ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

Пояснювальна записка

до бакалаврської роботи
на ступінь вищої освіти бакалавр

на тему: **«РОЗРОБКА ГРИ В ЖАНРІ «TOWER DEFENSE» З ВИКОРИСТАННЯМ RPG МЕХАНІК МОВОЮ C#»**

Виконав: студент 4 курсу, групи ПД-43
спеціальності
121 Інженерія програмного забезпечення
(шифр та назва спеціальності)

Бойко М.С.
(прізвище та ініціали)

Керівник Дібрівний О.А.
(прізвище та ініціали)

Рецензент_____
(прізвище та ініціали)

Київ - 2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Інженерії програмного забезпечення
Ступінь вищої освіти - «Бакалавр»
Спеціальність підготовки - 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Негоденко О.В.

«____« _____ 2023 року

З А В Д А Н Н Я
НА БАКАВРСЬКУ РОБОТУ СТУДЕНТА

БОЙКО МИКИТА СЕРГІЙОВИЧ
(призвіще, ім'я, по батькові)

1. Тема роботи: «Розробка гри в жанрі «Tower Defense» з використанням RPG механік мовою C#»
Керівник роботи: Дібрівний О.А доктор філософії, доцент кафедри ІПЗ
(прізвище, ім'я, по батькові, науковий ступінь, вчене знання)

Затверджені наказом вищого навчального закладу від «24» лютого 2023 року №26

2. Строк подання студентом роботи «1» червня 2023 року

3. Вхідні дані до роботи:

3.1 Науково-технічна література пов'язана з розробкою ігор на Unity
3.2 Алгоритми дії захисних веж
3.3 Алгоритми дії ворогів

4. Зміст розрахунково - пояснювальної записки (перелік питань, які потрібно розробити)

4.1 Аналіз літератури. Огляд наукових та практичних джерел, які були використані для підготовки проєкту
4.2 Аналіз Інтегрованого середовища розробки Unity для створення ігрової логіки гри
4.3 Аналіз 3D-редактора для створення 3D-моделей Blender

4.4 Розробка програмного забезпечення

5. Перелік демонстраційного матеріалу
   5.1 Мета, об’єкт та предмет дослідження
   5.2 Аналоги
   5.3 Технічне завдання
   5.4 Програмні засоби реалізації
   5.5 Шаблон проєктування «Visitor»
   5.6 Алгоритм роботи веж та ігрового інтерфейсу
   5.7 Алгоритм роботи внутрішньоігрового магазину та щоденної нагороди
   5.8 Алгоритм роботи наслідків виборів під час діалогів гри
   5.9 Оптимізація гри, за допомогою MeshBaker, розробки спільної текстури та засобів Unity

6. Дата видачі завдання «25» лютого 2023 року

Календарний план

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	25.02.23 – 27.02.23	Виконано
2	Аналіз основних концепцій Tower Defense	28.02.23 – 03.03.23	Виконано
3	Розробка структури та компонентів гри	04.03.23 – 27.03.23	Виконано
4	Моделювання об’єктів гри	27.03.23 – 06.04.23	Виконано
5	Створення текстур для ігрових компонентів та розробка дизайну рівнів.	07.04.23 – 26.04.23	Виконано
6	Оптимізування гри під мобільні пристрої.	27.04.23 – 12.05.23	Виконано
7	Тестування та документація гри	13.05.23 – 23.04.23	Виконано
8	Попередній захист	24.05.2023	Виконано
9	Здача роботи	01.06.2023	Виконано

Студент _____ _____
(підпис) (прізвище, ініціали)

Керівник роботи _____ _____
(підпис) (прізвище, ініціали)

РЕФЕРАТ

Текстова частина бакалаврської роботи 56с., 25 рис., 1 табл., 18 джерел.

C#, UNITY, BLENDER, GITHUB, UNITY ADS, GOOGLE PLAY, BLUESTACKS 5

Об'єкт дослідження – геймплей гри жанру «Tower Defense».

Предмет дослідження – гра жанру «Tower Defense» з елементами RPG для мобільних пристроїв.

Мета роботи – підвищення зацікавленості геймплеєм гри в жанрі «Tower Defense» гравців на мобільних пристроях за рахунок впровадження механік «Дерево діалогів» та «Дерево пасивних умінь».

Мета дослідження – методи проєктування та розробка програмного забезпечення, методи створення 3-Д моделей, методи оптимізації програмного забезпечення.

Наукова новизна – Розробка гри жанру «Tower Defense» з використанням міксу механік: механіка дерева пасивних умінь, механіка дерево діалогів.

Галузь використання – ігрова спільнота.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

UI – user interface

GUI – graphical user interface

AI – artificial intelligence

IDE - integrated Development Environments

Mesh - сукупність вершин, ребер та граней, що визначають форму тривимірного об'єкта.

Rigging - процес створення скелетної системи або іншої структури контролю за поведінкою 3D-моделі

Normal map - текстура, яка дозволяє створювати ілюзію рельєфності на поверхнях 3D-моделей

Аддон - додатковий модуль, який можна встановити в Blender для розширення його функціональності.

Ассет - будь-який файл, який може бути використаний для створення ігор

ПЗ – програмне забезпечення

ОС – операційна система

ПК – Персональний комп’ютер

Юніт – мирна або військова одиниця яка використовується у відеоіграх

Вежа – союзна споруда, яка атакує ворогів

Карутина - функція, яка дозволяє виконувати певний код асинхронно

Геймплей - термін, який використовується в ігровій індустрії для опису механіки та механізмів гри, включаючи взаємодію гравця з ігровим світом та об'єктами.

ВСТУП

У світі сьогодні існує велика кількість мобільних ігор різних жанрів, які викликають все більший інтерес у користувачів. Одним з найпопулярніших жанрів є «Tower Defense», який поєднує в собі елементи стратегії та аркади. Для розробки таких ігор використовуються різноманітні інструменти, такі як: мова програмування C#, ігровий двигун Unity, програма для створення 3D-моделей Blender, система контролю версій GitHub, рекламна платформа Unity Ads та магазин додатків Google Play.

Об'єктом дослідження є геймплей гри жанру «Tower Defense».

Предмет дослідження є розроблена гра жанру «Tower Defense» з елементами RPG для мобільних пристроїв.

Метою роботи є підвищення зацікавленості геймплеєм гри в жанрі «Tower Defense» гравців на мобільних пристроях за рахунок впровадження механік «Дерево діалогів» та «Дерево пасивних умінь».

Метою дослідження є методи проєктування та розробка програмного забезпечення, методи створення 3-Д моделей, методи оптимізації програмного забезпечення.

Науковою новизною магістерської роботи є поєднання механік двох різних жанрів ігор, використовуючи механіки з жанру RPG «Дерево пасивних умінь» та «Дерево діалогів» у грі жанру «Tower Defense»

Дослідження передбачає вивчення методів проєктування та розробки програмного забезпечення, методів створення 3D-моделей та методів оптимізації програмного забезпечення. Науковою новизною є розробка гри з використанням міксу механік: механіки дерева пасивних умінь та механіки дерева діалогів. Галуззю використання є ігрова спільнота, яка становить значну частину ринку мобільних ігор.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Мобільні додатки

Мобільні додатки стали невіддільною частиною життя сучасних людей. Вони змінюють спосіб, завдяки якому ми спілкуємось, працюємо, навчаємося, розважаємося і навіть спимо. Нині вони настільки повсюдні, що навіть люди похилого віку можуть дійти до точки, де вони не уявляють свого життя без них.

Додатки дозволяють нам звертатися до інформації та послуг в будь-який час і в будь-якому місці, забезпечуючи зручний та швидкий доступ до потрібної інформації. Це може бути корисним в багатьох аспектах життя, наприклад, в галузі охорони здоров'я, коли люди можуть відстежувати своє фізичне здоров'я та медичні записи, а також отримувати поради щодо здорового способу життя.

Одна з головних переваг програм для мобільних пристроїв полягає у тому, що вони роблять більш доступною інформацію про світ навколо нас. Наприклад, люди можуть швидко дізнатися про погоду, новини, забронювати місце у ресторані або купити квитки на захід, не виходячи з дому або з місця, де вони знаходяться.

Мобільні додатки також допомагають людям в розвитку та навчанні. Наприклад, існує велика кількість додатків для самостійного вивчення мови, знайомства з новими технологіями та навичками, а, також, збільшення своєї продуктивності на роботі або в навчанні.

Деякі мобільні застосунки, зокрема ігри, можуть покращити розумові здібності людини, в тому числі й зосередженість, увагу та реакцію. Дослідження показують, що деякі ігри-головоломки можуть допомогти в тренуванні мозку та покращенні когнітивних функцій.

Наприклад, гра «Тетрис» може допомогти покращити здатність до просторового мислення, а також здатність до управління увагою та зосередження

[1]. Гра «Слова з друзями» може покращити здатність до мовленнєвого спілкування та збагачення словника.

Деякі ігри також можуть допомогти у тренуванні пам'яті та концентрації. Наприклад, гра «Lumosity» містить низку ігор, що допомагають у розвитку різних когнітивних функцій, таких як пам'ять, зосередженість та реакція.

Проте важливо зазначити, що не всі ігри можуть мати позитивний вплив на розумові здібності, тому важливо обирати ігри з розумом та не перебирати їх у шкоду іншим аспектам життя, таким як робота, навчання та соціальні взаємини.

У сучасному світі мобільні додатки відіграють важливу роль в розвитку галузі розробки ігор. Мобільні застосунки дозволяють гравцям насолоджуватися іграми на своїх смартфонах та планшетах, забезпечуючи зручний та доступний спосіб грати улюблені ігри.

Існує багато різних майданчиків, де можна завантажити мобільні додатки на свій смартфон чи планшет. Ось кілька найбільш популярних з них:

1. App Store - офіційний магазин додатків для пристроїв Apple (iOS та iPadOS). Доступний на пристроях з цією операційною системою.
2. Google Play Store - офіційний магазин додатків для пристроїв з операційною системою Android. Доступний на більшості пристроїв з цією операційною системою.
3. Microsoft Store - магазин додатків для пристроїв з операційною системою Windows 10.
4. Amazon Appstore - альтернативний магазин додатків для пристроїв з операційною системою Android та Fire OS.
5. APKMirror - вебсайт, який дозволяє завантажувати APK-файли додатків для Android безпосередньо на свій пристрій.
6. F-Droid - магазин додатків з відкритим вихідним кодом для Android.
7. Apple TestFlight - сервіс для тестування бета-версій додатків на iOS та iPadOS.
8. HockeyApp - сервіс для дистрибуції бета-версій додатків на різних платформах, включаючи iOS, Android, Windows та macOS.

Залежно від операційної системи вашого пристрою, ви можете вибрати найбільш зручний для вас магазин додатків та завантажувати додатки безпосередньо на свій пристрій.

1.2 Мобільні ігри

Мобільні ігри це вид розваг на мобільних пристроях, таких як смартфони та планшети. Вони стали надзвичайно популярними в останні роки, оскільки забезпечують швидкий та легкий доступ до ігор у будь-який час та місце.

Однією з основних переваг мобільних ігор є їх доступність. До них можна легко отримати доступ через мобільний інтернет або Wi-Fi, що дозволяє грати в улюблені ігри в будь-якому місці та в будь-який час. Крім того, мобільні ігри зазвичай не потребують великої кількості часу, що робить їх ідеальними для людей, які шукають швидкий спосіб розважитися.

Другою перевагою мобільних ігор є їх багатство. Існує безліч різних жанрів ігор, від класичних аркад до складних стратегій та рольових ігор. Це означає, що кожен зможе знайти гру на свій смак, що забезпечить незабутні враження та відчуття задоволення.

Однією зі складнощів у розробці мобільних ігор є оптимізація під різні пристрої. Розробники повинні враховувати різні розміри екранів, обсяги пам'яті та характеристики процесорів, щоб забезпечити відмінну продуктивність та відчуття гри на будь-якому пристрої.

Розробники мобільних ігор також повинні розуміти свою аудиторію та її потреби. Наприклад, ігри для дітей мають бути простішими та містити менше насильства, в той час, як ігри для дорослих мають бути складнішими та можуть мати більше насильства. Крім того, вони повинні дбайливо підходити до теми гри, щоб уникнути неприйнятних асоціацій та конфліктів з культурними нормами та цінностями.

З іншого боку, розробники мобільних ігор повинні забезпечувати високу якість графіки та звукового супроводу, щоб зробити гру більш реалістичною та захопливою. Також вони повинні забезпечувати постійне оновлення гри, виправляти помилки та додавати новий контент для того, щоб зберегти інтерес гравців до гри.

Ще однією важливою складовою мобільних ігор є геймплей. У мобільних іграх він зазвичай дуже простий, але це не заважає розважатися та насолоджуватися грою. Розробники мобільних ігор повинні забезпечувати захопливий та динамічний геймплей, який зможе зацікавити гравців на тривалий час. Це може бути досягнуто за допомогою різноманітних механік гри, таких як: система прогресування, складні лабіринти та загадки, багатошаровий геймплей та багато іншого. Крім того, успіх мобільної гри залежить від її візуальної привабливості та здатності захопити гравців з першого погляду. Розробники мобільних ігор повинні використовувати високоякісні графічні ефекти, яскраві кольори та відповідні звукові ефекти, щоб забезпечити захопливий візуальний досвід для гравців.

