesState.EnterStateTime >=
Model.GetTimeUntilAggression();
    }
}using PlantTheBomb.Characters.Abstractions;

namespace PlantTheBomb.Characters.AI
{
    public class AICharacterInitializer :
BaseCharacterInitializer<AICharacterController>
    {
    }
}using PlantTheBomb.Characters.Abstractions;
using PlantTheBomb.Damage;
using PlantTheBomb.FieldOfViewSystem;
using PlantTheBomb.Movement.Focus;
using UnityEngine;

namespace PlantTheBomb.Characters.AI
{
    public abstract class AICharacterModel : BaseCharacterModel
    {
        [field: SerializeField] public float ExitAggressionDelay { get;
private set; }
        [SerializeField] protected float _aggressionDelayPerDistance;

        [field: SerializeField] public FieldOfView AggressionFieldOfView {
get; private set; }
        public TargetFocuser TargetFocuser { get; private set; }
        public Weapon Weapon { get; private set; }

        public WorriesState WorriesState { get; private set; }
        public AggressionState AggressionState { get; private set; }

        protected override void InitComponents()
        {
            base.InitComponents();

            TargetFocuser = new TargetFocuser(transform);

            Weapon = GetComponentInChildren<Weapon>();
            UnitModel.Equipment.Equip(Weapon);
        }

        protected override void InitStateMachine()
        {
```

```
        base.InitStateMachine();

        WorriesState = new WorriesState();
        AggressionState = new AggressionState();
     }

     public virtual float GetTimeUntilAggression() =>
_aggressionDelayPerDistance;
  }
}using PlantTheBomb.Characters.Abstractions;

namespace PlantTheBomb.Characters.AI
{
  public abstract class AICharacterView : BaseCharacterView
  {

  }
}using PlantTheBomb.FieldOfViewSystem;

namespace PlantTheBomb.Characters.AI.Patrolling
{
  public class AIPatrollingCharacterController : AICharacterController
  {
     public new AIPatrollingCharacterModel Model => base.Model as
AIPatrollingCharacterModel;
     public new AIPatrollingCharacterView View => base.View as
AIPatrollingCharacterView;

     protected FieldOfView PatrollingFieldOfView { get; set; }

     protected PatrollingState PatrollingState { get; set; }

     protected override void InitComponents()
     {
        base.InitComponents();

        PatrollingFieldOfView = Model.PatrollingFieldOfView;

        Model.AgentPlus.Init(View);
     }

     protected override void InitStates()
     {
        base.InitStates();

        WorriesState.Init(Model.TargetFocuser, PatrollingFieldOfView,
View);

        PatrollingState = Model.PatrollingState;
```

```
        PatrollingState.Init(Model.AgentPlus, PatrollingFieldOfView,
Model.StartPatrollingDelay);
     }

     protected override void InitStateMachineTransitions()
     {
        base.InitStateMachineTransitions();

        StateMachine.AddTransition(IdleState, WorriesState, () =>
IsTargetInFieldOfView(PatrollingFieldOfView));
        StateMachine.AddTransition(IdleState, PatrollingState, () =>
!IsTargetInFieldOfView(PatrollingFieldOfView));

        StateMachine.AddTransition(WorriesState, IdleState, () =>
!IsTargetInFieldOfView(PatrollingFieldOfView));

        StateMachine.AddTransition(PatrollingState, WorriesState, () =>
IsTargetInFieldOfView(PatrollingFieldOfView));
     }
  }
}using PlantTheBomb.FieldOfViewSystem;
using PlantTheBomb.Movement.Patrolling;
using UnityEngine;

namespace PlantTheBomb.Characters.AI.Patrolling
{
  public class AIPatrollingCharacterModel : AICharacterModel
  {
     [field: SerializeField] public float StartPatrollingDelay { get;
private set; }

     [field: SerializeField] public FieldOfView PatrollingFieldOfView {
get; private set; }
     public NavMeshAgentPlus AgentPlus { get; private set; }

     public PatrollingState PatrollingState { get; private set; }

     protected override void InitComponents()
     {
        base.InitComponents();

        AgentPlus = GetComponent<NavMeshAgentPlus>();
     }

     protected override void InitStateMachine()
     {
        base.InitStateMachine();

        PatrollingState = new PatrollingState();
     }
```

```
        public override float GetTimeUntilAggression()
        {
            if (PatrollingFieldOfView.visibleTargets.Count == 0) return 10;

            var distanceToTarget = Vector3.Distance(transform.position,
PatrollingFieldOfView.visibleTargets[0].position);

            return distanceToTarget * _aggressionDelayPerDistance;
        }
    }
}namespace PlantTheBomb.Characters.AI.Patrolling
{
    public class AIPatrollingCharacterView : AICharacterView
    {

    }
}using CoolTools.StateMachine;
using PlantTheBomb.Damage;
using PlantTheBomb.FieldOfViewSystem;
using PlantTheBomb.Movement.Focus;
using UnityEngine;

namespace PlantTheBomb.Characters.AI
{
    public class AggressionState : IState
    {
        private TargetFocuser _targetFocuser;
        private FieldOfView _fieldOfView;
        private Weapon _weapon;
        private AICharacterView _view;

        private Transform _target;

        public void Init(TargetFocuser targetFocuser, FieldOfView
fieldOfView, Weapon weapon, AICharacterView view)
        {
            _targetFocuser = targetFocuser;
            _fieldOfView = fieldOfView;
            _weapon = weapon;
            _view = view;
        }

        public void OnEnter()
        {
            _fieldOfView.SetDrawActive(true);

            _target = _fieldOfView.visibleTargets[0];
            _targetFocuser.SetTarget(_target);
```

```
        var damageTarget =
_target.GetComponentInParent<IDamageReceiver>();
        _weapon.SetTarget(damageTarget);

        _view.SetAttackActive(true);
     }

     public void Tick()
     {
        _targetFocuser.Focus();
        if (_weapon)
           _weapon.TryAttack();
     }

     public void FixedTick()
     {
     }

     public void OnExit()
     {
        _fieldOfView.SetDrawActive(false);
        _view.SetAttackActive(false);
     }
  }
}using CoolTools.StateMachine;
using PlantTheBomb.FieldOfViewSystem;
using PlantTheBomb.Movement.Patrolling;
using UnityEngine;

namespace PlantTheBomb.Characters.AI
{
  public class PatrollingState : IState
  {
     private NavMeshAgentPlus _navMeshAgentPlus;
     private FieldOfView _fieldOfView;
     private float _startDelay;

     public void Init(NavMeshAgentPlus navMeshAgentPlus, FieldOfView
fieldOfView, float startDelay)
     {
        _navMeshAgentPlus = navMeshAgentPlus;
        _fieldOfView = fieldOfView;
        _startDelay = startDelay;
     }

     public async void OnEnter()
     {
        _fieldOfView.SetDrawActive(true);

        await new WaitForSeconds(_startDelay);
```

```
            _navMeshAgentPlus.SetActive(true);
        }

        public void Tick()
        {
            _navMeshAgentPlus.Tick();
        }

        public void FixedTick()
        {
        }

        public void OnExit()
        {
            _navMeshAgentPlus.SetActive(false);
            _fieldOfView.SetDrawActive(false);
        }
    }
}using CoolTools.StateMachine;
using PlantTheBomb.Characters.AI.Patrolling;
using PlantTheBomb.FieldOfViewSystem;
using PlantTheBomb.Movement.Focus;
using UnityEngine;

namespace PlantTheBomb.Characters.AI
{
    public class WorriesState : IState
    {
        private TargetFocuser _targetFocuser;
        private FieldOfView _fieldOfView;

        private Transform _target;
        private AIPatrollingCharacterView _view;

        public float EnterStateTime { get; private set; }

        public void Init(TargetFocuser targetFocuser, FieldOfView
fieldOfView, AIPatrollingCharacterView view)
        {
            _fieldOfView = fieldOfView;
            _targetFocuser = targetFocuser;
            _view = view;
        }

        public void OnEnter()
        {
            _target = _fieldOfView.visibleTargets[0];
            _targetFocuser.SetTarget(_target);

            _fieldOfView.SetDrawActive(true);
```

```
            _fieldOfView.SetMaterial(_fieldOfView.TriggerMaterial);

            EnterStateTime = Time.time;

            _view.SetAttackActive(true);
        }

        public void Tick()
        {
            _targetFocuser.Focus();
        }

        public void FixedTick()
        {
        }

        public void OnExit()
        {
            _fieldOfView.SetDrawActive(false);
            _fieldOfView.SetMaterial(_fieldOfView.StartMaterial);
            _view.SetAttackActive(false);
        }
    }
}using SuperTools.CameraManager;
using PlantTheBomb.Bombs;
using PlantTheBomb.Characters.Abstractions;
using UnityEngine;

namespace PlantTheBomb.Characters.Player
{
    public class PlayerCharacterController : BaseCharacterController
    {
        public new PlayerCharacterModel Model => base.Model as
PlayerCharacterModel;
        public new PlayerCharacterView View => base.View as
PlayerCharacterView;

        protected BombsSystem BombsSystem { get; set; }

        protected override void InitComponents()
        {
            base.InitComponents();

            Model.UnitModel.Hitbox.OnEnter += OnEnter;
            Model.UnitModel.Hitbox.OnExit += OnExit;

            BombsSystem = Model.BombsSystem;
            BombsSystem.onCreateBomb += Model.EquipCurrentBomb;
            BombsSystem.noMoreBombs += View.PlayNoMoreBombs;
            BombsSystem.TryCreateNextBomb();
```

```
            HealthComponent.die += Dispatcher.Instance.FailLevel;
            HealthComponent.die += BombsSystem.SeparateCurrentBomb;


CameraManager.Instance.CurrentManagedCamera.CinemachineVirtualCameraBase.
Follow = transform;

CameraManager.Instance.CurrentManagedCamera.CinemachineVirtualCameraBase.
LookAt = transform;
        }

        protected override void InitStateMachineTransitions()
        {
            base.InitStateMachineTransitions();

            StateMachine.AddTransition(IdleState, MoveState,
                                            () => Model.Input.IsAvailable &&
Model.Input.InputDirection.magnitude > 0);

            StateMachine.AddTransition(MoveState, IdleState,
                                            () =>
Model.Input.InputDirection.magnitude == 0);
        }

        private void OnDestroy()
        {
            Model.UnitModel.Hitbox.OnEnter -= OnEnter;
            Model.UnitModel.Hitbox.OnExit -= OnExit;

            BombsSystem.onCreateBomb -= Model.EquipCurrentBomb;
            BombsSystem.noMoreBombs -= View.PlayNoMoreBombs;

            HealthComponent.die -= Dispatcher.Instance.FailLevel;
            HealthComponent.die -= BombsSystem.SeparateCurrentBomb;
        }

        private void OnEnter(Collider other)
        {
            var pointToPlant = other.GetComponent<PointToPlant>();

            if (pointToPlant)
            {
                BombsSystem.StartPlanting(pointToPlant);
            }
        }

        private void OnExit(Collider other)
        {
            var pointToPlant = other.GetComponent<PointToPlant>();
```

```
        if (pointToPlant)
        {
           BombsSystem.StopPlanting(pointToPlant);
        }
     }
  }
}using PlantTheBomb.Characters.Abstractions;

namespace PlantTheBomb.Characters.Player
{
  public class PlayerCharacterInitializer :
BaseCharacterInitializer<PlayerCharacterController>
  {
  }
}using PlantTheBomb.Bombs;
using PlantTheBomb.Characters.Abstractions;

namespace PlantTheBomb.Characters.Player
{
  public class PlayerCharacterModel : BaseCharacterModel
  {
     public BombsSystem BombsSystem { get; private set; }

     protected override void InitComponents()
     {
        base.InitComponents();

        BombsSystem = GetComponent<BombsSystem>();
     }

     public void EquipCurrentBomb()
     {
        UnitModel.Equipment.Equip(BombsSystem.CurrentBomb);
     }
  }
}using Super.UI;
using SuperTools.Menus;
using MoreMountains.NiceVibrations;
using PlantTheBomb.Characters.Abstractions;
using PlantTheBomb.Characters.Units.TracerUnit;
using UnityEngine;

namespace PlantTheBomb.Characters.Player
{
  public class PlayerCharacterView : BaseCharacterView
  {
     [SerializeField] private HealthBar _healthBar;

     public override void DisplayHealth(float health, float maxHealth)
```

```
    {
      _healthBar.Display(health / maxHealth);
    }

    public override void TakeDamage(float value)
    {
      base.TakeDamage(value);

      MenuManager.Instance.MenuList.GameMenu.PlayPlayerHit();
      MMVibrationManager.Haptic(HapticTypes.LightImpact);
    }

    public override void SetDeath(Vector3 hitDirection)
    {
      base.SetDeath(hitDirection);
      _healthBar.Hide();
    }

    public void PlayNoMoreBombs()
    {
      if (UnitView is TracerUnitView unit)
        unit.PlayNoMoreBombs();
    }
  }
}using PlantTheBomb.Input;
using UnityEngine;

namespace PlantTheBomb.Characters.Player.Hand
{
  public class PlayerHandController : MonoBehaviour
  {
    public PlayerHandModel Model { get; protected set; }
    protected PlayerHandView View { get; set; }

    protected IBaseInput Input { get; set; }

    public void Init(PlayerHandModel model, PlayerHandView view)
    {
      Model = model;
      View = view;

      Input = Model.Input;
      Input.OnStartPerform += Model.Detonate;
      Model.OnDetonate += View.Detonate;
    }

    private void OnDestroy()
    {
      Input.OnPerformTap -= Model.Detonate;
      Model.OnDetonate -= View.Detonate;
```

```
        }
    }
}using PlantTheBomb.Input;
using UnityEngine;

namespace PlantTheBomb.Characters.Player.Hand
{
    public class PlayerHandInitializer : MonoBehaviour
    {
        private void Awake()
        {
            Init();
        }

        private void Init()
        {
            var model = CreateModel(gameObject);
            var view = CreateView(gameObject);

            var controller = gameObject.GetComponent<PlayerHandController>();
            controller.Init(model, view);
        }

        private PlayerHandModel CreateModel(GameObject handGO)
        {
            var input = handGO.GetComponent<IBaseInput>();

            var model = handGO.GetComponent<PlayerHandModel>();
            model.Init(input);

            return model;
        }

        private PlayerHandView CreateView(GameObject handGO)
        {
            var view = handGO.GetComponent<PlayerHandView>();

            return view;
        }
    }
}using System;
using PlantTheBomb.Input;
using UnityEngine;

namespace PlantTheBomb.Characters.Player.Hand
{
    public class PlayerHandModel : MonoBehaviour
    {
        public IBaseInput Input { get; private set; }
```

```
        public event Action OnDetonate;

        public void Init(IBaseInput input)
        {
            Input = input;
        }

        public void Detonate()
        {
            OnDetonate?.Invoke();
        }
    }
}using UnityEngine;

namespace PlantTheBomb.Characters.Player.Hand
{
    public class PlayerHandView : MonoBehaviour
    {
        private Animator _anim;
        private static readonly int DetonateHash =
Animator.StringToHash("Detonate");

        private void Awake()
        {
            _anim = GetComponentInChildren<Animator>();
        }

        public void Detonate()
        {
            _anim.SetTrigger(DetonateHash);
        }
    }
}using PlantTheBomb.Characters.Abstractions;
using PlantTheBomb.Commands;
using UnityEngine;

namespace PlantTheBomb.Characters.Spawners
{
    public class CharacterSpawner : MonoBehaviour
    {
        [SerializeField] private ECharacters _characterType;

        public BaseCharacterController Spawn()
        {
            var characterGO = new InstantiatePlayerCharacterPrefabCommand()
                .Execute(_characterType, transform.position,
transform.rotation);

            return characterGO.GetComponent<BaseCharacterController>();
        }
```

```
  }
}using CoolTools.StateMachine;
using PlantTheBomb.Characters.Abstractions;
using PlantTheBomb.Movement;
using UnityEngine;

namespace PlantTheBomb.Characters.States
{
  public class DeadState : IState
  {
     private IMovement _movement;
     private BaseCharacterView _view;
     private Vector3 _hitDirection;

     public void Init(IMovement movement, BaseCharacterView view)
     {
        _movement = movement;
        _view = view;
     }

     public void SetHitDirection(Vector3 hitDirection)
     {
        _hitDirection = hitDirection;
     }

     public void OnEnter()
     {
        _movement.IsAvailable = false;
        _view.SetDeath(_hitDirection);
     }

     public void Tick()
     {
     }

     public void FixedTick()
     {
     }

     public void OnExit()
     {
     }
  }
}using CoolTools.StateMachine;
using PlantTheBomb.Characters.Abstractions;
using UnityEngine;

namespace PlantTheBomb.Characters.States
{
  public class IdleState : IState
```

```
  {
     private Rigidbody _unitRb;
     private BaseCharacterView _view;

     public void Init(Rigidbody unitRb, BaseCharacterView view)
     {
        _unitRb = unitRb;
        _view = view;
     }

     public void OnEnter()
     {
        _unitRb.velocity = Vector3.zero;
        _unitRb.angularVelocity = Vector3.zero;
        _view.SetIdle();
     }

     public void Tick()
     {
     }

     public void FixedTick()
     {
     }

     public void OnExit()
     {
     }
  }
}using CoolTools.StateMachine;
using SuperTools.CameraManager;
using PlantTheBomb.Characters.Abstractions;
using PlantTheBomb.Input;
using PlantTheBomb.Movement;
using UnityEngine;

namespace PlantTheBomb.Characters.States
{
  public class MoveState : IState
  {
     private IMovement _unitMovement;
     private IBaseInput _input;

     private BaseCharacterView _view;

     public void Init(IMovement unitMovement, IBaseInput input,
BaseCharacterView view)
     {
        _unitMovement = unitMovement;
        _input = input;
```

```
            _view = view;
        }

        public void OnEnter()
        {
            if (_unitMovement is MonoBehaviour movementMono)
            {
                movementMono.enabled = true;
            }
        }

        public void Tick()
        {
            var joystickValue = new Vector3(_input.InputDirection.x, 0,
_input.InputDirection.y);
            var movementDirection =
                Quaternion.Euler(Vector3.up *
CameraManager.Instance.CurrentManagedCamera.transform.rotation.eulerAngle
s.y) * joystickValue;

            _unitMovement.SetMovementDirection(movementDirection);
            _view.SetWalk(_unitMovement.MaxSpeed * joystickValue.magnitude,
_unitMovement.MaxSpeed);
        }

        public void FixedTick()
        {
        }

        public void OnExit()
        {
            _unitMovement.SetMovementDirection(Vector3.zero);
        }
    }
}using CoolTools.Colliders;
using UnityEngine;

namespace PlantTheBomb.Characters.Units.Abstractions
{
    public class BaseUnitController : MonoBehaviour
    {
        public BaseUnitModel Model { get; protected set; }
        public BaseUnitView View { get; protected set; }

        protected TriggerSource Hitbox { get; set; }

        public void Init(BaseUnitModel model, BaseUnitView view)
        {
            Model = model;
            View = view;
```

```
            InitComponents();
        }

        protected void InitComponents()
        {
            Hitbox = Model.Hitbox;
        }
    }
}using CoolTools.Colliders;
using PlantTheBomb.EquipSystem;
using PlantTheBomb.Movement;
using UnityEngine;

namespace PlantTheBomb.Characters.Units.Abstractions
{
    public class BaseUnitModel : MonoBehaviour
    {
        [field: SerializeField] public Transform BodyCenter { get; private
set; }

        public IMovement Movement { get; private set; }
        public Equipment Equipment { get; private set; }
        public TriggerSource Hitbox { get; private set; }
        public Rigidbody Rb { get; private set; }

        public virtual void Init()
        {
            Movement = GetComponent<IMovement>();
            Equipment = GetComponent<Equipment>();
            Hitbox = GetComponent<TriggerSource>();
            Rb = GetComponent<Rigidbody>();
        }
    }
}using MoreMountains.Feedbacks;
using UnityEngine;

namespace PlantTheBomb.Characters.Units.Abstractions
{
    public class BaseUnitView : MonoBehaviour
    {
        [SerializeField] protected Animator _anim;
        [SerializeField] private MMFeedbacks _hitFeedbacks;

        private static readonly int Walk = Animator.StringToHash("Walk");
        private static readonly int MoveSpeed =
Animator.StringToHash("MoveSpeed");
        private static readonly int HitHash = Animator.StringToHash("Hit");
        private static readonly int Death = Animator.StringToHash("Death");
```

```
        private static readonly int Attack =
Animator.StringToHash("Attack");

        public void SetIdle()
        {
            _anim.SetBool(Walk, false);
        }

        public void SetMove(float moveSpeed, float maxMoveSpeed)
        {
            _anim.SetBool(Walk, true);
            SetMoveSpeed(moveSpeed, maxMoveSpeed);
        }

        private void SetMoveSpeed(float moveSpeed, float maxMoveSpeed)
        {
            _anim.SetFloat(MoveSpeed, moveSpeed / maxMoveSpeed);
        }

        public void TakeDamage()
        {
            _anim.SetTrigger(HitHash);
            _hitFeedbacks.PlayFeedbacks();
        }

        public void SetAttackActive(bool value)
        {
            _anim.SetBool(Attack, value);
        }

        public virtual void SetDeath(Vector3 hitDirection)
        {
            _anim.SetBool(Death, true);
        }
    }
}using PlantTheBomb.Characters.Units.Abstractions;

namespace PlantTheBomb.Characters.Units.TracerUnit
{
    public class TracerUnitController : BaseUnitController
    {

    }
}using PlantTheBomb.Characters.Units.Abstractions;

namespace PlantTheBomb.Characters.Units.TracerUnit
{
    public class TracerUnitModel : BaseUnitModel
    {
```

```
  }
}using PlantTheBomb.Characters.Units.Abstractions;
using UnityEngine;
using UnityEngine.Animations.Rigging;

namespace PlantTheBomb.Characters.Units.TracerUnit
{
  public class TracerUnitView : BaseUnitView
  {
     [SerializeField] private Rig _holdBombRig;
     [SerializeField] private Transform _rootSpine;
     [SerializeField] private float _deathImpulseForce;
     private Rigidbody[] _ragdollRbParts;
     private Collider[] _ragdollColliderParts;

     private void Awake()
     {
        _ragdollRbParts =
_rootSpine.GetComponentsInChildren<Rigidbody>();
        _ragdollColliderParts = new Collider[_ragdollRbParts.Length];

        for (var i = 0; i != _ragdollRbParts.Length; ++i)
        {
           _ragdollColliderParts[i] =
_ragdollRbParts[i].GetComponent<Collider>();
        }

        SetRagdollActive(false);
     }

     private void SetRagdollActive(bool value)
     {
        for (var i = 0; i != _ragdollRbParts.Length; ++i)
        {
           _ragdollRbParts[i].isKinematic = !value;
           _ragdollColliderParts[i].enabled = value;
        }

        _anim.enabled = !value;

        _holdBombRig.weight = value ? 0 : 1;
     }

     public override void SetDeath(Vector3 hitDirection)
     {
        GetComponentInParent<Collider>().enabled = false;
        SetRagdollActive(true);
        _rootSpine.GetComponent<Rigidbody>().AddForce(hitDirection *
_deathImpulseForce, ForceMode.Impulse);
     }
```

```
        public void PlayNoMoreBombs()
        {
            _holdBombRig.weight = 0;
        }
    }
}using System.Threading.Tasks;
using PlantTheBomb.Characters.Player;
using PlantTheBomb.LevelSystem;
using PlantTheBomb.SaveSystem;
using UnityEngine.SceneManagement;

namespace PlantTheBomb.Commands
{
    public class InitLevelCommand
    {
        private LevelModel _currentLevelModel;
        private PlayerCharacterController _player;

        public async void Execute()
        {
            Save.Load();

            await CreateLevel();

            CreatePlayer();
        }

        private async Task CreateLevel()
        {
            _currentLevelModel = await
LevelsLoader.Instance.LoadCurrentLevelScene(LoadSceneMode.Additive);
        }

        private void CreatePlayer()
        {
            _player = (PlayerCharacterController)
_currentLevelModel.PlayerSpawner.Spawn();
        }
    }
}using UnityEngine;

namespace PlantTheBomb.Commands
{
    public class InstantiatePlayerCharacterPrefabCommand
    {
        private const string CHARACTER_PATH = "Prefabs/Characters/";

        public GameObject Execute(ECharacters character, Vector3 spawnPos,
Quaternion spawnRot, Transform parent = null)
```

```
    {
        var characterObject = Resources.Load(CHARACTER_PATH +
character.ToString());

        var characterGO = Object.Instantiate(characterObject, spawnPos,
spawnRot, parent) as GameObject;

        characterGO.name = characterObject.name;

        return characterGO;
    }
  }
}using PlantTheBomb.EquipSystem;
using UnityEngine;

namespace PlantTheBomb.Bombs
{
  public class Bomb : MonoBehaviour, IEquipable
  {
    public Transform Transform => transform;
  }
}using System;
using SuperTools.Menus;
using PlantTheBomb.LevelSystem;
using UnityEngine;

namespace PlantTheBomb.Bombs
{
  public class BombsSystem : MonoBehaviour
  {
    [field: SerializeField] public Bomb BombPrefab { get; private set; }
    private int _totalAmount { get; set; }

    public event Action onCreateBomb;
    public event Action noMoreBombs;

    public Bomb CurrentBomb { get; private set; }
    private int _currentBombId = -1;

    private void Start()
    {
        _totalAmount =
FindObjectOfType<LevelModel>().LevelProgression.AmountToPlant;
    }

    public async void StartPlanting(PointToPlant pointToPlant)
    {
        if (await pointToPlant.TryStartPlanting(CurrentBomb))
            TryCreateNextBomb();
    }
```