Також важливою складовою мобільних ігор є можливість грати в них з друзями або змагатися з іншими гравцями онлайн. Для цього розробники повинні включати у свої ігри різноманітні режими гри, які дозволяють гравцям змагатися між собою або спільно пройти складні рівні.

У цілому, успішна мобільна гра повинна мати простий, але захопливий геймплей, відповідну візуальну привабливість та можливість грати з друзями або змагатися з іншими гравцями онлайн. Для досягнення цих цілей розробники мобільних ігор повинні бути творчими та інноваційними, використовуючи нові технології та ідеї для створення унікальних та захопливих ігор.

1.3 Огляд та аналіз існуючих аналогів

1.31 Plants vs. Zombies 2

Plants vs. Zombies 2 (PVZ2) [2] є продовженням знаменитої гри, що була створена PopCap Games. Ця гра стала дуже популярною в народі, завдяки своїй простоті та глибині геймплею. Вона була випущена на мобільних платформах і до цього часу має велику кількість шанувальників.

Гра пропонує різні види рослин, які можуть відбивати наступ зомбі, який висувається на ваш газон, і захистити ваш дім від нападів на кожному рівні. В цій грі багато видів рослини та зомбі, різноманітні рівні та різні екранні ефекти, що роблять гру захопливою. Крім того, у PVZ2 є різні варіанти гри, включаючи турніри та щоденні виклики, які забезпечують більш довготривалий та цікавий геймплей.

Гра PVZ2 має дуже простий та доступний інтерфейс, що дозволяє новачкам легко розібратися з геймплеєм. Ігрова механіка також проста: гравець повинен розташовувати рослини, щоб зупинити наступ зомбі, та зібрати соняшники, щоб мати можливість купувати нові рослини. Крім того, гравець може використовувати різноманітні спеціальні атаки та заклинання, які можуть допомогти в боротьбі зі зомбі.

Одним з недоліків PVZ2 є те, що гра має деякий рівень випадковості, який може призвести до невдалих гравців. Наприклад, іноді гравець може мати погане стартове розташування рослин, яке може зробити його безнадійною відносно швидко. Також, деякі зомбі в цій грі можуть бути дуже складними для знищення, особливо якщо гравець не використовує правильну комбінацію рослин та відповідних здібностей. Крім того, деякі користувачі відзначають проблеми з оптимізацією гри на деяких пристроях, що може викликати проблеми з плавністю геймплею та завантаженням гри. Однак, ці випадки становлять тільки дрібні недоліки в порівнянні з загальною якістю гри.

Рис. 1.1. Зображення геймплею гри Plants vs. Zombies 2

Для розробки гри «Tower Dimension» було взято та покращено ідею кількості різноманітних веж та ворогів. Було додано можливість покращення веж під час гри, адже у PVZ геймплей майже закінчувався після того, як було розставлено усі вежі. Оскільки гравець отримає достатньо золота під час гри у сюжетному режимі, було позбавлено проблеми великої кількості внутрішньоігрових покупок.

1.3.2 Tower Madness 2

Tower Madness 2 - це гра в жанрі Tower defense, створена для мобільних платформ iOS та Android [3]. Гравець повинен захистити свою базу від наступу безлічі ворожих інопланетян, які намагаються здобути контроль над планетою.

Одним з найбільш яскравих аспектів гри є її графіка. Гра має кольоровий та деталізований дизайн з кумедними ворогами, що додає веселощів геймплею. Крім того, музика та звукові ефекти допомагають зануритись у світ гри.

Гра має різноманітність веж, які гравець може розташовувати на маршруті ворожих атак. Кожна вежа має свої унікальні характеристики та навички, які можуть бути покращені за допомогою накопиченого досвіду. Гра має десятки різних рівнів, де у кожного наступного зростає складність, через що гравець повинен розташовувати свої вежі та використовувати різні стратегії для захисту своєї бази.

Хоча Tower Madness 2 є досить цікавою і кумедною грою, вона також має деякі проблеми. Однією з основних проблем Tower Madness 2 є недостатня різноманітність ворогів. Хоча гра має кілька типів зомбі, їх кількість обмежена і швидко стає нудною для гравців. Вороги мають мало варіацій, і їхні атаки не змінюються, що знижує глибину геймплею. Це може призвести до того, що гравець швидко набридає гри та втрачає інтерес до неї.

Крім того, деякі гравці відзначають проблеми з управлінням грою, особливо на пристроях з меншим екраном, таких як смартфони. Деякі елементи інтерфейсу можуть бути надто маленькими, що може зробити їх важкими для торкання.

В цілому, Tower Madness 2 - це добре зроблена та захоплива гра, яка надає гравцям глибокий геймплей та виклик для розвитку своїх стратегічних та тактичних навичок. Гра має прекрасну графіку та звуковий дизайн, які доповнюють геймплей та роблять його більш захопливим. На жаль, одним з недоліків гри є недостатня різноманітність ворогів, що може призвести до того, що гра стає повторюваною та менш захопливою на довгий термін. Однак, загалом гра є дуже гарною та рекомендованою для любителів жанру.

Рис 1.2. Зображення геймплею гри Tower Madness 2

Для розробки гри «Tower Dimension» було взято ідею великої кількості рівнів, щоб користувачу не було нудно грати в одній і той самій локації. Кожна локація створювалась окремо, завдяки великої кількості префабів створених у програмі «Blender». Завдяки цьому було розв'язано проблему однотипного геймплею, адже в кожному рівні необхідно отримуватись різної стратегії, через різної кількості та типів ворогів, яких було занадто мало в Tower Madness 2. Також було проаналізовано проблему малого інтерфейсу у грі Tower Madness 2, через що на старих мобільних телефонах було неможливо створити вежу.

1.3.3 Infinitode 2

Infinitode 2 - це мобільна гра жанру Tower Defense, яка пропонує нескінчений режим гри. Ви займаєтеся захистом своєї бази від наступу хвиль ворогів, будуючи та покращуючи башти.

Основна особливість гри - це можливість грати в нескінченому режимі, де ви зіштовхуєтеся зі більш складними хвилями ворогів. Вам доведеться розміщувати башти в стратегічних місцях, використовуючи їх унікальні властивості та здібності, щоб зупинити противників. Гра пропонує різноманітність типів башт, які ви можете використовувати, і кожна з них має свої сильні та слабкі сторони.

У процесі гри ви будете заробляти гроші та ресурси, які можна витратити на покращення башт, розблокування нових типів башт та отримання унікальних ефектів. Ви також зможете отримувати досягнення та виконувати завдання, щоб отримати бонуси та винагороди.

Інтерфейс гри Infinitode 2 простий та зручний у використанні, що дозволяє легко розміщувати башти та керувати геймплеєм. Гра має досить просту графіку, але це дозволяє плавно працювати на різних мобільних пристроях.

Варто зазначити, що Infinitode 2 не має розгорнутої історії або кампанії. Головний акцент робиться на нескінченному режимі гри та стратегічному будівництві оборонних систем.

Якщо ви шукаєте нескінчені виклики у жанрі Tower Defense та любите будувати та стратегічно покращувати башти, то Infinitode 2 може бути цікавим варіантом для вас.

Основними плюсами гри є:

1. Нескінченний режим, де ви можете насолоджуватися грою без обмежень у часі та досліджувати все складніші хвилі ворогів.
2. Різноманітність башт зі своїми унікальними та стратегічними можливостями.

Основними недоліками гри є:

1. Обмежена графіка: Що стосується візуального аспекту, Infinitode 2 має досить просту та обмежену графіку.
2. Монотонність: З часом гра може стати монотонною, особливо в нескінченному режимі, де патерни ворогів повторюються. Це може призвести до відчуття одноманітності та втоми.
3. Відсутність сюжету.

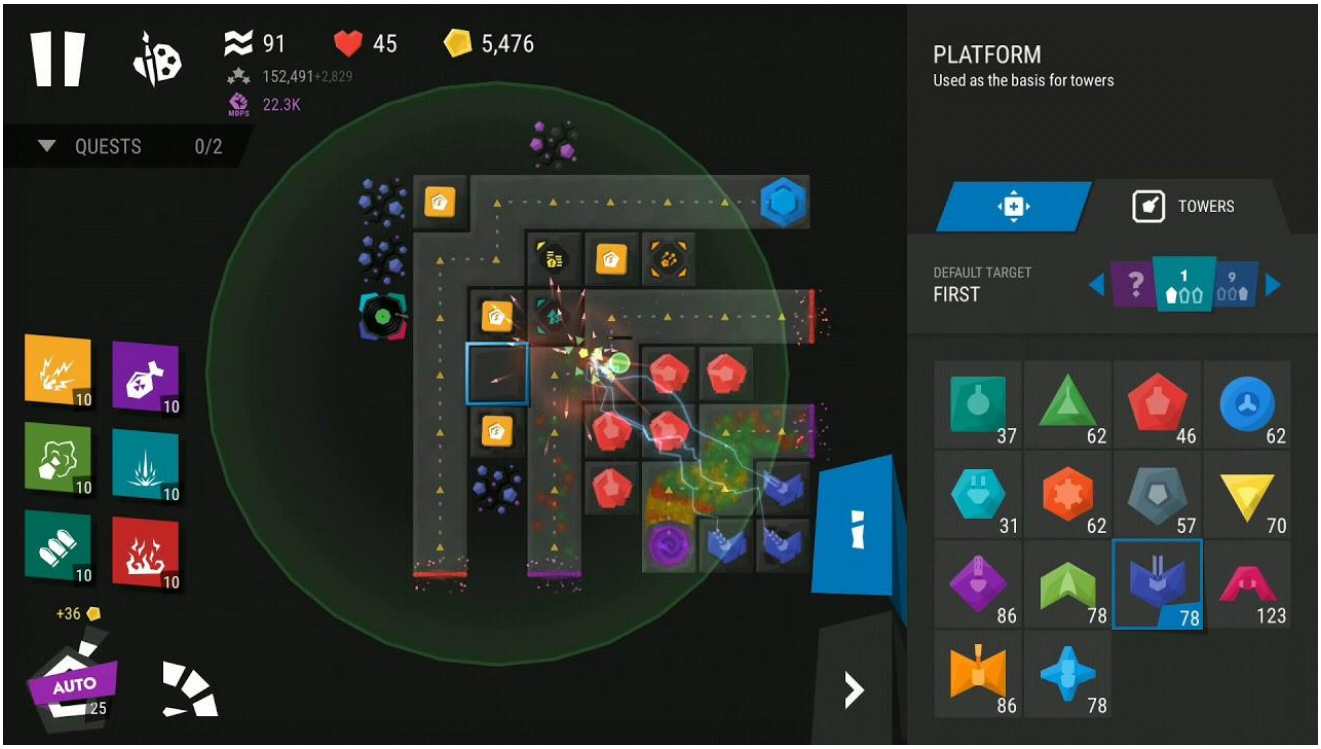


Рис 1.3. Зображення геймплею гри Infinitode 2

Для розробки гри «Tower Dimension» було взято ідею нескінченного режиму, де гравець зміг би змагатись с іншими гравцями у рекорді з набраного очків та механіку опису хвилі ворогів, завдяки якої ми можемо побачити які вороги будуть йти підчас рівню.

2. АНАЛІЗ ЗАСОБІВ РЕАЛІЗАЦІЇ

Гра жанру Tower Defense — це жанр ігор, у якому гравцеві потрібно будувати вежі для захисту від нападів ворогів. Для реалізації такої гри необхідно мати потужні інструменти для розробки ігор, такі як Unity, C# та Blender.

C# - це мова програмування, яка широко використовується для розробки ігор в Unity. Завдяки C# розробники можуть створювати складні ігрові механіки, обробляти події в грі, працювати з базами даних тощо. Використання C# дозволяє зробити гру жанру Tower Defense більш цікавою та різноманітною.

Blender - це потужний інструмент для 3D моделювання та анімації, який можна використовувати для створення об'єктів, ворогів та інших елементів гри жанру Tower Defense. Його можна використовувати для створення складних анімацій, налаштування текстур та освітлення. Використання Blender дозволяє створити якісну та реалістичну гру.

Unity - це ігровий двигун, який дозволяє створювати ігри для різних платформ, таких як ПК, мобільні пристрої, консолі тощо. Він надає широкі можливості для розробки ігор такі як створення рівнів, ворогів, веж, налаштування фізики, анімації та ефектів. Для того, щоб обрати Unity, було проаналізовано та розглянуто декілька популярних ігрових двигунів, таких як Unreal Engine, Game Maker Studio, Construct. Оскільки Unity використовує мову програмування C# та має перевагу над іншими двигунами у розробці для мобільних приладів, було обрано саме його для реалізації даної дипломної роботи.

Зведені результати аналізу характеристик розглянутих ігрових двигунів наведено у таблиці 2.1.

Таблиця 2.1 – Зведені результати аналізу програмного забезпечення для розробки ігор

Показник	**Unity**	**Unreal Engine**	**Game Maker studio**	**Construct**
Платформи	Windows, macOS	Windows, macOS, Linux	Windows, macOS	Windows, macOS, Linux
Обмеження у безкоштовній версії	-	-	+	+
Основна мова програмування	C#	C++	GML	відсутня
Рік появи	2005	1998	2012	2007
3D графіка	+	+	часткова	-
Фізичне моделювання	+	+	-	+
Місце на диску	4 ГБ	145 ГБ	3 ГБ	500 МБ
Мінімальний процесор	Intel Core i5-2500K	Quad-core Intel	Intel Pentium 4	Dual-core Intel
Мінімальна відеокарта	DX11 сумісна	DX11 сумісна	DX9 (shader model 2.0) сумісна	DX11 сумісна
Мінімальна оперативна пам’ять	8 ГБ ОЗП	8 ГБ ОЗП	512 МБ ОЗП	4 ГБ ОЗП
Кількість розробників	Має понад 5 млн активних розробників	має понад 7 млн завантажень	Не мають такої великої користувацької бази, як Unity та Unreal Engine, але все ще є досить популярними ігровими двигунами зі своїми унікальними можливостями.	
Приклади ігор	Subway Surfers; Holow Knight; Rust	Fortnite; Sea Of Thieves; Bioshock Infinite	Hotline Miami; Undertale;	TowerClimb
Інші особливості	використовується для розробки ігор на платформі ПК, консолі, мобільні пристрої та віртуальна реальність	використовується для створення ігрових проектів для ПК, консолей та віртуальної реальності.	використовується для створення 2D-ігор для ПК, консолей та мобільних пристроїв.	використовується для створення 2D-ігор різних жанрів для ПК, консолей та мобільних пристроїв.

Таким чином, використання Unity, C# та Blender дозволяє створити захопливу та високоякісну гру жанру Tower Defense з цікавими механіками та ефектами.

Рис. 2.1. демонстрація геймплею

2.1 Мова програмування C#

Аналіз мови програмування C# [4] є важливим елементом при розробці ігор на ігровому двигуні Unity. C# є мовою програмування, яка використовується для створення високоякісних, ефективних та безпечних програм. Мова C# є дуже потужним засобом для розробки ігор, оскільки має велику кількість вбудованих функцій і можливостей, які сприяють створенню високоякісної гри. Ця мова має ряд переваг, включаючи можливість використовувати різні типи даних, об'єктноорієнтований підхід, легкість налагодження та підтримку платформи .NET.

Також важливим аспектом є підтримка C# інструментів розробки, таких як Unity, який є одним з найпопулярніших ігрових движків на ринку. Unity дозволяє розробникам створювати гри з використанням C#, а також має вбудовану підтримку різних платформ, таких як iOS, Android та PC.

Розробка гри в жанрі Tower Defence потребує знань з різних галузей, включаючи гейм-дизайн, програмування та мистецтво. У цьому випадку, мова програмування C# може бути корисною для реалізації багатьох компонентів гри, таких як механіки взаємодії юніта з вежею, система прогресу, система нагород та покращень веж і т.д.

Основними перевагами використання мови програмування C# для розробки гри є можливість легко розширювати код, використовувати багато фреймворків та бібліотек для підвищення ефективності та швидкості розробки, а також використання широкого спектра інструментів для побудови графічного інтерфейсу та ефектів. Для розширення знань використано книгу «C# 8.0 and .NET Core 3.0 – Modern Cross-Platform Development» [5].

C# мова програмування, яка підтримується на багатьох IDE включаючи: Visual studio, JetBrains Rider, MonoDevelop, SharpDevelop. В Unity гарно підтримується Visual Studio, тому що обидві програми належать компанії Microsoft і розробляються з урахуванням взаємодії між собою. Це означає, що при використанні Visual Studio для програмування в Unity маємо більш точну інтеграцію, підсвічування синтаксису, автодоповнення коду та інші корисні функції, які спрощують розробку гри.

Однак, варто зазначити, що розробка гри в жанрі Tower Defence з використанням мови програмування C# може мати певні складнощі. Наприклад, може виникнути проблема з оптимізацією гри та її ефективністю, особливо на старіших пристроях або комп'ютерах з обмеженими ресурсами. Крім того, для успішного використання мови програмування C# у розробці ігор потрібно мати високу кваліфікацію та досвід в програмуванні загалом, а також знання певних ігрових двигунів та бібліотек. Без цього можуть виникнути проблеми з реалізацією складних механік гри та оптимізацією її процесів. Також важливо пам'ятати про оновлення мови та інструментів, адже їх використання може забезпечити покращення продуктивності та зменшення кількості помилок у процесі розробки. Загалом, мова програмування C# може бути потужним інструментом для розробки

ігор, проте вимагає від розробника великої уваги до деталей та знання найновіших тенденцій у світі ігрової індустрії.

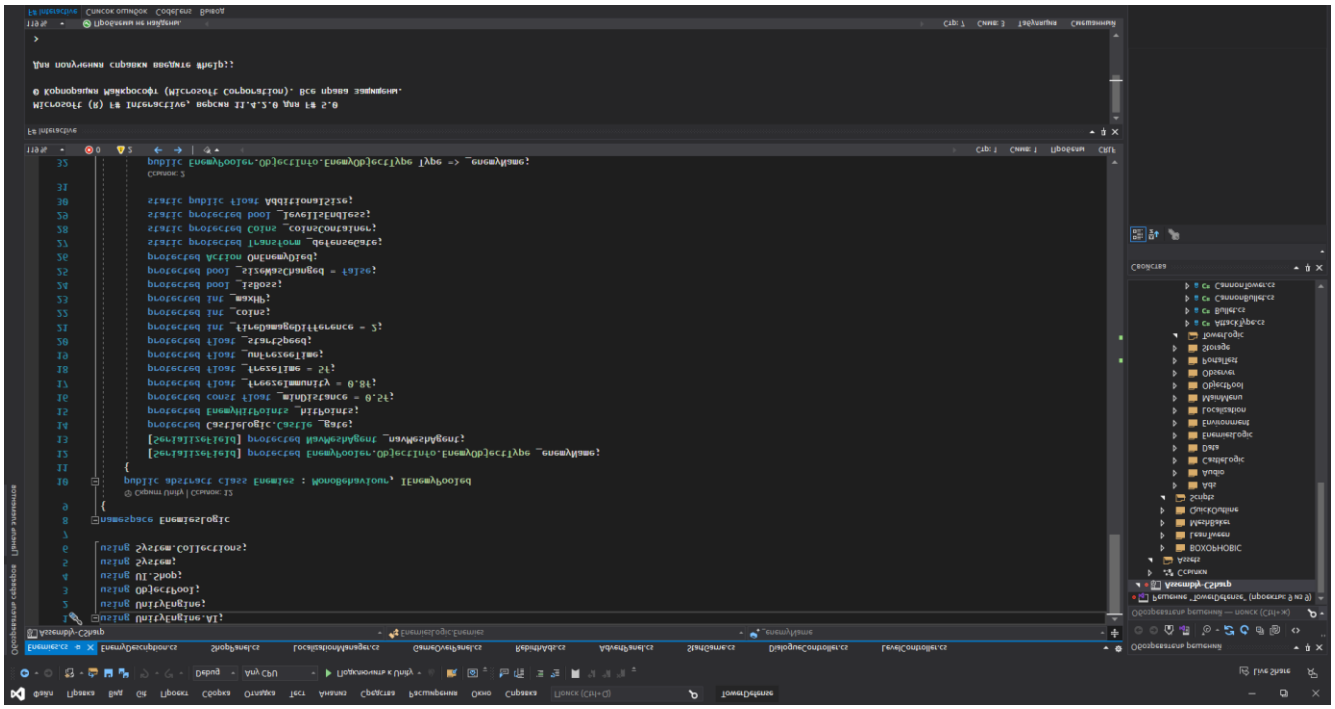


Рис. 2.2. демонстрація IDE Microsoft Visual Studio

2.2 Ігровий двигун Unity

Ігровий двигун Unity [6] є одним з найбільш популярних ігрових двигунів, який використовується для створення ігор в різних жанрах. Використання цього ігрового двигуна для розробки гри дозволить розробникам створити багато ефективних механік та ефектів, які зроблять гру цікавою та захопливою.

Unity також надає можливість створювати різноманітні ефекти, анімацію та надає потужні інструменти для розробки ігор, включаючи можливість створення та редагування об’єктів, що допоможе зробити гру більш привабливою та захопливою для гравців. Більшість ефектів можна створити за допомогою графічного інтерфейсу даного двигуна, що спростить процес розробки.

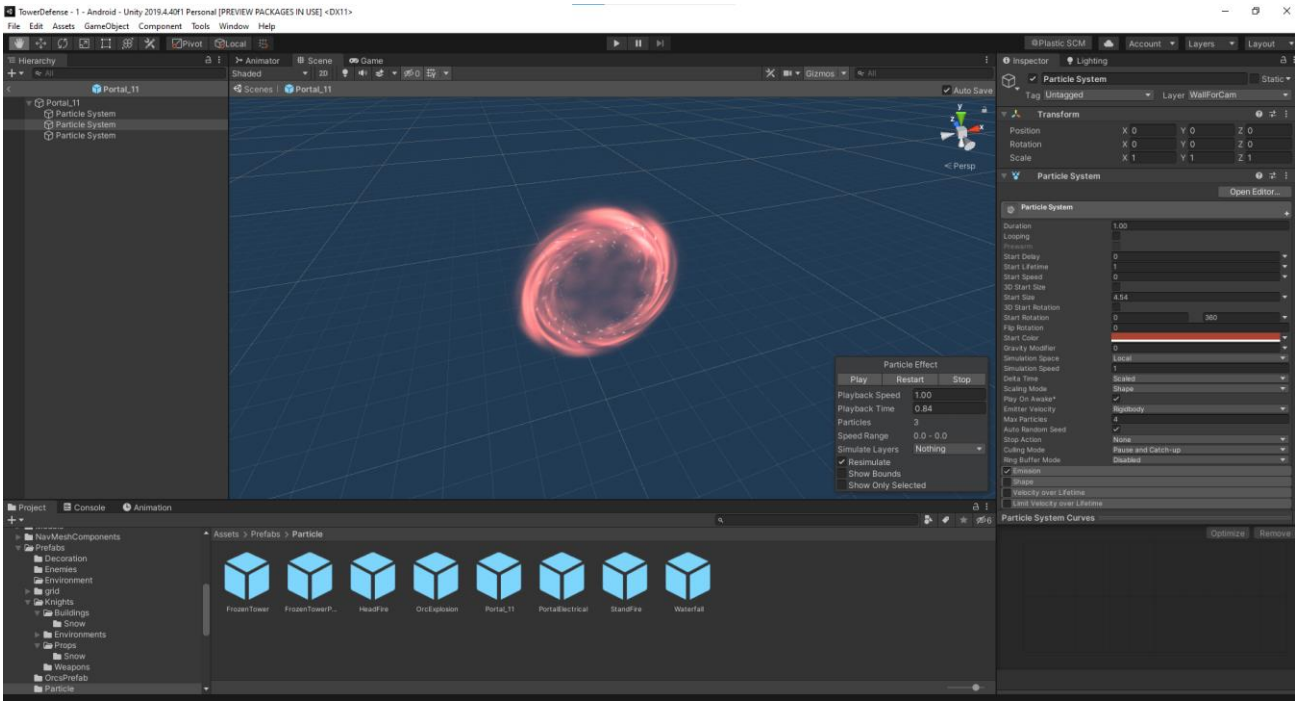


Рис. 2.3. створення ефектів на Unity

Однак, для успішної розробки гри з використанням Unity потрібні глибокі знання мови програмування C#, зокрема знання архітектури цієї платформи, класів та методів, які використовуються в процесі розробки. Також потрібно мати знання процесу розробки ігор та глибоке розуміння принципів роботи графічних двигунів. Середа розробки Unity має свої особливості, що можуть бути важкими для розуміння для новачків, тому для розробки гри в жанрі Tower Defence з використанням RPG механік необхідно мати практичний досвід роботи з цим ігровим двигуном або бути готовим швидко навчатися нового.

Одним з можливих недоліків використання Unity в розробці ігор є обмеження на деякі аспекти проєкту, такі як швидкість обробки великої кількості об'єктів чи робота з графікою в реальному часі, через що необхідно роздивлятись оптимізацію як невіддільну складову проєкту. Також, це IDE може вимагати великих ресурсів, що може стати проблемою для менших студій або індивідуальних розробників.

Однак, його використання має свої переваги. Це потужний ігровий двигун, який підтримує багато платформ, включаючи мобільні, десктопні та консольні

платформи. Unity має велику спільноту розробників та користувачів, що дозволяє легко знайти рішення на форумах та блогах. Також однією з особливостей цієї спільноти є підтримка розробників завдяки онлайн-магазину Asset Store [7], який є частиною Unity. Asset Store містить велику кількість готових ассетів, таких як моделі персонажів, текстури, звуки, плагіни та інші матеріали, які можуть бути використані для швидкої розробки ігор та додатків.

У Asset Store є безплатні та платні ассети. Безплатні ассети мають обмежені можливості або якість, але дозволяють користувачам познайомитись з різноманітністю різних ассетів та використовувати їх у своїх проєктах безплатно. Платні ассети можуть бути більш розширеними та якісними, залежно від ціни, яку встановив розробник.

Asset Store є корисним ресурсом для розробників, які шукають шляхи прискорення процесу розробки своїх ігор та додатків. Завдяки Asset Store розробники можуть швидко знайти необхідні ассети, які вони можуть використовувати у своїх проєктах, зменшуючи тим самим час та витрати на розробку. Крім того, Asset Store також дозволяє розробникам продавати свої власні ассети, що стимулює розвиток спільноти розробників та внесок у популяризацію Unity.