```
      public void TryCreateNextBomb()
      {
         _currentBombId++;

         if (_currentBombId >= 0 && _currentBombId < _totalAmount)
         {
            var nextBomb = Instantiate(BombPrefab);
            CurrentBomb = nextBomb;
            onCreateBomb?.Invoke();
         }
         else
         {
            noMoreBombs?.Invoke();
         }


MenuManager.Instance.MenuList.GameMenu.DisplayBombsAmount(_currentBombId,
_totalAmount);
      }

      public void StopPlanting(PointToPlant pointToPlant)
      {
         pointToPlant.StopPlanting();
      }

      public void SeparateCurrentBomb()
      {
         CurrentBomb.transform.parent = null;
         CurrentBomb.GetComponent<Collider>().enabled = true;
         CurrentBomb.GetComponent<Rigidbody>().isKinematic = false;
      }
   }
}using System;
using System.Threading;
using System.Threading.Tasks;
using Super.UI;
using SuperTools.Menus;
using SuperTools.Menus.Utils;
using SuperTools.PoolParty;
using MoreMountains.Feedbacks;
using MoreMountains.NiceVibrations;
using UnityEngine;

namespace PlantTheBomb.Bombs
{
   public class PointToPlant : MonoBehaviour
   {
      [field: SerializeField] private Transform Point { get; set; }
      [field: SerializeField] private float PlantDuration { get; set; }
```

```
        [SerializeField] private ParticleSystem _hintParticles;
        [SerializeField] private UITargetMarker _pointToPlantUiMarkerPrefab;
        [SerializeField] private PlantingUIMarker _plantingUiMarkerPrefab;

        [SerializeField] private MMFeedbacks _plantedFeedbacks;

        private UITargetMarker _pointToPlantUiMarker;
        private PlantingUIMarker _plantingUiMarker;

        public event Action onPlanted;

        private float _progress;
        private bool _isPlanting;
        private bool _isAvailable = true;

        private CancellationTokenSource _plantingCts;

        private void Start()
        {
           CreatePointToPlantMarker();
        }

        private void CreatePointToPlantMarker()
        {
           _pointToPlantUiMarker = Instantiate(_pointToPlantUiMarkerPrefab,
Vector3.zero, Quaternion.identity,

MenuManager.Instance.MenuList.GameMenu.GetComponentInChildren<SafeArea>()
.transform);
           _pointToPlantUiMarker.Init(transform);
        }

        public async Task<bool> TryStartPlanting(Bomb bomb)
        {
           if (!_isAvailable || _isPlanting) return false;

           _isPlanting = true;

           _plantingCts = new CancellationTokenSource();

           CreatePlantingUIMarker();

           while (_progress < 1)
           {
              _progress += Time.deltaTime / PlantDuration;
              _plantingUiMarker.Display(_progress);
              await Task.Yield();

              if (_plantingCts.IsCancellationRequested)
                 return false;
```

```
        }

        bomb.transform.parent = Point;
        bomb.transform.SetPositionAndRotation(Point.position,
Point.rotation);

        _plantedFeedbacks.PlayFeedbacks();
        MMVibrationManager.Haptic(HapticTypes.MediumImpact);

        HideAllView();

        onPlanted?.Invoke();

        _isAvailable = false;
        _isPlanting = false;

        return true;
    }

    private void CreatePlantingUIMarker()
    {
        _plantingUiMarker =
PoolParty.Instantiate(_plantingUiMarkerPrefab, Vector3.zero,
Quaternion.identity,

MenuManager.Instance.MenuList.GameMenu.GetComponentInChildren<SafeArea>()
.transform);

        _plantingUiMarker.Init(transform.position);
    }

    private void HideAllView()
    {
        _hintParticles.gameObject.SetActive(false);

        if (_pointToPlantUiMarker)
            _pointToPlantUiMarker.Hide();

        if (_plantingUiMarker)
            _plantingUiMarker.Hide();
    }

    public void StopPlanting()
    {
        _plantingCts.Cancel();
        _isPlanting = false;
        _progress = 0;

        if (_isAvailable && _plantingUiMarker)
            _plantingUiMarker.Hide();
```

```
        }
    }
}using UnityEngine;

namespace PlantTheBomb.Damage
{
    public static class DamageService
    {
        public static void PerformDamage(IDamageProducer producer,
IDamageReceiver receiver, Vector3 direction)
        {
            receiver.ReceiveDamage(producer.Damage, direction);
        }
    }
}using SuperTools.PoolParty;
using UnityEngine;

namespace PlantTheBomb.Damage
{
    public class Firearm : Weapon
    {
        [SerializeField] private Projectile _projectilePrefab;
        [SerializeField] private Transform _bulletSpawnPoint;

        [SerializeField] private float _yShootOffset;

        private void Awake()
        {
            cooldownTimer = _cooldown;
        }

        protected override void UseWeapon()
        {
            base.UseWeapon();

            var projectile = PoolParty.Instantiate(_projectilePrefab,
_bulletSpawnPoint.position, transform.rotation);

            var targetPos = ((MonoBehaviour)damageTarget).transform.position
+ new Vector3(0, _yShootOffset, 0);

            projectile.Init(this, targetPos);
        }
    }
}namespace PlantTheBomb.Damage
{
    public interface IDamageProducer
    {
        public float Damage { get; set; }
    }
```

```
}using UnityEngine;

namespace PlantTheBomb.Damage
{
  public interface IDamageReceiver
  {
     public void ReceiveDamage(float damage, Vector3 direction);
  }
}using UnityEngine;

namespace PlantTheBomb.Damage
{
  public class InstantGun : Weapon
  {
     protected override void UseWeapon()
     {
        base.UseWeapon();

        if (damageTarget != null)
        {
           DamageService.PerformDamage(this, damageTarget,
transform.forward);

           Debug.Log("REAL DAMAGE");
        }
     }
  }
}using System.Collections;
using System.Threading.Tasks;
using SuperTools.PoolParty;
using UnityEngine;

namespace PlantTheBomb.Damage
{
  public class Projectile : MonoBehaviour
  {
     [SerializeField] private float _speed;
     [SerializeField] private float _lifetime;

     private Vector3 _moveDirection;

     private Weapon _weaponSource;

     public void Init(Weapon weapon, Vector3 moveDirection)
     {
        _weaponSource = weapon;
        _moveDirection = moveDirection - transform.position;
        transform.forward = _moveDirection;
     }
```

```
    private void OnEnable()
    {
        StartCoroutine(DestroyAfterLifetime());
        StartMove();
    }

    private IEnumerator DestroyAfterLifetime()
    {
        yield return new WaitForSeconds(_lifetime);

        PoolParty.Destroy(this);
    }

    private async void StartMove()
    {
        while (gameObject.activeSelf)
        {
            Move();

            await Task.Yield();
        }
    }

    protected virtual void Move()
    {
        transform.position += _moveDirection * _speed * Time.deltaTime;
    }

    private void OnTriggerEnter(Collider other)
    {
        var damageReceiver =
other.GetComponentInParent<IDamageReceiver>();

        if (damageReceiver != null)
        {
            DamageService.PerformDamage(_weaponSource, damageReceiver,
transform.forward);
        }

        PoolParty.Destroy(this);
    }
  }
}using System.Threading.Tasks;
using MoreMountains.Feedbacks;
using PlantTheBomb.EquipSystem;
using UnityEngine;

namespace PlantTheBomb.Damage
{
```

```
  public abstract class Weapon : MonoBehaviour, IDamageProducer,
IEquipable
  {
     public Transform Transform => transform;

     [field: SerializeField] public float Damage { get; set; }
     [SerializeField] protected float _cooldown;
     [SerializeField] private MMFeedbacks _useWeaponFeedback;

     protected float cooldownTimer;
     protected bool isAvailable = true;

     public Transform HoldingHand { get; set; }

     protected IDamageReceiver damageTarget;

     public void SetTarget(IDamageReceiver target)
     {
        damageTarget = target;
     }

     public async void TryAttack()
     {
        if (!isAvailable) return;

        UseWeapon();

        await CooldownRoutine();
     }

     protected virtual void UseWeapon()
     {
        _useWeaponFeedback.PlayFeedbacks();
     }

     private async Task CooldownRoutine()
     {
        isAvailable = false;

        cooldownTimer = _cooldown;

        while (cooldownTimer > 0)
        {
           cooldownTimer -= Time.deltaTime;

           await Task.Yield();
        }

        isAvailable = true;
     }
```

```
        public void Equip(Transform holdingHand)
        {
            HoldingHand = holdingHand;
            transform.parent = HoldingHand;
            transform.position = HoldingHand.position;
            transform.localRotation = Quaternion.Euler(Vector3.zero);
            isAvailable = true;
        }

        public void Unequip(Transform target)
        {
            transform.parent = target;
            transform.position = target.transform.position;
            transform.rotation = Quaternion.identity;

            HoldingHand = null;
            isAvailable = false;
        }
    }
}using System;
using System.Threading.Tasks;
using Sirenix.OdinInspector;
using UnityEngine;

namespace PlantTheBomb.Damage.Health
{
    public class HealthComponent : MonoBehaviour
    {
        [field: SerializeField] public float MaxHealth { get; private set; }
        [SerializeField] private float _invincibleDuration;

        public float CurrentHealth { get; private set; }
        public bool IsDead { get; private set; }

        public event Action<float> increaseHealth;
        public event Action<float> takeDamage;
        public event Action healthChanged;
        public event Action die;

        private bool _isInvincible;

        private void Awake()
        {
            CurrentHealth = MaxHealth;
        }

        private void OnEnable()
        {
            increaseHealth += OnIncreaseHealth;
```

```
        takeDamage += OnTakeDamage;
        die += OnDie;
    }

    private void OnDisable()
    {
        increaseHealth -= OnIncreaseHealth;
        takeDamage -= OnTakeDamage;
        die -= OnDie;
    }

    private void OnIncreaseHealth(float value)
    {
        CurrentHealth += value;
    }

    private void OnTakeDamage(float damage)
    {
        CurrentHealth -= damage;

        if (CurrentHealth <= 0)
            Die();
    }

    private void OnDie()
    {
        IsDead = true;
    }

    public void IncreaseHealth(float value)
    {
        increaseHealth?.Invoke(value);
        healthChanged?.Invoke();
    }

    public bool TryTakeDamage(float damage)
    {
        if (_isInvincible || IsDead) return false;

        takeDamage?.Invoke(damage);
        healthChanged?.Invoke();

        return true;
    }

    [Button]
    private void Die()
    {
        die?.Invoke();
    }
```

```
        public void MakeInvincible(bool value)
        {
            _isInvincible = value;
        }

        public async void MakeInvincibleForTime()
        {
            if (_isInvincible) return;

            await MakeInvincibleForTimeRoutine();
        }

        private async Task MakeInvincibleForTimeRoutine()
        {
            _isInvincible = true;

            await new WaitForSeconds(_invincibleDuration);

            _isInvincible = false;
        }
    }
}using UnityEngine;

namespace PlantTheBomb.EquipSystem
{
    public class Equipment : MonoBehaviour
    {
        [field: SerializeField] public Transform ItemsHolder { get; private
set; }

        public void Equip(IEquipable item)
        {
            item.Transform.parent = ItemsHolder;
            item.Transform.localPosition = Vector3.zero;
            item.Transform.localRotation = Quaternion.identity;
        }
    }
}using UnityEngine;

namespace PlantTheBomb.EquipSystem
{
    public interface IEquipable
    {
        public Transform Transform { get; }
    }
}using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

```
namespace PlantTheBomb.FieldOfViewSystem
{
  public class FieldOfView : MonoBehaviour {

  public float viewRadius;
  [Range(0,360)]
  public float viewAngle;

  public LayerMask targetMask;
  public LayerMask obstacleMask;

  [HideInInspector]
  public List<Transform> visibleTargets = new List<Transform>();

  public float meshResolution;
  public int edgeResolveIterations;
  public float edgeDstThreshold;

  public MeshFilter viewMeshFilter;
  Mesh viewMesh;

  public Material StartMaterial { get; private set; }
  [field: SerializeField] public Material TriggerMaterial { get; private
set; }

  private Vector3 _startPos;
  private Renderer _renderer;

  private bool _isDrawActive;

  void Start() {
     viewMesh = new Mesh ();
     viewMesh.name = "View Mesh";
     viewMeshFilter.mesh = viewMesh;

     _renderer = GetComponent<Renderer>();
     StartMaterial = _renderer.material;
     _startPos = transform.localPosition;

     StartCoroutine (nameof(FindTargetsWithDelay), .2f);
  }

  public void SetDrawActive(bool value)
  {
     _isDrawActive = value;

     if (!value)
        viewMesh.Clear ();
  }
```

```
  public void SetMaterial(Material material)
  {
     _renderer.material = material;

     if (material == StartMaterial)
        transform.localPosition = _startPos;
     else
        transform.localPosition = new Vector3(_startPos.x, _startPos.y +
0.01f, _startPos.z);  // to avoid z fight between start material and
trigger material
  }

  IEnumerator FindTargetsWithDelay(float delay) {
     while (true) {
        yield return new WaitForSeconds (delay);
        FindVisibleTargets ();
     }
  }

  void LateUpdate() {
     if (_isDrawActive)
        DrawFieldOfView ();
  }

  void FindVisibleTargets() {
     visibleTargets.Clear ();
     Collider[] targetsInViewRadius = Physics.OverlapSphere
(transform.position, viewRadius, targetMask);

     for (int i = 0; i < targetsInViewRadius.Length; i++) {
        Transform target = targetsInViewRadius [i].transform;
        Vector3 dirToTarget = (target.position -
transform.position).normalized;
        if (Vector3.Angle (transform.forward, dirToTarget) < viewAngle /
2) {
           float dstToTarget = Vector3.Distance (transform.position,
target.position);
           if (!Physics.Raycast (transform.position, dirToTarget,
dstToTarget, obstacleMask)) {
              visibleTargets.Add (target);
           }
        }
     }
  }

  void DrawFieldOfView() {
     int stepCount = Mathf.RoundToInt(viewAngle * meshResolution);
     float stepAngleSize = viewAngle / stepCount;
     List<Vector3> viewPoints = new List<Vector3> ();
     ViewCastInfo oldViewCast = new ViewCastInfo ();
```

```
      for (int i = 0; i <= stepCount; i++) {
         float angle = transform.eulerAngles.y - viewAngle / 2 +
stepAngleSize * i;
         ViewCastInfo newViewCast = ViewCast (angle);

         if (i > 0) {
            bool edgeDstThresholdExceeded = Mathf.Abs (oldViewCast.dst -
newViewCast.dst) > edgeDstThreshold;
            if (oldViewCast.hit != newViewCast.hit || (oldViewCast.hit &&
newViewCast.hit && edgeDstThresholdExceeded)) {
               EdgeInfo edge = FindEdge (oldViewCast, newViewCast);
               if (edge.pointA != Vector3.zero) {
                  viewPoints.Add (edge.pointA);
               }
               if (edge.pointB != Vector3.zero) {
                  viewPoints.Add (edge.pointB);
               }
            }

         }


         viewPoints.Add (newViewCast.point);
         oldViewCast = newViewCast;
      }

      int vertexCount = viewPoints.Count + 1;
      Vector3[] vertices = new Vector3[vertexCount];
      int[] triangles = new int[(vertexCount-2) * 3];

      vertices [0] = Vector3.zero;
      for (int i = 0; i < vertexCount - 1; i++) {
         vertices [i + 1] = transform.InverseTransformPoint(viewPoints
[i]);

         if (i < vertexCount - 2) {
            triangles [i * 3] = 0;
            triangles [i * 3 + 1] = i + 1;
            triangles [i * 3 + 2] = i + 2;
         }
      }

      viewMesh.Clear ();

      viewMesh.vertices = vertices;
      viewMesh.triangles = triangles;
      viewMesh.RecalculateNormals ();
   }
```

```
  EdgeInfo FindEdge(ViewCastInfo minViewCast, ViewCastInfo maxViewCast) {
     float minAngle = minViewCast.angle;
     float maxAngle = maxViewCast.angle;
     Vector3 minPoint = Vector3.zero;
     Vector3 maxPoint = Vector3.zero;

     for (int i = 0; i < edgeResolveIterations; i++) {
        float angle = (minAngle + maxAngle) / 2;
        ViewCastInfo newViewCast = ViewCast (angle);

        bool edgeDstThresholdExceeded = Mathf.Abs (minViewCast.dst -
newViewCast.dst) > edgeDstThreshold;
        if (newViewCast.hit == minViewCast.hit &&
!edgeDstThresholdExceeded) {
           minAngle = angle;
           minPoint = newViewCast.point;
        } else {
           maxAngle = angle;
           maxPoint = newViewCast.point;
        }
     }

     return new EdgeInfo (minPoint, maxPoint);
  }


  ViewCastInfo ViewCast(float globalAngle) {
     Vector3 dir = DirFromAngle (globalAngle, true);
     RaycastHit hit;

     if (Physics.Raycast (transform.position, dir, out hit, viewRadius,
obstacleMask)) {
        return new ViewCastInfo (true, hit.point, hit.distance,
globalAngle);
     } else {
        return new ViewCastInfo (false, transform.position + dir *
viewRadius, viewRadius, globalAngle);
     }
  }

  public Vector3 DirFromAngle(float angleInDegrees, bool angleIsGlobal) {
     if (!angleIsGlobal) {
        angleInDegrees += transform.eulerAngles.y;
     }
     return new Vector3(Mathf.Sin(angleInDegrees *
Mathf.Deg2Rad),0,Mathf.Cos(angleInDegrees * Mathf.Deg2Rad));
  }

  public struct ViewCastInfo {
     public bool hit;
```

```
    public Vector3 point;
    public float dst;
    public float angle;

    public ViewCastInfo(bool _hit, Vector3 _point, float _dst, float
_angle) {
      hit = _hit;
      point = _point;
      dst = _dst;
      angle = _angle;
    }
  }

  public struct EdgeInfo {
    public Vector3 pointA;
    public Vector3 pointB;

    public EdgeInfo(Vector3 _pointA, Vector3 _pointB) {
      pointA = _pointA;
      pointB = _pointB;
    }
  }

  }
}using UnityEditor;
using UnityEngine;

namespace PlantTheBomb.FieldOfViewSystem.Editor
{
  [CustomEditor (typeof (FieldOfView))]
  public class FieldOfViewEditor : UnityEditor.Editor {

    void OnSceneGUI() {
      FieldOfView fow = (FieldOfView)target;
      Handles.color = Color.white;
      Handles.DrawWireArc (fow.transform.position, Vector3.up,
Vector3.forward, 360, fow.viewRadius);
      Vector3 viewAngleA = fow.DirFromAngle (-fow.viewAngle / 2,
false);
      Vector3 viewAngleB = fow.DirFromAngle (fow.viewAngle / 2, false);

      Handles.DrawLine (fow.transform.position, fow.transform.position
+ viewAngleA * fow.viewRadius);
      Handles.DrawLine (fow.transform.position, fow.transform.position
+ viewAngleB * fow.viewRadius);

      Handles.color = Color.red;
      foreach (Transform visibleTarget in fow.visibleTargets) {
        Handles.DrawLine (fow.transform.position,
visibleTarget.position);
```

```
            }
        }

    }
}using System;
using UnityEngine;

namespace PlantTheBomb.Input
{
    public class EmptyInput : MonoBehaviour, IBaseInput
    {
        public bool IsAvailable { get; set; } = true;
        public Vector2 InputDirection { get; set; }
        public Action OnStartPerform { get; set; }
        public Action OnPerform { get; set; }
        public Action OnStopPerform { get; set; }
        public Action OnPerformTap { get; set; }
        public void Init()
        {
        }

        public void StartPerform()
        {
        }

        public void Perform()
        {
        }

        public void StopPerform()
        {
        }

        public void PerformTap()
        {
        }
    }
}using System;
using UnityEngine;

namespace PlantTheBomb.Input
{
    public interface IBaseInput
    {
        public bool IsAvailable { get; set; }
        public Vector2 InputDirection { get; set; }

        public Action OnStartPerform { get; set; }
        public Action OnPerform { get; set; }
        public Action OnStopPerform { get; set; }
```

```
        public Action OnPerformTap { get; set; }

        public void Init();

        public void StartPerform();
        public void Perform();
        public void StopPerform();
        public void PerformTap();
    }
}using System;
using SuperTools.VirtualJoysticks;
using UnityEngine;

namespace PlantTheBomb.Input
{
    public class JoystickInput : MonoBehaviour, IBaseInput
    {
        [SerializeField] private VirtualJoystick _joystickPrefab;
        private VirtualJoystick _joystick;

        private bool _isAvailable;
        public bool IsAvailable
        {
            get => _isAvailable;
            set
            {
                if (value == true)
                    _joystick.Activate();
                else
                    _joystick.Deactivate();

                _isAvailable = value;
            }
        }

        public Vector2 InputDirection
        {
            get =>
                (_isAvailable && _joystick.InputIsMinValid)
                    ? _joystick.Input
                    : Vector2.zero;
            set { }
        }

        public Action OnStartPerform { get; set; }
        public Action OnPerform { get; set; }
        public Action OnStopPerform { get; set; }
        public Action OnPerformTap { get; set; }

        public void Init()
```

```
      {
         if (!_joystick)
         {
            _joystick = Instantiate(_joystickPrefab);
            IsAvailable = true;
         }
      }

      private void OnDestroy()
      {
         if (_joystick)
            Destroy(_joystick.gameObject);
      }

      public void StartPerform()
      {
      }

      public void Perform()
      {
      }

      public void StopPerform()
      {
      }

      public void PerformTap()
      {
      }
   }
}using System;
using SuperTools.TouchInput;
using UnityEngine;

namespace PlantTheBomb.Input
{
   public class TapInput : BaseTouchInput, IBaseInput
   {
      public bool IsAvailable { get; set; }
      public Vector2 InputDirection { get; set; }
      public Action OnStartPerform { get; set; }
      public Action OnPerform { get; set; }
      public Action OnStopPerform { get; set; }
      public Action OnPerformTap { get; set; }

      public void Init()
      {
      }

      protected override void OnDown(FingerTouch finger)
```

```
      {
         StartPerform();
      }

      protected override void OnHeld(FingerTouch finger)
      {
         Perform();
      }

      protected override void OnUp(FingerTouch finger)
      {
         StopPerform();
      }

      protected override void OnTap(FingerTouch finger)
      {
         PerformTap();
      }

      public void StartPerform()
      {
         OnStartPerform?.Invoke();
      }

      public void Perform()
      {
         OnPerform?.Invoke();
      }

      public void StopPerform()
      {
         OnStopPerform?.Invoke();
      }

      public void PerformTap()
      {
         OnPerformTap?.Invoke();
      }
   }
}using System;
using System.Threading.Tasks;
using UnityEngine;

namespace PlantTheBomb.Movement
{
   public interface IMovement
   {
      public bool IsAvailable { get; set; }
      public float MaxSpeed { get; }
      public void SetMovementDirection(Vector3 movementDirection);
```

```
        public Task SetDestination(Vector3 target, Action callback = null);
    }
}using UnityEngine;

namespace PlantTheBomb.Movement.Focus
{
    public class TargetFocuser
    {
        private readonly Transform _owner;
        private Transform _target;

        public TargetFocuser(Transform owner)
        {
            _owner = owner;
        }

        public void SetTarget(Transform target)
        {
            _target = target;
        }

        public void Focus()
        {
            var targetForward = _target.transform.position - _owner.position;
            targetForward.y = 0f;

            var angle = Vector3.Angle(_owner.forward, targetForward);
            _owner.forward = Vector3.RotateTowards(_owner.forward,
targetForward, angle * Time.deltaTime * .1f, 1f);
        }
    }
}using System;
using System.Threading.Tasks;
using PlantTheBomb.Characters.Abstractions;
using UnityEngine;
using UnityEngine.AI;
using Random = UnityEngine.Random;

namespace PlantTheBomb.Movement.Patrolling
{
    [RequireComponent(typeof(NavMeshAgent))]
    public class NavMeshAgentPlus : MonoBehaviour, IMovement
    {
        public enum State
        {
            Idle = 0,
            Moving = 1,
            Custom = 2
        }
```

```
public enum Behaviour
{
    None = 0,
    Patrol = 1,
    Rotate = 2
}

public bool IsAvailable { get; set; } = true;

public State state;
public Behaviour behaviour;
public float pauseDuration = 2f;

[Header("Patrol")]
public PatrollingWaypoint[] waypoints;
public bool randomWaypoint;

[Header("Rotate")]
public float[] angles = new float[1] { 0f };
public float rotationSpeed = 180f;
public bool randomAngle;

NavMeshAgent agent;