Крім того, Unity підтримує мову програмування C#, яка має велику кількість бібліотек та інструментів, що значно полегшує розробку гри. C# є потужною мовою програмування зі зручним синтаксисом та досить легкою для вивчення для програмістів, які раніше використовували інші мови програмування. За допомогою цієї мови програмування, розробники можуть досить просто створювати та керувати поведінкою об'єктів гри. Крім того, C# дозволяє більш ефективно управляти всією грою, зокрема збереженням інформації про героїв, ворогів, вежі, а також управлінням графічними ефектами, звуком та іншими аспектами гри.

Таким чином, використання Unity та мови програмування C# є вдалим вибором для розробки гри в жанрі «Tower Defence». Однак, необхідно мати

достатній досвід у розробці ігор та знати особливості роботи з цим ігровим двигуном для досягнення успіху у розробці гри.

2.3 Програмне забезпечення для створення тривимірної комп'ютерної графіки, моделювання та анімації Blender

Blender [8] - це потужний вільний інструмент для 3D-моделювання, анімації та візуалізації. У розробці гри в жанрі «Tower Defence» Blender може бути використаний для створення високоякісних 3D-моделей, а також для створення реалістичних анімацій та ефектів.

Для вивчення цього програмного забезпечення є багато онлайн ресурсів, один з таких - Blender Guru [9]. Blender Guru пропонує статті, навчальні відео та допоміжні файли, які включають від основ до просунених технік 3D-моделювання та анімації. Курси розроблені таким чином, щоб вони були легкими для зрозуміння навіть для початківців, але водночас інформативними та детальними для більш досвідчених користувачів.

Одним з головних переваг Blender є його безплатність та відкритий код, що дозволяє розробникам та художникам використовувати програму безплатно та налаштовувати її під свої потреби. Blender має широкий набір інструментів для моделювання, включаючи можливість створювати складні форми за допомогою багатьох методів, таких як скульптування, моделювання з використанням геометричних форм та багатьох інших.

Також, Blender має вбудовані інструменти для розробки анімації та візуалізації, що дозволяє розробникам створювати реалістичні анімаційні сцени, зокрема для руху ворогів та зброї. Blender також підтримує різні формати файлів, що дозволяє розробникам легко інтегрувати 3D-моделі та анімації в гру.

Під час розробки гри, Blender використовувався для створення рівнів, ворогів, декору та веж. За допомогою цього програмного забезпечення було створено анімації для руху ворогів та атак веж. Крім того, було використано Blender

для створення різних ефектів та декорацій, які допомагали збагатити гру та зробити її більш привабливою для гравців.

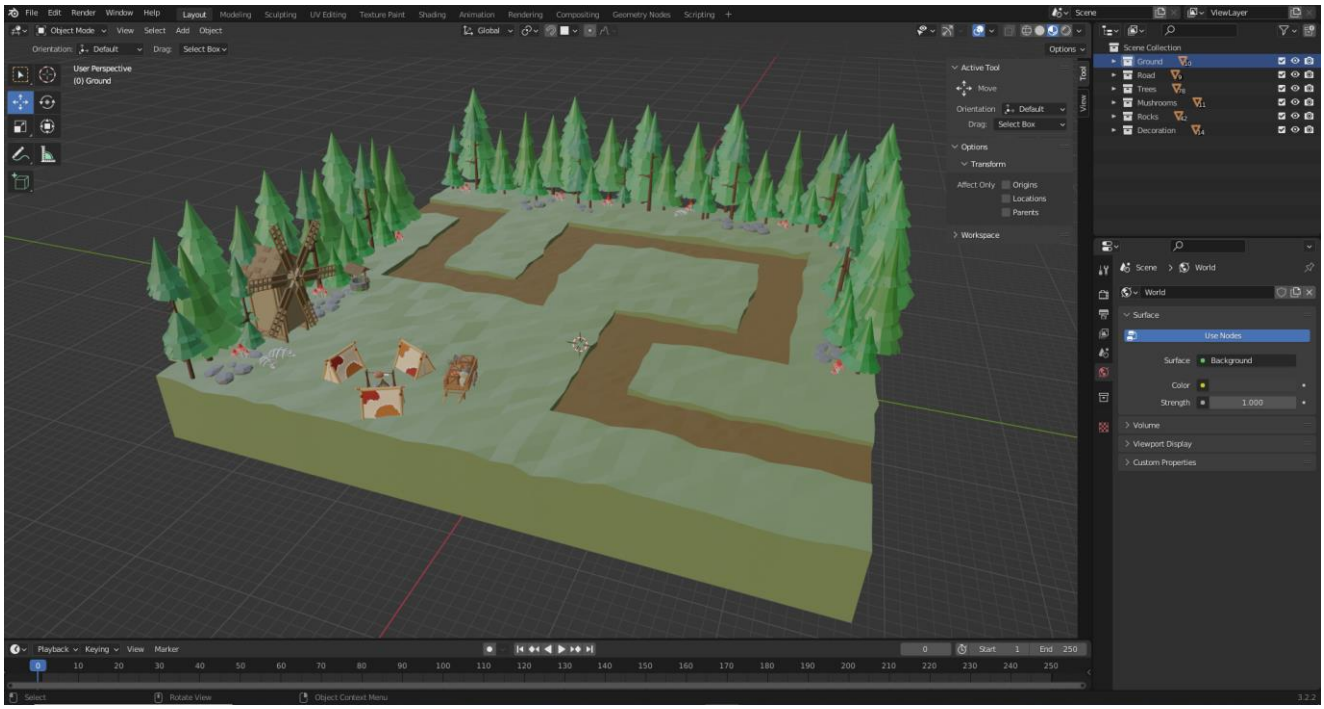


Рис. 2.4. сцена програмного простору Blender

Також Blender було використано для малювання спільної текстури для усіх об'єктів, завдяки розташуванню їх точок в одній спільній системі координат. Малювання спільної текстури дуже допомогло для оптимізації гри. Також завдяки цьому програмного простору можливо спостерігати за кількість полігонів об'єктів та оптимізувати їх, щоб забезпечити максимальну продуктивність гри. Крім того, Blender надав можливість створювати реалістичні та деталізовані об'єкти для гри, що підвищувало її якість та привабливість для гравців. Використання Blender також дозволило нам забезпечити сумісність між об'єктами в грі та легко їх імпортувати в ігровий двигун. Це дозволило досягнути єдності стилю та зовнішньому вигляду усіх елементів гри.

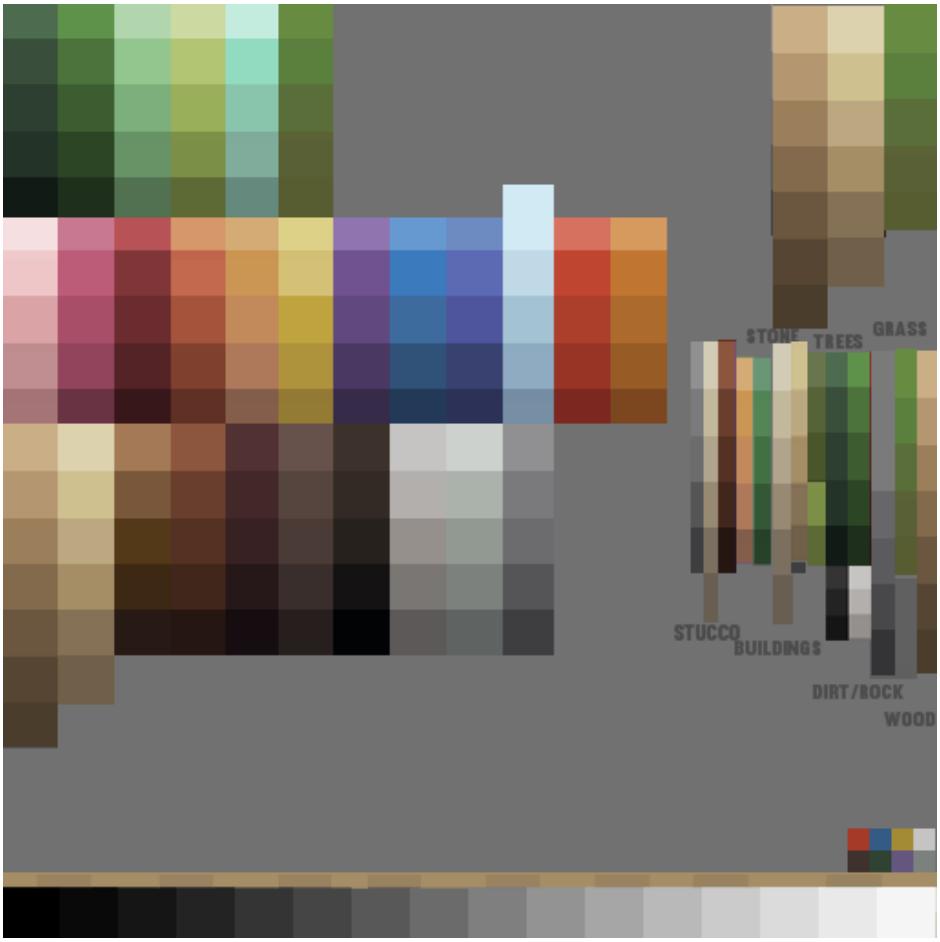


Рис. 2.5. Спільна текстура для об’єктів гри

Крім того, використання Blender дозволило зробити розробку гри більш ефективною та продуктивною, забезпечивши можливість працювати з усіма об'єктами гри в одному програмному середовищі. Це значно зменшило час та зусилля, необхідні для розробки гри, та дозволило зосередитись на створенні високоякісної геймплейної та графічної складових. Важливо зазначити, що Blender має велику кількість різноманітних аддонів, які можна використовувати для поліпшення роботи з програмою. Наприклад, існують аддони для створення пейзажів, редакторів анімації, плагіни для імпорту та експорту різних форматів файлів, та багато інших корисних додатків. Використання цих аддонів може значно полегшити роботу з Blender та зробити процес створення гри ще більш продуктивним.

3. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ГРИ ЖАНРУ «TOWER DEFENSE» З ЕЛЕМЕНТАМ RPG

3.1 Розробка основних механізмів гри

Перед тим як перейти до розробки гри, необхідно проаналізувати, що у грі жанру «Tower Defense» має бути.

В грі жанру Tower Defense зазвичай мають бути такі елементи:

1. Вежі або інші засоби оборони, які можна розміщувати на карті, щоб захистити від наступу ворожих сил.
2. Ворожі сили, які наступають на територію гравця і намагаються пройти до кінця карти.
3. Ресурси, які гравець може витрачати на покупку та підвищення рівня веж.
4. Система прогресу гравця, така як очки досвіду, рівні, досягнення та нагороди.
5. Різноманітні картки або рівні з різними ландшафтами, характеристиками та складністю проходження.
6. Механізм управління вежами або іншими засобами оборони, які гравець може покращувати та удосконалювати.
7. Елементи геймплею, такі як пауза, відновлення гри, магазин з можливістю купівлі нових веж та інші додаткові функції.
8. Графіка та звукові ефекти, які допомагають створити атмосферу гри та доповнюють геймплей.
9. Нескінченний режим, в якому гравець зможе змагатися з іншими гравцями
10. Діалогова система, вибори та наслідки, щоб гравець ще більше понурився до ігрового всесвіту.

3.1.1 Взаємодії ворогів з вежами

Для взаємодії веж з ворогами, було розроблено базовий абстрактний клас «Tower», який зберігає метод «CheckIfCanShoot». Логіка цього метода складається в тому, що коли об’єкт з LayerMask «enemy» входить до Physics.OverlapSphere, вежа зберігає найближчого ворога та починає по ньому стріляти.

Для взаємодії ворогів з кулями, було використано шаблон проєктування Visitor [10], який дозволяє додатково розширити функціональність об'єкта без зміни його основної структури. Кожен тип куль має унікальні характеристики та ефекти, тому кожен ворог повинен мати власну унікальну реакцію на кожен тип куль. У грі було використано інтерфейс IEnemyVisitor для взаємодії між об'єктами куль та ворогів. Коли куля торкається ворога, вона передає себе інтерфейсу IEnemyVisitor як параметр, що дозволяє ворогові унікально реагувати на кожну кулю. Таким чином, застосування шаблону проєктування «Visitor» на двигуні Unity, дозволяє ефективно управляти взаємодією між об'єктами гри, легко додавати нові типи об'єктів та алгоритмів, без зміни основного коду.

Для реалізації шаблону «Visitor» на двигуні Unity використано наступний підхід: створення інтерфейсу IEnemyVisitor, який містить методи TakeDamage, що приймають кожен тип об'єктів з якими повинні бути взаємодії. Далі було створено різні класи відвідувачів, які у випадку дотику до ворога звертаються до його інтерфейсу IEnemyVisitor і викликають метод TakeDamage, передаючи себе як параметр.