public NavMeshAgent Agent => agent;

public event Action OnDestinationReached;
Vector3? destination = null;
bool triggerDestinationReachedEvent;
Vector3 prevPosition;
Quaternion prevRotation;
float lastInstructionTime;
private int _prevIndex = -1;
int index = -1;

Vector3[] prevWaypointsPosition;
Vector3 prevPatrolPosition;

private BaseCharacterView _view;
private int _waypointsMovementOrder = 1;
private float _startSpeed;

public float MaxSpeed => _startSpeed;

void Awake()
{
    prevPosition = transform.position;
    prevRotation = transform.rotation;
    lastInstructionTime = Time.time;

    OnDestinationReached += BackToIdle;
```

```
            prevWaypointsPosition = new Vector3[waypoints.Length];

            for (int i = 0; i < prevWaypointsPosition.Length; i++)
                prevWaypointsPosition[i] =
waypoints[i].transform.position;

            agent = GetComponent<NavMeshAgent>();
            _startSpeed = agent.speed;
        }

        public void Init(BaseCharacterView view)
        {
            _view = view;
        }

        public void SetActive(bool value)
        {
            if (value)
                agent.speed = _startSpeed;
            else
                agent.speed = 0;
        }

        void BackToIdle()
        {
            if (state != State.Custom)
            {
                state = State.Idle;
                lastInstructionTime = Time.time;

                if (waypoints[index].Type !=
PatrollingWaypoint.WaypointType.NoAction)
                    _view.SetIdle();
            }
        }

        public void Tick()
        {
            if (behaviour == Behaviour.Patrol)
                Update_Patrol();
            else if (behaviour == Behaviour.Rotate)
                Update_Rotate();
        }

        void DestinationReached()
        {
            if (state == State.Moving)
            {
                triggerDestinationReachedEvent = false;
```

```
                OnDestinationReached?.Invoke();

                if (waypoints[index].Type ==
PatrollingWaypoint.WaypointType.ReverseOrder)
                {
                    if (_prevIndex == -1) return;

                    _waypointsMovementOrder *= -1;
                }
            }
        }

        public Task SetDestination(Vector3 target, Action callback = null)
        {
            return null;
        }

        public void SetAgentDestination(Vector3 destination)
        {
            if (state != State.Custom)
            {
                state = State.Custom;

                prevPatrolPosition = transform.position;
            }

            Agent.SetDestination(destination);
        }

        public void BackToNormal()
        {
            Debug.Log("BackToNormal");

            state = State.Moving;
            Agent.SetDestination(prevPatrolPosition);
        }

        #region Patrol
        void Update_Patrol()
        {
            if (IsAgentWorking())
            {
                CheckDestinationReached();
                UpdateWaypointsPosition();

                if (state == State.Idle)
                {
                    if (Time.time - lastInstructionTime > pauseDuration ||
(index >= 0 && waypoints[index].Type !=
PatrollingWaypoint.WaypointType.Stoppable))
```

```
                {
                    MoveToNextWaypoint();
                }
            }
            else
            {
                _view.SetWalk(agent.speed, MaxSpeed);
            }
        }
    }

    bool IsAgentWorking()
    {
        return Agent.enabled && Agent.isOnNavMesh && IsAvailable;
    }

    void CheckDestinationReached()
    {
        if (destination != Agent.destination)
        {
            triggerDestinationReachedEvent = destination != null;
            destination = Agent.destination;
        }

        if (triggerDestinationReachedEvent && Agent.remainingDistance
< .1f)
        {
            DestinationReached();
        }
    }

    void UpdateWaypointsPosition()
    {
        if (transform.position != prevPosition || transform.rotation
!= prevRotation)
        {
            for (int i = 0; i < waypoints.Length; i++)
                waypoints[i].transform.position =
prevWaypointsPosition[i];

            prevPosition = transform.position;
            prevRotation = transform.rotation;
        }
        else
        {
            for (int i = 0; i < prevWaypointsPosition.Length; i++)
                prevWaypointsPosition[i] =
waypoints[i].transform.position;
        }
    }
```

```
        void MoveToNextWaypoint()
        {
        
Agent.SetDestination(waypoints[GetNextWaypointIndex()].transform.position
);

            state = State.Moving;
        }

        int GetNextWaypointIndex()
        {
            _prevIndex = index;

            if (!randomWaypoint)
            {
                index = (index + 1 * _waypointsMovementOrder) %
waypoints.Length;

                if (index < 0)
                    index = waypoints.Length + index;

                agent.autoBraking = waypoints[index].Type ==
PatrollingWaypoint.WaypointType.Stoppable;
            }
            else if (waypoints.Length > 1)
            {
                int prevIndex = index;

                do index = Random.Range(0, waypoints.Length);
                while (index == prevIndex);
            }
            else
                index = 0;

            return index;
        }
        #endregion

        #region Rotate
        void Update_Rotate()
        {
            if (state == State.Idle)
            {
                if (Time.time - lastInstructionTime > pauseDuration ||
(index >= 0 && waypoints[index].Type !=
PatrollingWaypoint.WaypointType.Stoppable))
                {
                    StartRotating();
                }
```

```
            }
            else if (state == State.Moving)
            {
                RotateTowardsAngle();
            }
        }

        void StartRotating()
        {
            GetNextAngleIndex();

            state = State.Moving;
        }

        int GetNextAngleIndex()
        {
            if (!randomAngle)
            {
                index = (index + 1 * _waypointsMovementOrder) %
angles.Length;

                if (index < 0)
                    index = waypoints.Length + index;
            }
            else if (angles.Length > 1)
            {
                int prevIndex = index;

                do index = Random.Range(0, angles.Length);
                while (index == prevIndex);
            }
            else
                index = 0;

            return index;
        }

        Vector3 GetAngleTargetPoint(float localAngle, float magnitude =
1f)
        {
            return transform.position + Quaternion.Euler(Vector3.up *
(localAngle + transform.rotation.eulerAngles.y -
transform.localRotation.eulerAngles.y)) * Vector3.forward * magnitude;
        }

        void RotateTowardsAngle()
        {
            Vector3 targetForward = GetAngleTargetPoint(angles[index]) -
transform.position;
            targetForward.y = 0f;
```

```
            float angle = Vector3.Angle(transform.forward, targetForward);

            if (angle > 0f)
                transform.forward =
Vector3.RotateTowards(transform.forward, targetForward, rotationSpeed *
Mathf.Deg2Rad * Time.deltaTime, 1f);
            else
                BackToIdle();
        }
        #endregion

        void OnDrawGizmos()
        {
            if (behaviour == Behaviour.Patrol)
            {
                Gizmos.color = Color.white;
                foreach (PatrollingWaypoint waypoint in waypoints)
                    if (waypoint != null)
                        Gizmos.DrawLine(transform.position,
waypoint.transform.position);

                Gizmos.color = Color.red;
                for (int i = 0; i < waypoints.Length; i++)
                    for (int j = 0; j < waypoints.Length; j++)
                        if (i != j && waypoints[i] != null && waypoints[j]
!= null)

Gizmos.DrawLine(waypoints[i].transform.position,
waypoints[j].transform.position);
            }
            else if (behaviour == Behaviour.Rotate)
            {
                Gizmos.color = Color.red;
                foreach (float angle in angles)
                    Gizmos.DrawLine(transform.position,
GetAngleTargetPoint(angle, 5f));
            }
        }

        public void SetMovementDirection(Vector3 movementDirection)
        {
        }
    }
}using UnityEngine;

namespace PlantTheBomb.Movement.Patrolling
{
  public class PatrollingWaypoint : MonoBehaviour
  {
```

```
      public enum WaypointType
      {
         NoAction,
         Stoppable,
         ReverseOrder
      }

      [field: SerializeField] public WaypointType Type { get; private set;
}
  }
}using SuperTools.Core.Extensions;
using UnityEngine;

namespace PlantTheBomb.Movement.TracerController
{
  public struct CollisionData
  {
     public GameObject gameObject;
     public Vector3 normal;
     public Vector3 center;

     public CollisionData(Collision collision)
     {
        gameObject = collision.gameObject;
        normal = collision.GetAverageNormal();
        center = collision.GetAverageCenter();
     }
  }

}using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using FluffyUnderware.Curvy;
using FluffyUnderware.Curvy.Controllers;
using SuperTools.Core.Extensions;
using UnityEngine;

namespace PlantTheBomb.Movement.TracerController
{
  [RequireComponent(typeof(Rigidbody), typeof(CapsuleCollider),
typeof(SplineController))]
   public class TracerController : MonoBehaviour, IMovement
   {
       public bool IsAvailable { get; set; } = true;

       public bool canMove = true;
       public float moveSpeed = 5f;
       public bool canJump = true;
       public float jumpForce = 10f;
       public LayerMask walkableLayers;
```

```
        public LayerMask climbableLayers;
        public LayerMask ropeLayers;

        List<GameObject> grounds = new List<GameObject>();
        List<GameObject> walls = new List<GameObject>();
        List<GameObject> ropes = new List<GameObject>();

        List<CollisionData> collisions = new List<CollisionData>();
        List<CollisionData> groundsIn = new List<CollisionData>();
        List<GameObject> groundsOut = new List<GameObject>();
        List<CollisionData> wallsIn = new List<CollisionData>();
        List<GameObject> wallsOut = new List<GameObject>();
        List<CollisionData> ropesIn = new List<CollisionData>();
        List<GameObject> ropesOut = new List<GameObject>();

        public Vector3 MovementDirection { get; private set; }
        public Vector3 TrueMovementDirection =>
Helper.TransformDirection(MovementDirection);

        Vector3 prevPosition;
        CurvySpline ropeSpline;
        Transform ropeStart;
        Transform ropeEnd;
        Vector3 ropeDirection;

        public bool Grounded { get; private set; }
        bool wasGrounded;

        public bool Climbing { get; private set; }
        bool wasClimbing;

        public bool WalkingOnRope { get; private set; }
        bool wasWalkingOnRope;

        new Rigidbody rigidbody;
        public Rigidbody Rigidbody => rigidbody ?? (rigidbody =
GetComponent<Rigidbody>());

        new CapsuleCollider collider;
        public CapsuleCollider CapsuleCollider => collider ?? (collider =
GetComponent<CapsuleCollider>());

        SplineController splineController;
        public SplineController SplineController => splineController ??
(splineController = GetComponent<SplineController>());

        Transform helper;
        public Transform Helper
        {
            get
```

```
            {
                if (helper == null)
                {
                    helper = new GameObject("helper").transform;
                    helper.SetParent(transform);
                    helper.localPosition = Vector3.zero;
                }
                return helper;
            }
        }

        public event Action OnJump;

        public float MaxSpeed => moveSpeed;

        public void ClearCollisions()
        {
            grounds.Clear();
            walls.Clear();
            ropes.Clear();

            collisions.Clear();
        }


        public void SetMovementDirection(Vector3 movementDirection)
        {
            if (!IsAvailable) return;

            if (!canMove)
                movementDirection = Vector3.zero;

            MovementDirection = movementDirection;
        }

        public async Task SetDestination(Vector3 target, Action callback =
null)
        {
            while (IsAvailable && Vector3.Distance(target,
transform.position) > .1f)
            {
                moveSpeed = MaxSpeed;
                SetMovementDirection(target - transform.position);
                await Task.Yield();
            }

            if (IsAvailable)
                callback?.Invoke();
        }
```

```
        #region Movement
        void FixedUpdate()
        {
            CheckCollisions();

            Vector3 normal = GetNormal();
            Helper.up = normal;

            if (Grounded || Climbing)
                Rigidbody.velocity = Vector3.zero;

            Rigidbody.useGravity = !Climbing && !WalkingOnRope;

            if (canMove)
            {
                if (WalkingOnRope && splineController.enabled)
                    ControlWalkOnRope();
                else
                    transform.position += TrueMovementDirection *
(moveSpeed * Time.fixedDeltaTime);
            }

            // Jump
            if (canJump && wasGrounded && !Grounded && !Climbing &&
!wasClimbing && !WalkingOnRope && !wasWalkingOnRope)
            {
                Rigidbody.velocity += Vector3.up * jumpForce;
                OnJump?.Invoke();
            }

            // When stopping climbing, cancel out y velocity
            if (wasClimbing && !Climbing)
                Rigidbody.velocity = new Vector3(Rigidbody.velocity.x, 0f,
Rigidbody.velocity.z);

            // When climbing, force contact with surface
            if (canMove && Climbing && !Grounded && !WalkingOnRope)
                StickToSurface(normal);

            prevPosition = transform.position;
        }

        void CheckCollisions()
        {
            collisions.Clear();

            // grounds
            foreach (GameObject ground in groundsOut)
                grounds.Remove(ground);
```

```
            foreach (CollisionData ground in groundsIn)
            {
                if (!grounds.Contains(ground.gameObject))
                    grounds.Add(ground.gameObject);

                collisions.Add(ground);
            }

            wasGrounded = Grounded;
            Grounded = grounds.Count > 0;

            // ropes
            foreach (GameObject rope in ropesOut)
                ropes.Remove(rope);

            foreach (CollisionData rope in ropesIn)
            {
                if (!splineController.enabled && !WalkingOnRope)
                    StartWalkingOnRope(rope.gameObject);

                if (!ropes.Contains(rope.gameObject))
                    ropes.Add(rope.gameObject);

                collisions.Add(rope);
            }

            wasWalkingOnRope = WalkingOnRope;
            WalkingOnRope = ropes.Count > 0;

            // walls
            foreach (GameObject wall in wallsOut)
                walls.Remove(wall);

            foreach (CollisionData wall in wallsIn)
            {
                if (wall.center.y > transform.position.y +
CapsuleCollider.radius)
                {
                    if (!walls.Contains(wall.gameObject))
                        walls.Add(wall.gameObject);

                    collisions.Add(wall);
                }
            }

            wasClimbing = Climbing;
            Climbing = walls.Count > 0;

            // clear
            groundsIn.Clear();
```

```
            groundsOut.Clear();
            wallsIn.Clear();
            wallsOut.Clear();
            ropesIn.Clear();
            ropesOut.Clear();
        }

        void StartWalkingOnRope(GameObject collisionGameObject)
        {
            ropeSpline =
collisionGameObject.GetComponentInParent<CurvySpline>();
            float nearestPointTF =
ropeSpline.GetNearestPointTF(ropeSpline.transform.InverseTransformPoint(t
ransform.position));

            if (nearestPointTF == 0f)
                nearestPointTF = .001f;
            else if (nearestPointTF == 1f)
                nearestPointTF = .999f;

            splineController.RelativePosition = nearestPointTF;

            ropeStart = ropeSpline.FirstVisibleControlPoint.transform;
            ropeEnd = ropeSpline.LastVisibleControlPoint.transform;

            ropeDirection =
ropeSpline.LastVisibleControlPoint.transform.position -
ropeSpline.FirstVisibleControlPoint.transform.position;
            ropeDirection.y = 0f;
            ropeDirection.Normalize();

            splineController.Spline = ropeSpline;
            splineController.enabled = true;
        }

        public void OnSplineEndReached(CurvySplineMoveEventArgs args)
        {
            splineController.enabled = false;
        }

        void ControlWalkOnRope()
        {
            float dot = Vector3.Dot(ropeDirection, MovementDirection);

            splineController.MovementDirection = dot > 0 ?
FluffyUnderware.Curvy.Controllers.MovementDirection.Forward :
FluffyUnderware.Curvy.Controllers.MovementDirection.Backward;

            if (dot < 0f)
                dot *= -1;
```

```
            splineController.Speed = moveSpeed * dot;
        }

        public Vector3 GetNormal()
        {
            if (collisions.Count == 0)
                return Vector3.up;

            Vector3 normal = Vector3.zero;

            foreach (CollisionData collision in collisions)
                normal += collision.normal;

            return (normal / collisions.Count).normalized;
        }

        void StickToSurface(Vector3 normal)
        {
            Vector3 center =
transform.TransformPoint(CapsuleCollider.center) + normal * .5f;
            Vector3 point1 = center + Vector3.down *
(CapsuleCollider.height - CapsuleCollider.radius);
            Vector3 point2 = point1 + Vector3.up * CapsuleCollider.height;

            Ray ray = new Ray(center, -normal);

            if (Physics.CapsuleCast(point1, point2,
CapsuleCollider.radius, -normal, out RaycastHit hitInfo, 10f,
climbableLayers))
                transform.position = ray.GetPoint(hitInfo.distance) -
CapsuleCollider.center;
        }
        #endregion

        #region Orientation
        void LateUpdate()
        {
            if (MovementDirection != Vector3.zero)
            {
                Vector3 forward;

                if (Grounded || !Climbing)
                    forward = MovementDirection;
                else
                {
                    forward = -GetNormal();
                    forward.y = 0f;
                }
```

```
        if (forward != Vector3.zero)
            transform.forward = forward;
    }
}
#endregion

#region Collisions
void OnCollisionEnter(Collision collision)
{
    AddCollisionIn(collision);
}

void OnCollisionStay(Collision collision)
{
    AddCollisionIn(collision);
}

void OnCollisionExit(Collision collision)
{
    AddCollisionOut(collision);
}

void AddCollisionIn(Collision collision)
{
    int layer = collision.gameObject.layer;

    if (walkableLayers.Contains(layer))
        groundsIn.Add(new CollisionData(collision));
    else if (climbableLayers.Contains(layer))
        wallsIn.Add(new CollisionData(collision));
    else if (ropeLayers.Contains(layer))
        ropesIn.Add(new CollisionData(collision));
}

void AddCollisionOut(Collision collision)
{
    int layer = collision.gameObject.layer;

    if (walkableLayers.Contains(layer))
        groundsOut.Add(collision.gameObject);
    else if (climbableLayers.Contains(layer))
        wallsOut.Add(collision.gameObject);
    else if (ropeLayers.Contains(layer))
        ropesOut.Add(collision.gameObject);
}
#endregion

void OnDrawGizmos()
{
    if (Application.isPlaying)
```

```
            {
                Gizmos.color = Color.blue;
                Gizmos.DrawLine(transform.position, transform.position +
TrueMovementDirection * moveSpeed);

                Gizmos.color = Color.green;
                Gizmos.DrawLine(transform.position, transform.position +
GetNormal() * moveSpeed);
            }
        }
    }

}using MoreMountains.Feedbacks;
using RayFire;
using UnityEngine;

namespace PlantTheBomb.Demolition
{
  public class DemolishBomb : MonoBehaviour
  {
     [SerializeField] private MMFeedbacks _explosionFeedbacks;
     private RayfireBomb _rayfireBomb;

     private void Awake()
     {
        _rayfireBomb = GetComponent<RayfireBomb>();
     }

     public void Explode()
     {
        _rayfireBomb.Explode(0);
        _explosionFeedbacks.PlayFeedbacks();
     }
  }
}using System.Threading.Tasks;
using SuperTools.CameraManager;
using MoreMountains.NiceVibrations;
using PlantTheBomb.Cameras;
using PlantTheBomb.Characters.Player.Hand;
using RayFire;
using Sirenix.OdinInspector;
using UnityEngine;
using Random = UnityEngine.Random;

namespace PlantTheBomb.Demolition
{
  public class DemolitionByBombs : MonoBehaviour
  {
     [SerializeField] private RayfireRigid[] _targets;
     [SerializeField] private DemolishBomb[] _bombs;
```

```
        [SerializeField] private RayfireRigid[] _partsRb;

        private PlayerHandController _player;

      private bool _isAvailable = true;

      private void Start()
      {
         _player = FindObjectOfType<PlayerHandController>();
         _player.Model.OnDetonate += Demolish;
      }

      private void OnDestroy()
      {
         _player.Model.OnDetonate -= Demolish;
      }

      [Button]
      public async void Demolish()
      {
         if (_targets.Length == 0 || !_isAvailable) return;

         _isAvailable = false;

         _targets.Initialize();

         await new WaitForSeconds(.4f); // delay in animation before hand
press the button

         await Explode();

         Dispatcher.Instance.SuccessLevel();
      }

      private async Task Explode()
      {
            foreach (var rigid in _partsRb)
            {
                rigid.Activate();
            }

            foreach (var target in _targets)
            {
                target.Activate();
            }

         for (var i = 0; i != _bombs.Length; ++i)
         {
            _bombs[i].Explode();
            MMVibrationManager.Haptic(HapticTypes.HeavyImpact);
```

```
            await new WaitForSeconds(Random.Range(.1f, .2f));
        }

        var fpsCamera = 
CameraManager.Instance.GetManagedCamera("FPSCamera") as FPSCamera;
        fpsCamera.ShakeExplosion();
        }
    }
}using System.Collections.Generic;
using UnityEditor;
using UnityEngine;
using UnityEngine.Rendering;

namespace PlantTheBomb.Demolition
{
    public class MeshCombineWizard : ScriptableWizard
    {
        public GameObject combineParent;
        public bool is32bit = true;

        [MenuItem("E.S. Tools/Mesh Combine Wizard")]
        static void CreateWizard()
        {
            var wizard = DisplayWizard<MeshCombineWizard>("Mesh Combine 
Wizard");

            // If there is selection, and the selection of one Scene 
object, auto-assign it
            var selectionObjects = Selection.objects;

            if (selectionObjects != null && selectionObjects.Length == 1)
            {
                var firstSelection = selectionObjects[0] as GameObject;

                if (firstSelection != null)
                {
                    wizard.combineParent = firstSelection;
                }
            }
        }

        void OnWizardCreate()
        {
            // Verify there is existing object root, ptherwise bail.
            if (combineParent == null)
            {
                Debug.LogError("Mesh Combine Wizard: Parent of objects to 
combne not assigned. Operation cancelled.");

                return;
```

```
            }

            // Remember the original position of the object.
            // For the operation to work, the position must be temporarily
set to (0,0,0).
            Vector3 originalPosition = combineParent.transform.position;
            combineParent.transform.position = Vector3.zero;

            // Locals
            Dictionary<Material, List<MeshFilter>>
materialToMeshFilterList =
                new Dictionary<Material, List<MeshFilter>>();
            List<GameObject> combinedObjects = new List<GameObject>();

            MeshFilter[] meshFilters =
combineParent.GetComponentsInChildren<MeshFilter>();

            // Go through all mesh filters and establish the mapping
between the materials and all mesh filters using it.
            foreach (var meshFilter in meshFilters)
            {
                var meshRenderer =
meshFilter.GetComponent<MeshRenderer>();

                if (meshRenderer == null)
                {
                    Debug.LogWarning("The Mesh Filter on object " +
meshFilter.name +
                                     " has no Mesh Renderer component
attached. Skipping.");

                    continue;
                }

                var materials = meshRenderer.sharedMaterials;

                if (materials == null)
                {
                    Debug.LogWarning("The Mesh Renderer on object " +
meshFilter.name +
                                     " has no material assigned.
Skipping.");

                    continue;
                }

                // If there are multiple materials on a single mesh,
cancel.
                if (materials.Length > 1)
                {
```

```
                    // Rollback: return the object to original position
                    combineParent.transform.position = originalPosition;

                    Debug.LogError(
                        "Objects with multiple materials on the same mesh
are not supported. Create multiple meshes from this object's sub-meshes
in an external 3D tool and assign separate materials to each. Operation
cancelled.");

                    return;
                }

                var material = materials[0];

                // Add material to mesh filter mapping to dictionary
                if (materialToMeshFilterList.ContainsKey(material))
materialToMeshFilterList[material].Add(meshFilter);
                else materialToMeshFilterList.Add(material, new
List<MeshFilter>() {meshFilter});
            }

            // For each material, create a new merged object, in the scene
and in the assets folder.
            foreach (var entry in materialToMeshFilterList)
            {
                List<MeshFilter> meshesWithSameMaterial = entry.Value;
                // Create a convenient material name
                string materialName = entry.Key.ToString().Split(' ')[0];

                CombineInstance[] combine = new
CombineInstance[meshesWithSameMaterial.Count];

                for (int i = 0; i < meshesWithSameMaterial.Count; i++)
                {
                    combine[i].mesh =
meshesWithSameMaterial[i].sharedMesh;
                    combine[i].transform =
meshesWithSameMaterial[i].transform.localToWorldMatrix;
                }

                // Create a new mesh using the combined properties
                var format = is32bit ? IndexFormat.UInt32 :
IndexFormat.UInt16;
                Mesh combinedMesh = new Mesh {indexFormat = format};
                combinedMesh.CombineMeshes(combine);

                // Create asset
                materialName += "_" + combinedMesh.GetInstanceID();
                AssetDatabase.CreateAsset(combinedMesh,
"Assets/CombinedMeshes_" + materialName + ".asset");
```

```
                // Create game object