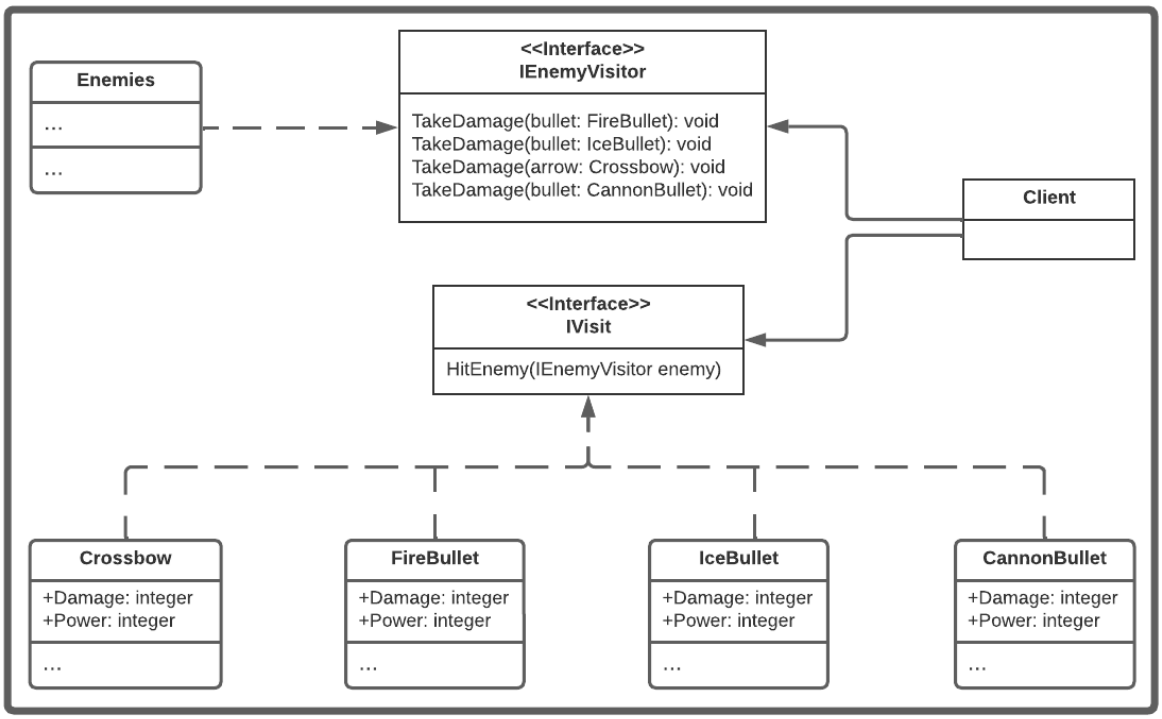


Рис. 3.1. UML Діаграма класів шаблону проєктування «Visitor»

У цьому випадку, було створено інтерфейс IEnemyVisitor, який містить методи для обробки кожного типу куль та ворогів. Далі було розроблено інтерфейс IVisit, який містить метод HitEnemy. В цьому методі використовується метод TakeDamage інтерфейсу IEnemyVisitor якому передається поточний клас. Таким чином, коли куля торкається ворога, викликається метод HitEnemy, який передає кулю відповідному класу відвідувачу, який реалізує логіку обробки взаємодії між кулею та ворогом.

Отже, використання шаблону проєктування «Visitor» на двигуні Unity дозволяє з легкістю реалізувати логіку взаємодії між об'єктами, що знаходяться на сцені гри. Це дозволяє зосередитися на розробці ігрових механік, не звертаючи увагу на складну логіку взаємодії між об'єктами.

Застосування шаблону проєктування «Visitor» допомагає зменшити зв'язаність між класами та забезпечити легку розширюваність системи. Це дозволяє змінювати логіку гри та додавати нові елементи без великих змін в коді. Також, використання

шаблону проєктування «Visitor» допомагає зберегти код чистим та організованим, що робить його легким для розуміння та підтримки.

3.1.2 Логіка ворогів

У грі жанру «Tower Defense» вороги відіграють важливу роль. Вони є головними перешкодами, які гравцеві потрібно подолати, щоб досягти успіху в грі. Вороги різних типів і з різними характеристиками можуть мати різний ефект на гру.

По-перше, наявність різних ворогів робить гру більш складною та цікавою. Якщо гравець не матиме жодних перешкод, то гра швидко стане нудною та простою. Вороги створюють напругу та стимулюють гравця думати стратегічно, вибирати правильну стратегію та відповідати на непередбачувані ситуації.

По-друге, вороги дозволяють розвивати систему балансу в грі. Різні типи ворогів можуть мати різні характеристики, такі як швидкість, життєві очки, сила атаки, унікальні здібності тощо. Це дає можливість розробникам гри налаштовувати баланс, щоб гра була чесною та цікавою.

По-третє, наявність ворогів дозволяє гравцеві взаємодіяти зі своїм середовищем. Гравець повинен ставити башти та інші будівлі в такий спосіб, щоб вони ефективно захищали його від ворогів. Це вимагає від гравця не тільки знання характеристик своїх башт, але й розуміння того, як вороги можуть рухатися та як їхні атаки можуть впливати на башти та замок.

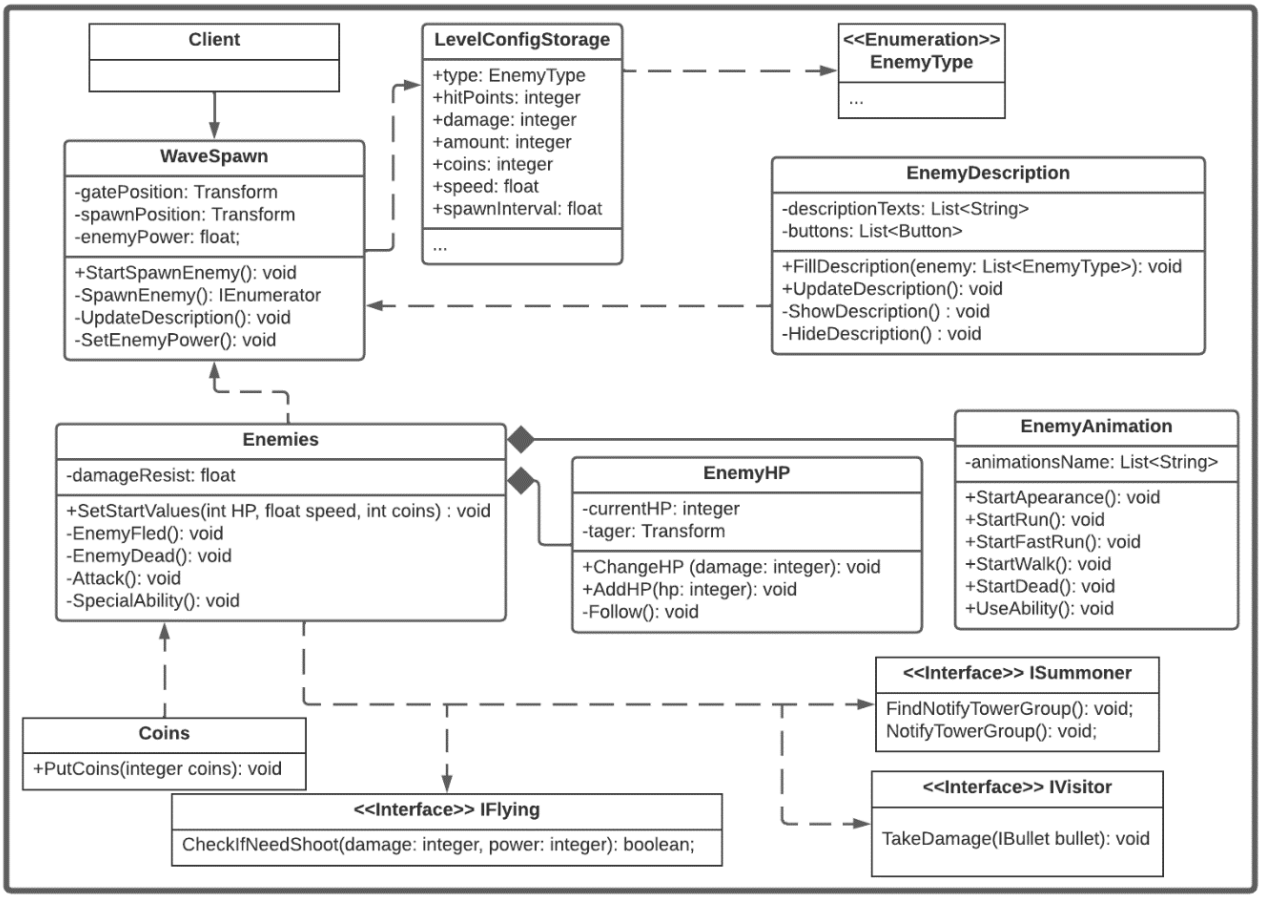


Рис. 3.2. UML діаграма основних классів відповідаючих за ворогів.

На рисунку 3.2 зображені класи «WaveSpawn» та «LevelConfigStorage» відповідають за кількість та характеристики ворогів. Клас «LevelConfigStorate» зберігає налаштування поточного рівня, а саме кількість, тип та характеристики ворогів. Коли усі ресурси гри завантажені, клас «WaveSpawn» звертається до класу «LevelConfigStorage», щоб взяти характеристики ворогів та викликає свій метод «StartSpawnEnemy», який запускає карутину «SpawnEnemy». Після цього, клас «WaveSpawn» звертається до класу «EnemyDescription», викликаючи його метод «FillDescription» передаючи йому поточний тип ворогів.

Коли карутина «SpawnEnemy» відпрацьовує, клас «WaveSpawn» звертається до пулу об’єктів, та створює ворога поточного типу, передаючи йому параметри з класу «LevelConfigStorage». Усі вороги успадковуються від класу «Enemies», тому

ми можемо викликати у будь-якого ворога метод «SetStartValues», таким чином передаючи стартові параметри.

Кожен ворог містить клас «EnemyAnimation», який відповідає за анімації кожної дії ворога та клас «EnemyHP», який відповідає за здоров’я та отримане пошкодження. Також кожен ворог реалізує інтерфейс «IVisitor», який було розглянуто раніше та застосовує метод класу «Coins» під назвою «PutCoins». Цей метод дозволяє давати гравцю золоту, якщо ворога було вчасно знищено.

В грі є унікальні вороги зі здібностями, які впливають на ігровий процес. Для таких ворогів було розроблено інтерфейси «ISummoner» та «IFly». Наприклад, вороги, які можуть літати, до яких неможливо дібратись без спеціальної вежі, саме такі вороги реалізують інтерфейс «IFly», який підстроєний спеціально для цих особливих веж. Оскільки повітряні вежі дуже повільні, у гравця може статись проблема, коли декілька таких веж водночас почнуть стріляти по ворогу. Для цього у ворога з інтерфейсом «IFly» є метод «CheckIfNeedShoot», який повертає false, якщо для перемоги ворога вже цілиться достатньо веж. В свою чергу вороги з інтерфейсом «ISummoner» можуть створювати менших ворогів. Через те, що в проєкті було використано шаблон проєктування «Observer», вежі перестають шукати ворогів після того, як певна кількість ворогів вже загинула. Тому, вороги з цим інтерфейсом мають сповістити вежі, що з’явились нові вороги, котрих потрібно шукати.

3.1.3 Механізм створення веж та магазину

В грі є певні міста, натиснувши на які гравець відкриває магазин, в якому він може створити декілька типів веж, продати вежу, або покращити вежу що вже існує. Для цього було розроблено клас «TowerPlace», який взаємодіє класом «Tower» та «ShopPanel». TowerPlace зберігає інформацію, чи є на ньому зараз вежа, чи максимальний її рівень. При натисканні на TowerPlace, він викликає у ShopPanel метод «ChangeMenu()». Цей метод дивиться на дані цього TowerPlace, та залежності вид них викликає один з методів «ShowMainMenu» або

«ShowUpdateTowerMenu()». Ці методи відрізняються відображенням кількістю кнопок, їх функціоналом. Метод «ShowMainMenu» відображає усі кнопки створення веж (якщо поточний рівень дозволяє), в той час, коли метод «ShowUpdateTowersMenu()» відображає лише кнопки покращення (якщо у вежі не максимальний рівень) та кнопку продажі.

При натисканні на кнопку купівлі вежі ми звертаємось до класу «Coins», та викликаємо в нього метод «TakeCoins» передаючи йому дані зі статичного класу «Prices», який зберігає ціни веж (цей статичний клас необхідний для того, щоб у головному меню була можливість зменшити ціну на вежу, завдяки зароблено золоту впродовж гри). Метод «TakeCoins» перевіряє, чи достатньо в гравця зараз золота для того, щоб створити вежу. Якщо золота достатньо, метод повертає true, клас «ShopPanel» викликає метод «CreateTower» та звертається до класу «TowerPlace», який викликає у класу «Tower» метод «CreateTower». Цей метод відповідає за поступове відображення частин вежі, щоб зробити процес будівництва більш складним та цікавим. Якщо метод «TakeCoins» повертає false, гравцю буде показано попередження, в якому сказано, що гравцю недостатньо золота для будівництва вежі.

При взаємодії з кнопкою, яка продає вежу, клас «ShopPanel» викликає метод «SellTower», який взаємодіє з метод «ReturnCoins» у класу «Coins», та методом «DestoyTower» класу «TowerPlace».

Також був розроблений клас «CastlePanel», який завжди відображає кнопки для взаємодії з замком. Ці кнопки при натисканні працюють майже аналогічним чином, тільки замість веж будують барикаду, відновлюють здоров’я замку, або створюють пастку.