                string goName = (materialToMeshFilterList.Count > 1)
                    ? "CombinedMeshes_" + materialName
                    : "CombinedMeshes_" + combineParent.name;
                GameObject combinedObject = new GameObject(goName);
                var filter = combinedObject.AddComponent<MeshFilter>();
                filter.sharedMesh = combinedMesh;
                var renderer =
combinedObject.AddComponent<MeshRenderer>();
                renderer.sharedMaterial = entry.Key;
                combinedObjects.Add(combinedObject);
            }

            // If there were more than one material, and thus multiple GOs
created, parent them and work with result
            GameObject resultGO = null;

            if (combinedObjects.Count > 1)
            {
                resultGO = new GameObject("CombinedMeshes_" +
combineParent.name);
                foreach (var combinedObject in combinedObjects)
combinedObject.transform.parent = resultGO.transform;
            }
            else
            {
                resultGO = combinedObjects[0];
            }

            // Create prefab
            Object prefab = PrefabUtility.CreateEmptyPrefab("Assets/" +
resultGO.name + ".prefab");
            PrefabUtility.ReplacePrefab(resultGO, prefab,
ReplacePrefabOptions.ConnectToPrefab);

            // Disable the original and return both to original positions
            combineParent.SetActive(false);
            combineParent.transform.position = originalPosition;
            resultGO.transform.position = originalPosition;
        }
    }
}using SuperTools.Menus;
using PlantTheBomb.Characters.Player.Hand;
using PlantTheBomb.Demolition;
using PlantTheBomb.LevelSystem;
using UnityEngine;
using UnityEngine.SceneManagement;

namespace PlantTheBomb.GameStates
```

```
{
  public class BuildingDemolitionState
  {
     private readonly LevelModel _levelModel;

     private PlayerHandController _player;
     private DemolitionByBombs _demolitionByBombs;

     public BuildingDemolitionState()
     {
        _levelModel = GameObject.FindObjectOfType<LevelModel>();
     }

     public async void PerformTransition()
     {
        await
LevelsLoader.Instance.LoadScene(_levelModel.LevelData.BuildingDemolitionS
cene, LoadSceneMode.Additive);
        LevelsLoader.Instance.UnloadScene(_levelModel.LevelData.Level,
UnloadSceneOptions.None);
        MenuManager.Instance.HideAllMenusImmediately();
        MenuManager.Instance.MenuList.DemolitionMenu.Show();
     }
  }
}using Sirenix.OdinInspector;
using UnityEngine;

namespace PlantTheBomb.GlobalDebug
{
  public class GlobalDebug : MonoBehaviour
  {
     [Button]
     public void Success()
     {
        Dispatcher.Instance.SuccessLevel();
     }

     [Button]
     public void Fail()
     {
        Dispatcher.Instance.FailLevel();
     }
  }
}using CoolTools.Colliders;
using MoreMountains.Feedbacks;
using PlantTheBomb.Characters.Player;
using PlantTheBomb.LevelObjects.Helicopter;
using UnityEngine;

namespace PlantTheBomb.LevelObjects
```

```
{
  public class LevelEndPoint : MonoBehaviour
  {
     [SerializeField] private MMFeedbacks _availableFeedback;
     [SerializeField] private HelicopterView _helicopterView;

     private TriggerSource _triggerSource;

     private bool _isAvailable;

     private void Awake()
     {
        _triggerSource = GetComponentInChildren<TriggerSource>();
     }

     private void OnEnable()
     {
        _triggerSource.OnEnter += OnEnter;
     }

     private void OnDisable()
     {
        _triggerSource.OnEnter -= OnEnter;
     }

     private async void OnEnter(Collider other)
     {
        if (!_isAvailable) return;

        var player =
other.GetComponentInParent<PlayerCharacterController>();

        if (player)
        {
           player.Model.Input.IsAvailable = false;

           await player.Model.UnitModel.Movement

.SetDestination(_helicopterView.PlayerJumpPos.position);

           if (player.Model.HealthComponent.IsDead) return;

           Destroy(player.gameObject);

           _helicopterView.PlayFly();

           await new
WaitForSeconds(_helicopterView.HelicopterFlyDuration);

           Dispatcher.Instance.StartBuildingDemolitionState();
```

```
        }
      }

      public void SetAvailable(bool value)
      {
         _isAvailable = value;
         _availableFeedback.PlayFeedbacks();
      }
   }
}using SuperTools.CameraManager;
using UnityEngine;

namespace PlantTheBomb.LevelObjects.Helicopter
{
   public class HelicopterView : MonoBehaviour
   {
      [field: SerializeField] public float HelicopterFlyDuration { get;
private set; }
      [field: SerializeField] public Transform PlayerJumpPos { get;
private set; }

      [SerializeField] private Transform _helicopter;
      [SerializeField] private GameObject _helicopterIdle;
      [SerializeField] private GameObject _helicopterFly;

      public void PlayFly()
      {
         _helicopterIdle.SetActive(false);
         _helicopterFly.SetActive(true);

         var helicopterCamera =
CameraManager.Instance.GetManagedCamera("HelicopterCamera");
         helicopterCamera.CinemachineVirtualCameraBase.Follow =
_helicopter;
         helicopterCamera.CinemachineVirtualCameraBase.LookAt =
_helicopter;
         helicopterCamera.Transition();
      }
   }
}using System;
using UnityEditor;
using UnityEngine;

namespace PlantTheBomb.LevelSystem
{
   [CreateAssetMenu]
   public class LevelData : ScriptableObject
   {
      [field: SerializeField] public bool PlayOnce { get; private set; }
```

```
     [field: SerializeField] public SceneReference Level { get; private
set; }
     [field: SerializeField] public SceneReference
BuildingDemolitionScene { get; private set; }

#if UNITY_EDITOR
     public string GetLevelName()
     {
        var result = "";

        var path = AssetDatabase.GUIDToAssetPath(Level.guid);

        var startIndex = path.IndexOf("Level", StringComparison.Ordinal);
        var endIndex = path.LastIndexOf(".unity",
StringComparison.Ordinal);

        if (startIndex >= 0 && endIndex >= 0)
           result = path.Substring(startIndex, endIndex - startIndex);

        return result;
     }
#endif
  }
}using PlantTheBomb.Characters.Spawners;
using UnityEngine;

namespace PlantTheBomb.LevelSystem
{
  public class LevelModel : MonoBehaviour
  {
     [field: SerializeField] public CharacterSpawner PlayerSpawner { get;
private set; }
     public LevelData LevelData { get; set; }

     public LevelProgression LevelProgression { get; private set; }

     public void Init()
     {
        LevelProgression = GetComponent<LevelProgression>();
        LevelProgression.Init();
     }
  }
}using PlantTheBomb.Bombs;
using PlantTheBomb.LevelObjects;
using UnityEngine;

namespace PlantTheBomb.LevelSystem
{
  public class LevelProgression : MonoBehaviour
  {
```

```
        public int AmountToPlant { get; private set; }

        private PointToPlant[] _pointsToPlant;
        private LevelEndPoint _levelEndPoint;

        public void Init()
        {
            InitPointsToPlant();

            _levelEndPoint = GetComponentInChildren<LevelEndPoint>();
        }

        private void InitPointsToPlant()
        {
            _pointsToPlant = GetComponentsInChildren<PointToPlant>();
            AmountToPlant = _pointsToPlant.Length;

            for (var i = 0; i != _pointsToPlant.Length; ++i)
            {
                _pointsToPlant[i].onPlanted += () =>
                {
                    AmountToPlant--;
                    TrySetLevelEndAvailable();
                };
            }
        }

        private void TrySetLevelEndAvailable()
        {
            if (AmountToPlant > 0) return;

            _levelEndPoint.SetAvailable(true);
        }
    }
}using System.Collections;
using System.Threading.Tasks;
using SuperTools.Core.DesignPatterns;
using PlantTheBomb.SaveSystem;
using Sirenix.OdinInspector;
using UnityEngine;
using UnityEngine.SceneManagement;

namespace PlantTheBomb.LevelSystem
{
    [Singleton(SingletonAttribute.DontDestroyOnLoadMode.Disable,
SingletonAttribute.MultipleInstanceMode.Replace,
SingletonAttribute.AutoCreateMode.Disable,
SingletonAttribute.DestroyMode.GameObject)]
    [ExecuteInEditMode]
    public class LevelsLoader : Singleton<LevelsLoader>
```

```
  {
     [SerializeField]
     private LevelsSequence _levelsSequence;

     #region EDITOR

#if UNITY_EDITOR

     [ValueDropdown(nameof(GetAllLevelsFromSequence)),
OnValueChanged(nameof(UpdateCurrentLevel)), SerializeField]
     private string _currentLevel;

     [SerializeField] private bool _loopCurrentLevel;

     private const string LevelInspectorProperty_PlayerPrefs =
"LevelInspectorProperty";

     protected override void Awake()
     {
        base.Awake();


UpdateCurrentLevelInspectorProperty(PlayerPrefs.GetInt(LevelInspectorProp
erty_PlayerPrefs));
     }

     private IEnumerable GetAllLevelsFromSequence()
     {
        for (var i = 0; i != _levelsSequence.Levels.Length; ++i)
        {
           yield return _levelsSequence.Levels[i].GetLevelName();
        }
     }

     private void UpdateCurrentLevel()
     {
        for (var i = 0; i != _levelsSequence.Levels.Length; ++i)
        {
           if (_levelsSequence.Levels[i].GetLevelName() == _currentLevel)
           {
              Save.data.level = i;
              Save.Store();

              UpdateCurrentLevelInspectorProperty(i);

              if (Application.isPlaying)
                 MoveToNextLevel();

              return;
           }
```

```
        }
      }

      private void UpdateCurrentLevelInspectorProperty(int saveLevelId)
      {
         var realLevelId = saveLevelId % _levelsSequence.Levels.Length;

         PlayerPrefs.SetInt(LevelInspectorProperty_PlayerPrefs,
realLevelId);

         _currentLevel =
_levelsSequence.Levels[realLevelId].GetLevelName();
      }

#endif

      #endregion

      public async Task LoadScene(SceneReference sceneReference,
LoadSceneMode loadSceneMode)
      {
         await SceneManager.LoadSceneAsync(sceneReference.sceneIndex,
loadSceneMode);
      }

      public async Task<LevelModel> LoadCurrentLevelScene(LoadSceneMode
loadSceneMode)
      {
         var levelIDToLoad = Save.data.level;
         var realLevelId = levelIDToLoad  % _levelsSequence.Levels.Length;

         while (levelIDToLoad > realLevelId &&
_levelsSequence.Levels[realLevelId].PlayOnce)
         {
            levelIDToLoad++;
            realLevelId = levelIDToLoad  % _levelsSequence.Levels.Length;
         }

         var currentLevelData = _levelsSequence.Levels[realLevelId];

         await
SceneManager.LoadSceneAsync(currentLevelData.Level.sceneIndex,
loadSceneMode);

         var levelModel = FindObjectOfType<LevelModel>();

         levelModel.LevelData = currentLevelData;
         levelModel.Init();

         return levelModel;
```

```
    }

    public void UnloadScene(SceneReference sceneReference,
UnloadSceneOptions options)
    {
        SceneManager.UnloadSceneAsync(sceneReference.sceneIndex,
options);
    }

    [Button("Reload")]
    public void MoveToNextLevel()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
    }

    public void IncreaseLevel()
    {
#if UNITY_EDITOR
        if (_loopCurrentLevel) return;
#endif

        var currentLevel = Save.data.level;
        currentLevel++;
        Save.data.level = currentLevel;
        Save.Store();

#if UNITY_EDITOR
        UpdateCurrentLevelInspectorProperty(Save.data.level);
#endif
    }
  }
}using Sirenix.OdinInspector;
using UnityEngine;

namespace PlantTheBomb.LevelSystem
{
  [CreateAssetMenu]
  public class LevelsSequence : ScriptableObject
  {
    [InlineEditor] [field: SerializeField]
    public LevelData[] Levels { get; private set; }
  }
}using UnityEngine;
using System;
using System.Collections;
using System.Collections.Generic;

namespace Menus
{
  public class DemolitionMenu : SuperTools.Menus.Menu
```

```
  {
     #region Properties

     #endregion

     #region Menu Methods

     // Call at the start of the show animation
     protected override void OnShowStart()
     {
        base.OnShowStart();
     }

     // Call at the end of the show animation
     protected override void OnShowEnd()
     {
        base.OnShowEnd();
     }

     // Call at the start of the hide animation
     protected override void OnHideStart()
     {
        base.OnHideStart();
     }

     // Call at the end of the hide animation
     protected override void OnHideEnd()
     {
        base.OnHideEnd();
     }

     #endregion
  }
}
namespace SuperTools.Menus
{
  public partial class MenuList
  {
     [SerializeField]
     private global::Menus.DemolitionMenu demolitionMenu;
     public global::Menus.DemolitionMenu DemolitionMenu =>
demolitionMenu;
  }
}
using System;
using SuperTools.Menus;
using Menus;
using MoreMountains.NiceVibrations;
using PlantTheBomb.LevelSystem;
using PlantTheBomb.SaveSystem;
```

```
using TMPro;
using UnityEngine;
using UnityEngine.UI;

namespace Menus
{
  public class FailMenu : Menu
  {
     #region Properties

     [SerializeField] private TMP_Text _levelNumText;
     [SerializeField] private float _delayBeforeShow;
     [SerializeField] private Button _continueButton;

     public event Action showWindow;

     #endregion

     #region Menu Methods

     // Call at the start of the show animation
     protected override void OnShowStart()
     {
        base.OnShowStart();
     }

     // Call at the end of the show animation
     protected override void OnShowEnd()
     {
        base.OnShowEnd();
     }

     // Call at the start of the hide animation
     protected override void OnHideStart()
     {
        base.OnHideStart();
     }

     // Call at the end of the hide animation
     protected override void OnHideEnd()
     {
        base.OnHideEnd();
     }

     #endregion

     private void OnEnable()
     {
        _levelNumText.text = $"Level {Save.data.level + 1} Failed";
        _continueButton.onClick.AddListener(ClickToContinue);
```

```
        }

        private void OnDisable()
        {
            _continueButton.onClick.RemoveListener(ClickToContinue);
        }

        private void ClickToContinue()
        {
            LevelsLoader.Instance.MoveToNextLevel();
        }

        public async void ShowWithDelay()
        {
            await new WaitForSeconds(_delayBeforeShow);

            MenuManager.Instance.HideAllMenusImmediately();

            showWindow?.Invoke();

            MMVibrationManager.Haptic(HapticTypes.Failure);

            Show();
        }
    }
}
namespace SuperTools.Menus
{
    public partial class MenuList
    {
        [SerializeField]
        private FailMenu failMenu;
        public FailMenu FailMenu => failMenu;
    }
}
using SuperTools.Menus;
using Menus;
using PlantTheBomb.SaveSystem;
using TMPro;
using UnityEngine;

namespace Menus
{
    public class GameMenu : Menu
    {
        #region Properties

        [SerializeField] private TMP_Text _levelNumLabel;
        [SerializeField] private TMP_Text _bombsAmountLabel;
        [SerializeField] private Animator _hitFeedbackAnim;
```

```
        private static readonly int HitHash = Animator.StringToHash("Hit");

        #endregion

        #region Menu Methods

        // Call at the start of the show animation
        protected override void OnShowStart()
        {
            base.OnShowStart();
        }

        // Call at the end of the show animation
        protected override void OnShowEnd()
        {
            base.OnShowEnd();
        }

        // Call at the start of the hide animation
        protected override void OnHideStart()
        {
            base.OnHideStart();
        }

        // Call at the end of the hide animation
        protected override void OnHideEnd()
        {
            base.OnHideEnd();
        }

        #endregion

        private void OnEnable()
        {
            _levelNumLabel.text = $"Level {Save.data.level + 1}";
        }

        public void DisplayBombsAmount(int currentAmount, int totalAmount)
        {
            _bombsAmountLabel.text = $"{currentAmount}/{totalAmount}";
        }

        public void PlayPlayerHit()
        {
            _hitFeedbackAnim.SetTrigger(HitHash);
        }
    }
}
namespace SuperTools.Menus
{
```

```
  public partial class MenuList
  {
     [SerializeField]
     private GameMenu gameMenu;
     public GameMenu GameMenu => gameMenu;
  }
}
using System;
using Sirenix.OdinInspector;
using UnityEngine;
using UnityEngine.UI;

namespace Super.UI
{
  [RequireComponent(typeof(CanvasGroup))]
  public class HealthBar : MonoBehaviour
  {
     [ShowInInspector] public Slider _slider;
     [SerializeField] private bool _playHitView;
     [SerializeField] private bool _hideOnStart;

     private RectTransform _rectTransform;
     private CanvasGroup _canvasGroup;

     private Animator _anim;
     private static readonly int Hit = Animator.StringToHash("Hit");

     private void Awake()
     {
        _rectTransform = GetComponent<RectTransform>();
        _canvasGroup = GetComponent<CanvasGroup>();
        _anim = GetComponent<Animator>();
     }

     private void Start()
     {
        GetComponent<Canvas>().worldCamera = Camera.main;

        if (_hideOnStart)
           _canvasGroup.alpha = 0;
     }

     private void LateUpdate()
     {
        transform.LookAt(Camera.main.transform);
        transform.Rotate(0, 180, 0);
     }

     public void Display(float value)
     {
```

```
            _canvasGroup.alpha = 1;

            _slider.value = value;

            if (_playHitView)
                PlayHitView();
        }

        private void PlayHitView()
        {
            if (!_anim) return;

            _anim.SetTrigger(Hit);
        }

        public void Hide()
        {
            gameObject.SetActive(false);
        }
    }
}using SuperTools.Menus;
using Menus;
using PlantTheBomb;
using PlantTheBomb.SaveSystem;
using TMPro;
using UnityEngine;
using UnityEngine.UI;

namespace Menus
{
    public class MainMenu : Menu
    {
        #region Properties

        [SerializeField] private TMP_Text _levelNumText;
        [SerializeField] private Button _startGameButton;

        #endregion

        #region Menu Methods

        // Call at the start of the show animation
        protected override void OnShowStart()
        {
            base.OnShowStart();
        }

        // Call at the end of the show animation
        protected override void OnShowEnd()
        {
```

```
        base.OnShowEnd();
      }

      // Call at the start of the hide animation
      protected override void OnHideStart()
      {
        base.OnHideStart();
      }

      // Call at the end of the hide animation
      protected override void OnHideEnd()
      {
        base.OnHideEnd();
      }

      #endregion

      private void OnEnable()
      {
        _levelNumText.text = $"Level {Save.data.level + 1}";

_startGameButton.onClick.AddListener(Dispatcher.Instance.StartPlayLevel);
      }

      private void OnDisable()
      {
        _startGameButton.onClick.RemoveAllListeners();
      }
  }
}
namespace SuperTools.Menus
{
  public partial class MenuList
  {
      [SerializeField]
      private MainMenu mainMenu;
      public MainMenu MainMenu => mainMenu;
  }
}
using System;
using System.Threading.Tasks;
using CoolTools.UI;
using SuperTools.PoolParty;
using UnityEngine;
using UnityEngine.UI;

namespace Super.UI
{
  public class PlantingUIMarker : MonoBehaviour
  {
```

```
      [SerializeField] private float _yOffset;
      [SerializeField] private Slider _slider;

      private Vector3 _worldPos;

      public void Init(Vector3 worldPos)
      {
         _worldPos = worldPos;

         _slider.value = 0;
      }

      public void Display(float value)
      {
         _slider.value = value;
      }

      public void Hide()
      {
         PoolParty.Destroy(gameObject);
      }

      private void LateUpdate()
      {
         transform.position = Camera.main.WorldToScreenPoint(_worldPos) +
new Vector3(0, _yOffset, 0);
         transform.up = Vector3.up;
      }
   }
}using PlantTheBomb;
using UnityEngine;
using UnityEngine.EventSystems;

namespace Super.UI
{
   public class StartGameButton : MonoBehaviour, IPointerDownHandler
   {
      public void OnPointerDown (PointerEventData eventData)
      {
         Dispatcher.Instance.StartPlayLevel();
      }
   }
}using System;
using System.Threading;
using SuperTools.Menus;
using Menus;
using MoreMountains.NiceVibrations;
using PlantTheBomb.LevelSystem;
using PlantTheBomb.SaveSystem;
using TMPro;
```

```
using UnityEngine;
using UnityEngine.UI;

namespace Menus
{
  public class SuccessMenu : Menu
  {
     #region Properties

     [SerializeField] private TMP_Text _levelNumText;
     [SerializeField] private float _delayBeforeShow;
     [SerializeField] private Button _continueButton;

     public event Action showWindow;

     #endregion

     #region Menu Methods

     // Call at the start of the show animation
     protected override void OnShowStart()
     {
        base.OnShowStart();
     }

     // Call at the end of the show animation
     protected override void OnShowEnd()
     {
        base.OnShowEnd();
     }

     // Call at the start of the hide animation
     protected override void OnHideStart()
     {
        base.OnHideStart();
     }

     // Call at the end of the hide animation
     protected override void OnHideEnd()
     {
        base.OnHideEnd();
     }

     #endregion

     private void OnEnable()
     {
        _levelNumText.text = $"Level {Save.data.level} Complete";

        _continueButton.onClick.AddListener(ClickToContinue);
```

```
        }

        private void OnDisable()
        {
            _continueButton.onClick.RemoveListener(ClickToContinue);
        }

        private void ClickToContinue()
        {
            LevelsLoader.Instance.MoveToNextLevel();
        }

        public async void ShowWithDelay(CancellationToken token)
        {
            await new WaitForSeconds(_delayBeforeShow);

            if (token.IsCancellationRequested) return;

            MenuManager.Instance.HideAllMenusImmediately();

            showWindow?.Invoke();

            MMVibrationManager.Haptic(HapticTypes.Success);

            Show();
        }
    }
}
namespace SuperTools.Menus
{
    public partial class MenuList
    {
        [SerializeField]
        private SuccessMenu successMenu;
        public SuccessMenu SuccessMenu => successMenu;
    }
}
using System;
using SuperTools.PoolParty;
using UnityEngine;

namespace Super.UI
{
    public class UITargetMarker : MonoBehaviour
    {
        public Transform target;
        public Transform fixedPart;
        public float yWorldOffset = 1.5f;
        public float yScreenOffset = 150;
        public Vector2 margin;
```

```
Vector3 targetPosition;
float minX, maxX, midX;
float minY, maxY, midY;
float xLerp, yLerp;

Camera mainCamera;

public virtual void Init(Transform newTarget)
{
   target = newTarget;

   CalculateScreenSizes();
}

private void CalculateScreenSizes()
{
   minX = margin.x;
   maxX = Screen.width - margin.x;
   midX = (minX + maxX) / 2f;
   minY = margin.y;
   maxY = Screen.height - margin.y;
   midY = (minY + maxY) / 2f;
}

private void OnEnable()
{
   mainCamera = Camera.main;
}

public void Hide()
{
   gameObject.SetActive(false);
}

private void LateUpdate()
{
   if (target != null)
   {
      CalculateTargetScreenPosition();

      if (targetPosition.x < minX)
         xLerp = Mathf.InverseLerp(midX, targetPosition.x, minX);
      else if (targetPosition.x > maxX)
         xLerp = Mathf.InverseLerp(midX, targetPosition.x, maxX);
      else
         xLerp = 0f;

      if (targetPosition.y < minY)
         yLerp = Mathf.InverseLerp(midY, targetPosition.y, minY);
```

```
            else if (targetPosition.y > maxY)
               yLerp = Mathf.InverseLerp(midY, targetPosition.y, maxY);
            else
               yLerp = 0f;

            if (InsideScreen())
            {
               MoveInsideScreen();
            }
            else
            {
               MoveOutsideScreen();
            }

            // KeepRotation(fixedPart);
         }
      }

      private void CalculateTargetScreenPosition()
      {
         targetPosition = mainCamera.WorldToScreenPoint(target.position +
Vector3.up * yWorldOffset);

         if (targetPosition.z < 0f)
            targetPosition *= -1;
      }

      private bool InsideScreen() => xLerp == 0f && yLerp == 0f;

      private void MoveInsideScreen()
      {
         transform.position = targetPosition + Vector3.up * yScreenOffset;
         transform.up = Vector3.up;
      }

      private void MoveOutsideScreen()
      {
         if (xLerp > yLerp)
            transform.position = new Vector3(midX + (targetPosition.x -
midX) * xLerp,
                                             midY + (targetPosition.y -
midY) * xLerp,
                                             targetPosition.z);
         else
            transform.position = new Vector3(midX + (targetPosition.x -
midX) * yLerp,
                                             midY + (targetPosition.y -
midY) * yLerp,
                                             targetPosition.z);
```

```
            transform.up = transform.position - targetPosition;
        }

        private void KeepRotation(Transform target)
        {
            target.rotation = Quaternion.identity;
        }
    }
}using UnityEngine;

namespace PlantTheBomb.SaveSystem
{
    [System.Serializable]
    public class Data
    {
        public bool firstAppLaunch = false;
        public int level = 0;
        public int coins = 0;
        public float volume = 0.5f;
    }

    public static class Save
    {
        public static Data data;
        static string keyName = "data";

        // Convert the data into PlayPrefs as XML
        public static void Store()
        {
            PlayerPrefs.SetString(keyName,
Serializer.Serialize<Data>(data));
        }

        // Load the data from the PlayerPrefs and parse it into the data
object
        public static void Load(System.Action whenLoad = null)
        {
            if (PlayerPrefs.HasKey(keyName))
            {
                data =
Serializer.Deserialize<Data>(PlayerPrefs.GetString(keyName));
            }
            else
            {
                data = new Data();
                Store();
            }

            if (whenLoad != null) whenLoad.Invoke();
        }
```

```
        // Delete the save
        public static void Delete()
        {
            DeleteKey(keyName);
        }

        // Delete the given key inside the PlayerPrefs : DEVELOPER ONLY
        public static void DeleteKey(string key)
        {
            if (PlayerPrefs.HasKey(key))
            {
                PlayerPrefs.DeleteKey(key);
            }
        }
    }
}using System.IO;
using System.Xml.Serialization;

namespace PlantTheBomb.SaveSystem
{
    public static class Serializer
    {
        // Serialize <T> any type
        // Convert <T> any type of data into a string
        public static string Serialize<T>(this T toSerialize)
        {
            XmlSerializer xml = new XmlSerializer(typeof(T));
            StringWriter writer = new StringWriter();
            xml.Serialize(writer, toSerialize);

            return writer.ToString();
        }

        // De-serialize
        public static T Deserialize<T>(this string toDeserialize)
        {
            XmlSerializer xml = new XmlSerializer(typeof(T));
            StringReader reader = new StringReader(toDeserialize);

            return (T) xml.Deserialize(reader);
        }
    }
}
```