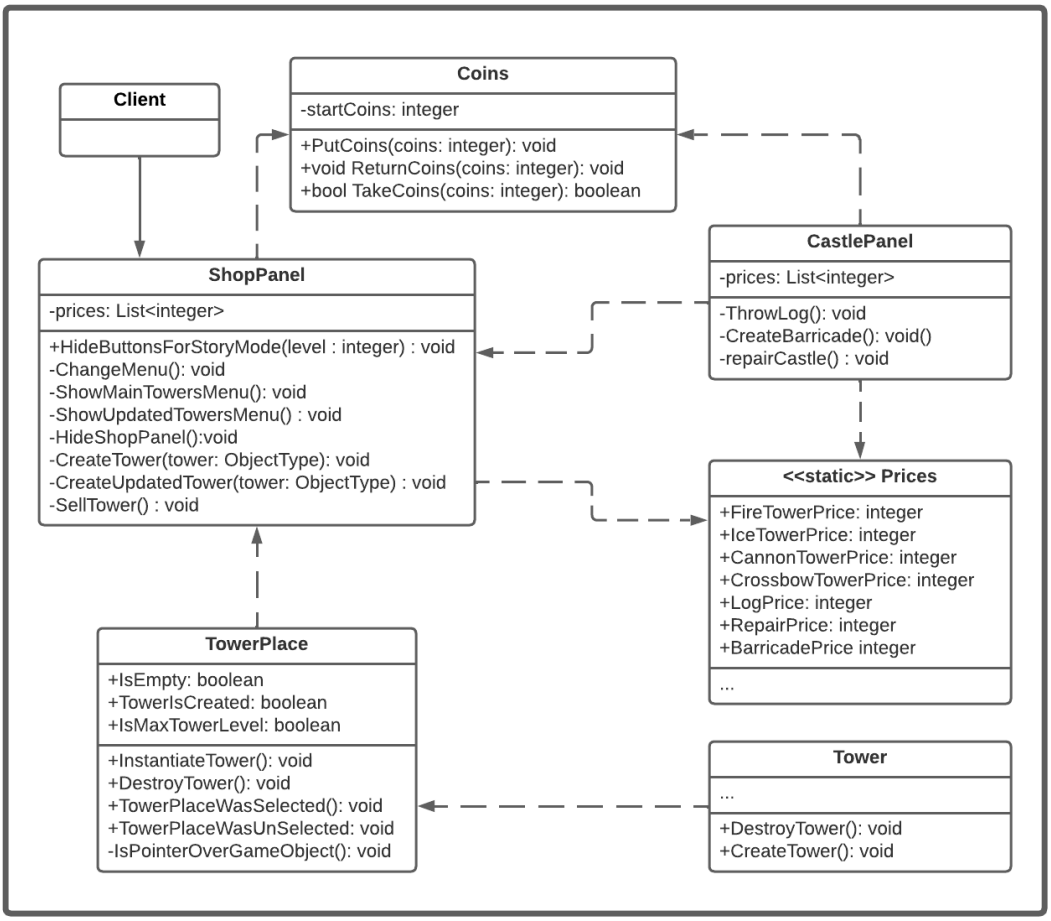


Рис. 3.3. UML діаграма классів відповідальних за логіку магазину.

Рис. 3.4. GUI магазину для будівлі веж

3.1.3 Локалізація гри

Локалізація гри є важливим етапом в її розробці, оскільки дозволяє пристосувати гру до потреб різних мовних та культурних груп користувачів, що розширює аудиторію гри та підвищує її популярність.

Завдяки локалізації, гра може бути доступна для гравців з різних країн, які говорять різними мовами. Локалізована гра може бути більш зрозумілою та привабливою для користувачів, оскільки вона містить нативні мовні та культурні елементи, що дозволяє гравцям зануритися в гру більш повно.

Крім того, локалізація гри може збільшити дохід від продажу, оскільки вона привертає більше користувачів з різних країн, які готові придбати гру. Також це дозволяє розширювати ринки збуту та залучати нових інвесторів.

У процесі локалізації необхідно перекласти ігрові текстові рядки на різні мови, а також враховувати культурні особливості різних країн, щоб зробити гру якомога більш привабливою для користувачів.

Варто пам'ятати, що локалізація - це постійний процес, і після випуску гри можуть з'являтися нові оновлення, які потребують локалізації. Тому важливо забезпечити можливість легкої та швидкої локалізації додаткового контенту.

Для реалізації локалізації у грі розроблено класи «LocalizationManager», «LocalizedText» та дві структури «LocalizationData», «LocalizationItem». Клас «LocalizationManager» містить у собі методи для завантаження локалізаційних даних з файлу, збереження вибраної користувачем мови в налаштуваннях гри, отримання локалізованого тексту за ключем та методи для сповіщення про зміну мови.

Клас «LocalizedText» відповідає за оновлення текстових полів, які містять локалізовані дані. Цей клас підписується на подію «OnLanguageChanged», яка виникає під час зміни мови у «LocalizationManager», та оновлює текст з використанням методу «GetLocalizedValue» класу «LocalizationManager».

Також клас «LocalizationManager» містить в собі дві вкладені структури – «LocalizationData» та «LocalizationItem», які являють собою структури для збереження ключа та значення локалізованого тексту.

В класі «LocalizationManager» реалізовано підтримку різних мов за допомогою завантаження даних з JSON-файлів, які містять дані локалізації. Для зручності використання у коді було реалізовано статичну змінну «_isReady», яка дозволяє перевірити, чи завантажились всі дані для локалізації.

У загальному, реалізація локалізації у грі здійснюється за допомогою класу «LocalizationManager», який надає зручний спосіб для завантаження та отримання локалізованих даних. Клас «LocalizedText» дозволяє автоматично оновлювати текстові поля у грі під час зміни мови.

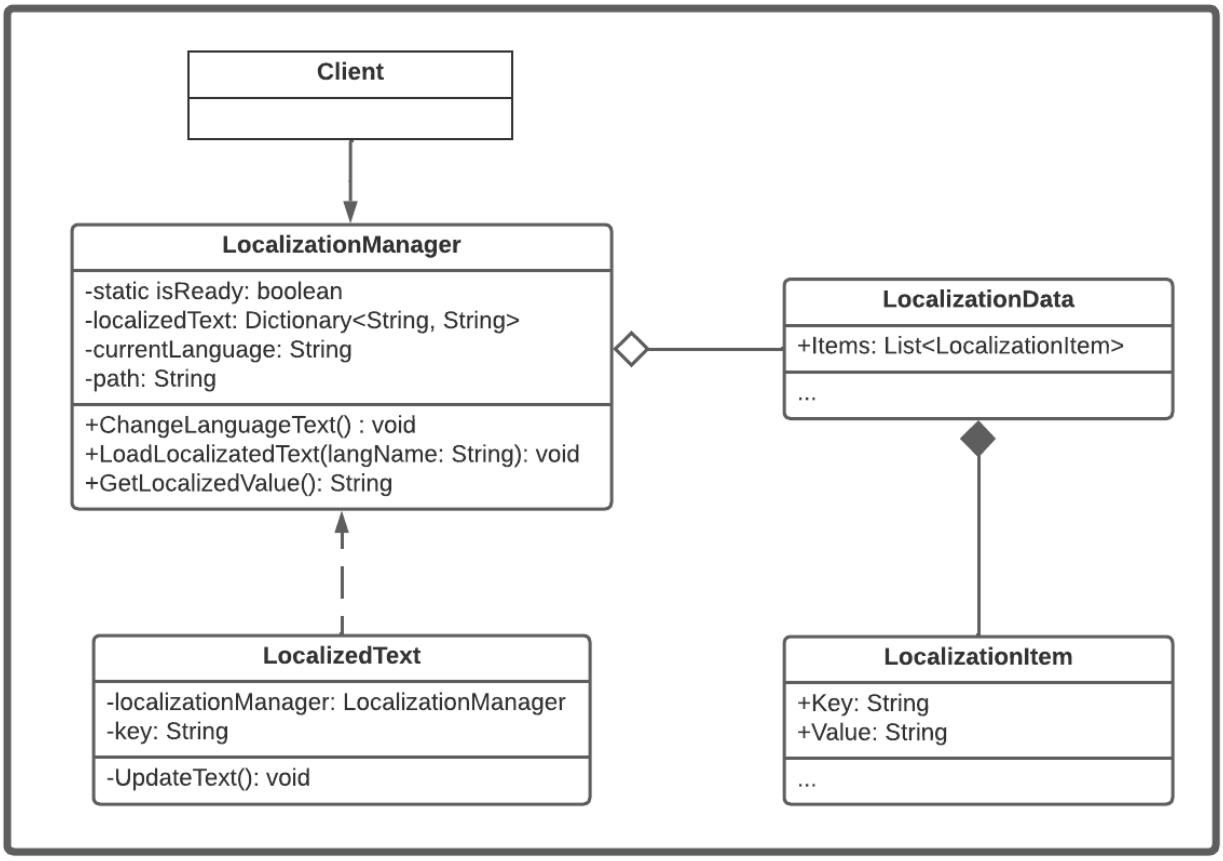


Рис 3.5. UML діаграма класів відповідальниї за локалізацію

3.1.4 Магазин у головному меню гри

У багатьох іграх зазвичай присутній ігровий магазин, де гравець може купити покращення для своїх персонажів або предметів, що збільшують їх ефективність у грі. Ігрові магазини з покращеннями дозволяють гравцям витрачати внутрішню валюту гри на різноманітні предмети, що дозволяє їм підвищувати свої шанси на перемогу або поліпшувати ігровий досвід.

Іноді в магазині можуть бути навіть ігрові скіни які також відомі як косметичні предмети, що впливають тільки на зовнішню складову гри. Скіни можуть містити нові зовнішні вигляди персонажів, такі як одяг, кольори шкіри, зачіски, аксесуари або зброя. Вони також можуть вносити зміни візуального вигляду предметів або оточення, такі як текстури, колір, анімація тощо.

Зазвичай, ігровий магазин містить різноманітні предмети, які гравець може купувати за внутрішню валюту гри. Окрім зовнішнього вигляду в магазині також можуть бути нова зброя, різні предмети, енергетичні напої та інші покращення, які дозволяють гравцю стати сильнішим в грі.

Крім того, ігрові магазини можуть додавати до гри елементи геймплею, які дозволяють гравцям збільшувати власний рівень та отримувати нові можливості. Наприклад, в іграх типу RPG гравець може купувати різні покращення, які дозволяють йому розвивати свого персонажа, збільшувати його здібності та отримувати нові навички.

Одна з форм реалізації ігрових скинів полягає у використанні внутрішньої грошової одиниці, яку гравець може заробити, обміняти або придбати за допомогою реальних грошей. За допомогою цієї грошової одиниці гравець може придбати доступ до різних скинів у внутрішньому магазині гри. Інша форма - це додатковий контент, який можна придбати окремо в магазині платформи, де доступні ігри.

Ігрові скини мають переваги як для розробників, так і для гравців. Розробники отримують додаткові доходи, що дозволяє їм продовжувати підтримку гри та розробляти новий контент. Гравці, з свого боку, можуть насолоджуватися

ігровим процесом, налаштовуючи свій вигляд унікальними способами та виражаючи свою індивідуальність.

Усі ці фактори зробили ігрові магазини з покращеннями важливим елементом багатьох ігор, що дозволяє розширювати можливості гравців та додавати до гри нові елементи геймплею.

Для гри «Tower Dimension» було розроблено магазин, який дозволяє покращувати характеристики кожної вежі окремо, що дозволить гравцю вигадувати свою стратегію, за якою він буде грати. Також у кожної вежі є свої унікальні характеристики, наприклад у гармати є покращення, яке дозволяє збільшити радіус вибуху кулі, на відмінну від інших.

Для цього було розроблено статичні класи «ShopPrices» та «TowerStandartCharacteristics». Ці класи містять стартові дані о характеристиках веж та цін магазину. Завдяки цьому, при першому запуску гри ми можемо зберегти ці дані за допомогою «PlayerPrefs».

PlayerPrefs - це простий інструмент збереження даних в Unity. Він дозволяє зберігати значення різних типів даних, таких як рядки, числа, булеві значення та інші, між сеансами гри. Усі дані зберігаються під ключами з класу «TowerCharacterics» або «TowerMenu». Якщо у гравця недостатньо ігрової валюти при натисканні на кнопку покращення, йому буде запропоновано подивиться рекламу. В іншому випадку характеристика вежі буде покращена, та ціна підвищена завдяки методу «UpdateButton».

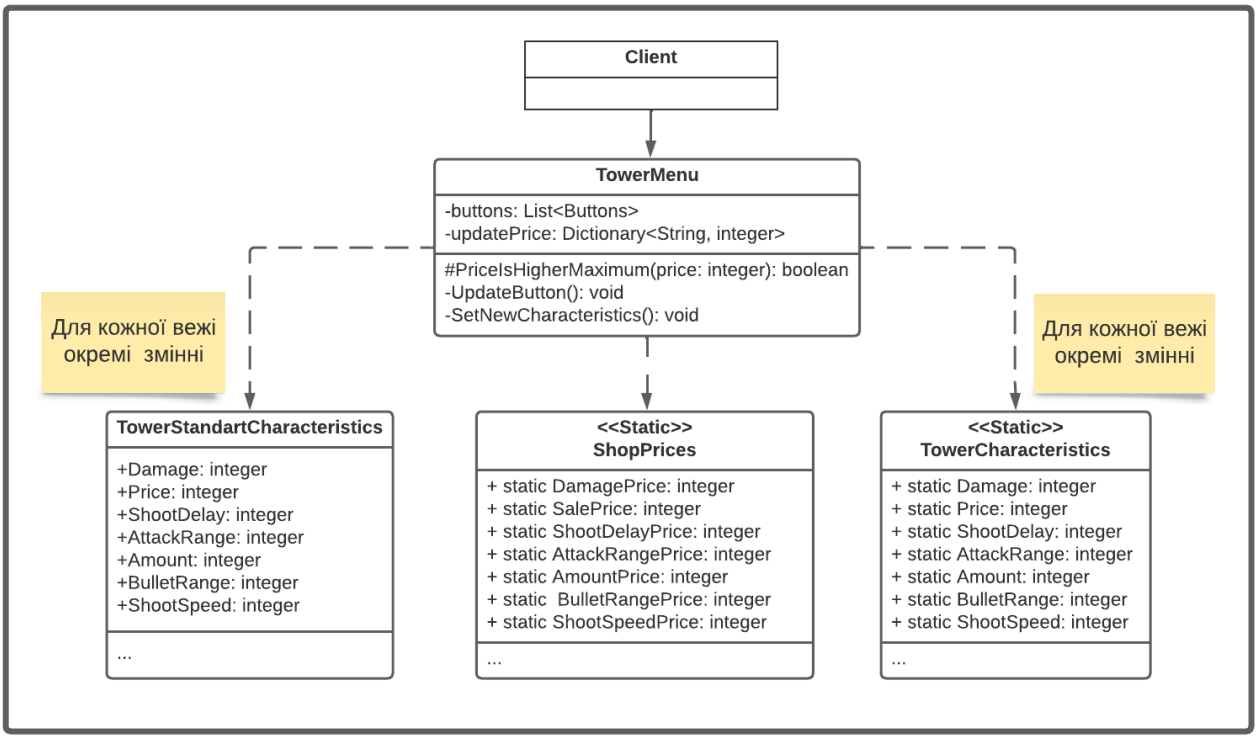


Рис. 3.6. UML діаграма класів відповідаючих за роботу внутрішньо-ігрового магазину в головному меню

Рис. 3.7. Демонстрація GUI внутрішньо-ігрового магазину

3.1.5 Щоденна нагорода у головному меню гри

Щоденна нагорода - це тип винагороди в іграх, яка пропонує гравцеві заробляти певну кількість валюти або ресурсів за входження до гри кожен день. Це може стимулювати гравців грати в гру щодня і збільшувати активність гравців у грі.

Щоденні нагороди можуть бути різними в залежності від гри, але вони зазвичай мають прогресивну структуру, де кожен день, який гравець входить до гри, він отримує вищу нагороду, ніж попереднього дня. Як правило, якщо гравець пропустив день, прогрес нагороди скидається на початковий рівень.

Наприклад, у деяких іграх щоденні нагороди можуть бути наступними:

1. День 1: 100 монет
2. День 2: 200 монет
3. День 3: 300 монет
4. День 4: 400 монет
5. День 5: 500 монет

Гравець може отримати ці нагороди, залежно від того, чи входив він до гри протягом п'яти днів підряд. Якщо гравець не входив до гри протягом одного дня, то його нагорода скидається на 100 монет (початковий рівень).

Щоденні нагороди мають кілька цілей. Вони стимулюють гравців повертатися до гри щодня, що підвищує активність гравців і сприяє підтримці грою активності гравців. Регулярне входження до гри також допомагає гравцям підтримувати зв'язок з ігровою спільнотою, дозволяючи їм спілкуватися, спільно грати або конкурувати з іншими гравцями.

Також нагороди надають гравцям додаткові ресурси, які можна використовувати для розвитку персонажа, покращення або отримання нових предметів. Це може стимулювати гравців досягати нових досягнень, просуватися в ігровому світі та виконувати більш складні завдання, витрачаючи менше часу.

Для реалізації щоденних нагород був розроблені класи: «ButtonDailyReward», «DailyReward», «RewardPref», «Reward» та enum RewardType.

В головному меню у гравця буде кнопка, яка буде використовувати клас «ButtonDailyReward» та перевіряти за допомогою класу «DailyReward», чи є у гравця нагорода. У випадку true клас почне використовувати метод «ShowRewardMark» та показувати зображення «sign», даючи гравцю знати, що в нього є нагода. При натисканні на цю кнопку, буде викликатись метод «HideAndShow», який покаже панель щоденної нагороди.

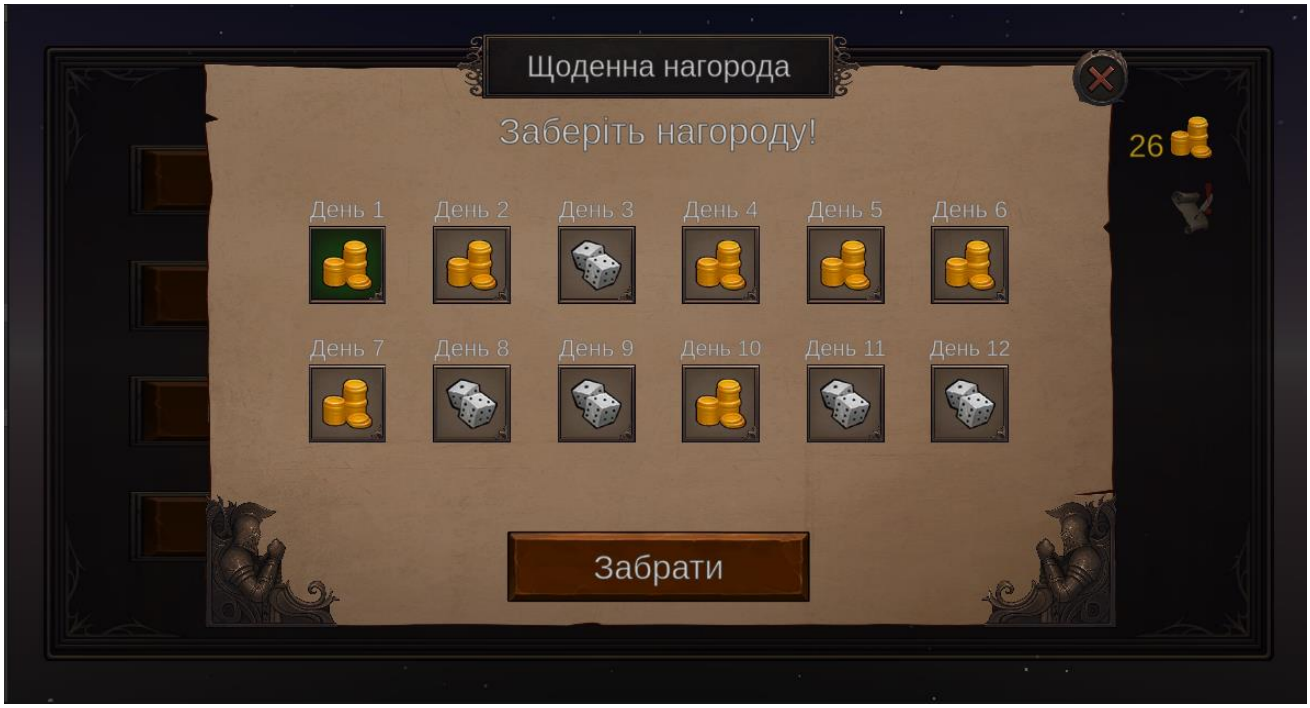


Рис. 3.8. Демонстрація GUI щоденної нагороди

Клас «DailyReward» відповідає за відображення іконок нагород, часу до наступної нагороди, збереження поточного дня нагороди та отримання нагороди. Завдяки списку «RewardPref», який зберігає тип нагороди, клас «Daily Reward» викликає метод «InitPrefabs», передаючи елементам списку дні яким ці нагороди належать та поточний день нагороди, викликаючи метод «SetRewardData» кожного елементу. Якщо їх день менший за поточний, вони змінюють колір заднього фону.

При натисканні на кнопку «Claim» клас «DailyReward» викликає метод «ClaimReward» та звертається до класу «ClaimRewardPanel», у якого викликає метод «Show», передаючи йому тип нагороди та поточний день. Після цього клас «ClaimRewardPanel» перевіряє тип нагороди завдяки методу «CheckRewardType», та відштовхуючись від нагороди обирає необхідну логіку.

В кінці гравець отримає ігрову валюту, клас «ClaimRewardPanel» звертається до класу «DailyReward» та викликає у нього метод «UpdateRewardState», завдяки чому поточний день збільшується, поточна нагорода змінює свій колір.

Після цього клас «DailyReward» звертається до класу «ButtonDailyReward» та викликає у нього метод «RewardWasTaked», який ховає зображення нагороди «sign».

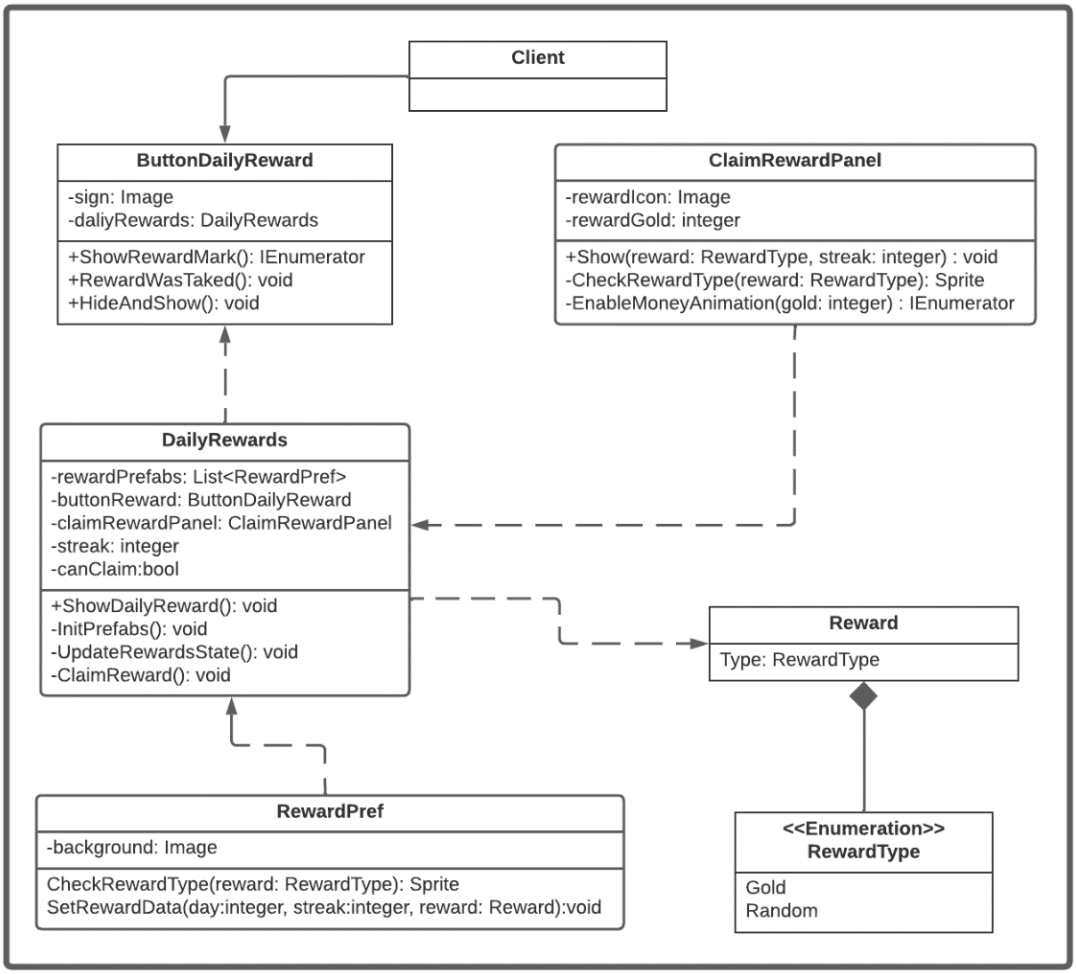


Рис. 3.9. UML діаграма класів відповідаючих за щоденну нагороду

3.1.6 Дерево діалогів

Дерево діалогів - це інструмент для створення послідовних діалогів між персонажами в іграх або програмах. Воно дозволяє створювати різні сценарії розмов між персонажами, в залежності від того, які відповіді вибере користувач.

Дерево діалогів може мати різні рівні, залежно від складності діалогу. На кожному рівні можуть бути різні варіанти відповідей для гравця, і в залежності від вибору гравця можуть бути відкриті різні гілки діалогу.

У дереві діалогів можуть бути також додаткові елементи, такі як інформаційні повідомлення, запити на виконання завдань або спеціальні події, які можуть змінювати хід ігри.

Створення дерева діалогів може бути складним процесом, який вимагає від розробника планування та детального пророблення всіх можливих варіантів діалогів. Однак, правильно спроєктоване дерево діалогів може створити цікаву імерсивну гру, де гравець буде відчувати, що його дії впливають на хід сюжету.

В контексті діалогів в іграх, вибір може мати різні наслідки в залежності від ситуації та контексту гри. Зазвичай від цього залежить далі розвиток сюжету та взаємодії гравця з ігровим світом. Наприклад, вибір діалогової опції може призвести до того, що гравець отримає нові завдання, знайде додаткову інформацію про персонажів або відкриє нові локації. З іншого боку, вибір також може мати негативні наслідки, наприклад, може спричинити конфлікт з іншими персонажами, зменшити можливості гравця або призвести до закінчення гри.

Розгалужені діалоги та вибір гравця є важливою складовою багатьох ігрових жанрів, таких як RPG, пригодницькі ігри та ігри з відкритим світом. Вони дозволяють гравцеві відчути вплив своїх виборів на гру, створити власний унікальний досвід та сприяють поглибленню імерсії в ігровий світ.

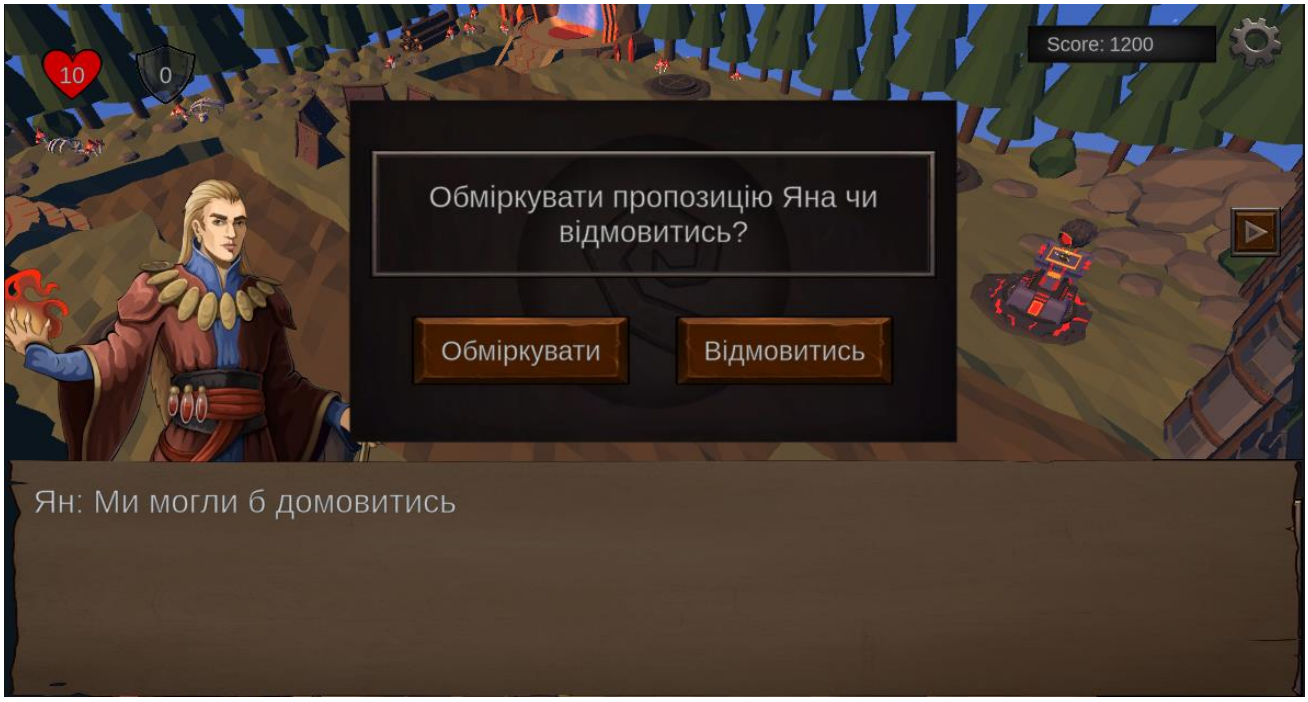


Рис. 3.10. демонстрація дерева діалогів гри «Tower Dimension»

Для створення такого дерева діалогів було розроблено класи «DialogueControlle», «Dialogue», «DialoguePressAnyKey», «ChoiceStatus», «ChoiceController».

Клас «DialogueController» відповідає за відображення діалогів, та передає кожному класу «Dialogue» подію «ChangeSpriteOnImage». Це необхідно для того, щоб у кожного діалогу відображався персонаж який каже репліку. На початку рівні, якщо є діалоги, цей клас відображає перший діалог завдяки методу «ActivateStartText», в залежності від минулих виборів. Ці вибори ми отримаємо від класу «ChoiceController», який зберігає минулі вибори через «PlayerPrefs» за string ключами статичного класу «ChoiceStatus».

Клас «Dialogue» відповідає за відображення наступного діалогу, викликаючи у нього метод «CreateText», якщо змінна «nextDialogue» не пуста. Якщо у поточного діалогу змінна «isHaveChoice» не пуста, відображає панель вибору, викликаючи у класу «ChoiceController» метод «ShowChoicePanel». Інакше якщо «nextDialogue» та «isHaveChoice» пусті, діалог закінчується, та починається гра.

Клас «ChoiceController» відповідає за збереження вибору, після натискання однієї з кнопок, цей клас зберігає дані по ключу завдяки класу «ChoiceStatus» та звертається до «DialogueController» та викликає у нього метод «MakeChoice», передаючи йому номер вибору. В залежності від цього номера, клас «DialogueController» активує діалог з номером переданим йому.

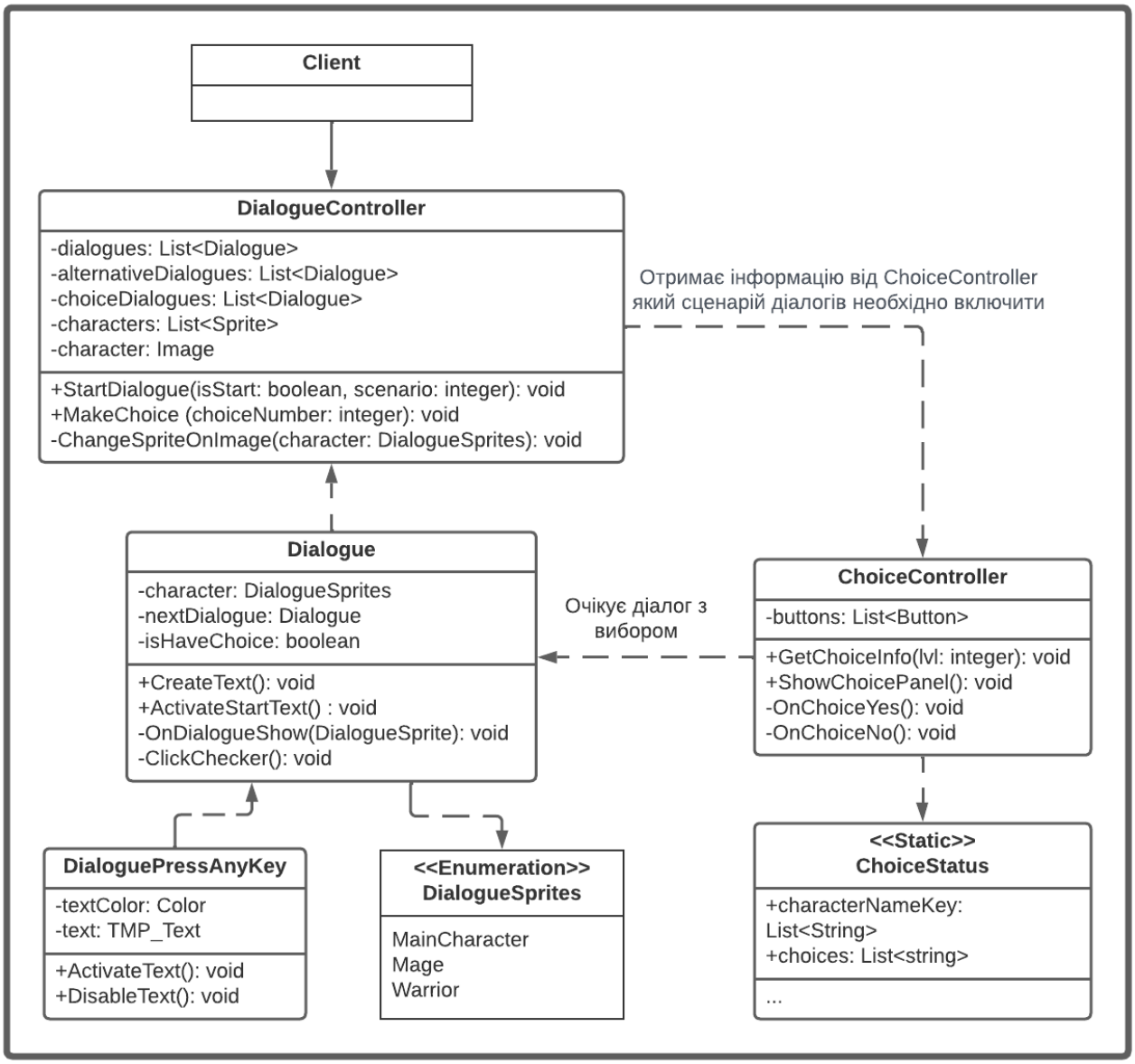


Рис. 3.11. UML діаграма класів відповідаючих за дерево діалогів

3.1.7 Аудіо у грі

Аудіо є необхідною частиною будь-якої гри, оскільки воно дозволяє підсилювати ефективність гри, створюючи атмосферу та підкреслюючи дії гравців. У грі можуть використовуватись різноманітні ефекти звуку, музика, голосові елементи, залежно від жанру і задуму авторів.

Для того, щоб забезпечити аудіо в грі, зазвичай використовуються бібліотеки аудіодвигунів, які забезпечують відтворення аудіофайлів в режимі реального часу зі змінною гучністю, темпом, та іншими параметрами. Основні бібліотеки аудіодвигунів, які використовуються в гральній індустрії, це FMOD, Wwise, Unity Audio, Unreal Audio, і CRI ADX2.

Крім того, для того, щоб забезпечити якісне аудіо, необхідно використовувати якісні аудіо-файли з високою роздільною здатністю та бітрейтом. Також важливо правильно розміщувати звукові джерела в грі, щоб досягти найбільшого ефекту імерсії для гравців.

Для розробки гри було використано «Audio Listener» та «Audio Source» [11]. Audio Listener - це компонент, який дозволяє відстежувати звук у всьому 3D просторі гри. Кожна сцена має лише один Audio Listener, який можна розмістити на будь-якому об'єкті в сцені. Якщо Audio Listener розміщений на персонажі гравця, то звук буде сприйматися так, ніби він його чує. Якщо Audio Listener розміщений на статичному об'єкті, то звук буде поширюватися з центру цього об'єкту.

Audio Source - це компонент, який прикріплюється до об'єкта, що генерує звук, наприклад, до музичного плеєра, ефекту звуку вибуху чи діалогового вікна. Кожен Audio Source може відтворювати лише один звук. Audio Source містить параметри, такі як гучність, панорамування (співвідношення звуку між лівим і правим каналами), висоту (співвідношення звуку між динаміками) та інші.

Коли гра відтворює звук, звук від Audio Source збирається у Audio Listener, який потім відтворює звук згідно з його настройками. Обидва компоненти дозволяють налаштовувати різні параметри, такі як відстань, звукові ефекти,

просторові налаштування тощо, щоб створювати більш реалістичну атмосферу для гравця. Для створення налаштувань audio було використано клас «OptionManager», який зберігає усі налаштування гри. У налаштуваннях в нас є повзунок, який зберігає значення аудіо використовуючи «Exposed Parameters» мікшерів «Music», та «Effects».

Exposed Parameters в аудіо - це інструмент в Unity, який дозволяє контролювати різні аспекти звуку, такі як гучність, панорамування та фільтрацію, прямо в редакторі Unity. Використовуючи «Exposed Parameters», можна забезпечити динамічний звук в грі, що залежить від ігрових подій, таких як зіткнення гравця зі стіною або збільшення рівня енергії головного героя.

Для використання «Exposed Parameters», потрібно спочатку створити мікшер звуку в Unity і додати до нього аудіокліпи. Після цього можна додати «Exposed Parameters», визначити їх назви та значення за замовчуванням. Потім можна встановити значення цих параметрів через скрипти.

Для відтворення аудіо було розроблено клас «AudioManager», до якого звертаються усі об’єкти, якщо їм необхідно відтворити звук. Також для збереження налаштувань звуку було розроблено клас «OptionManager», який зберігає значення повзунка звуку у GUI завдяки PlayerPrefs, та виставляє пропорційне значення в «Exposed Parameters», що дозволяє зберігати налаштування гри. Завдяки цьому користувачу не потрібно кожного разу змінювати звук, адже гра зберігає його вибір, та відтворює його минулі налаштування при запуску гри.

3.2 Створення моделей для веж та ворогів

Для веж та ворогів був використаний стиль «Low poly». Цей стиль в основному використовується для створення 3D-моделей мобільних ігор, де оптимізація графіки грає важливу роль. Зниження кількості полігонів дозволяє зменшити навантаження на графічний процесор та збільшити продуктивність гри.

Для створення моделей були використані примітиви. Примітиви – це геометричні форми, такі як куби, циліндри, сфери, конуси та інші, які можна створити одним кліком. Вони дозволяють швидко створювати базові форми, які потім можна змінювати та деталізувати.

Для зміни форми об’єкта використовувались такі базові функції:

1. Extrude - це один з базових інструментів моделювання в Blender, який дозволяє створювати нові геометричні форми шляхом витягування (видавлювання) вибраних областей з поверхні моделі.
2. Subdivide - дозволяє розділити обрану грань, ребро або поверхню на більшу кількість менших граней або поверхонь.
3. Inset Edge Loop - це інструмент, який дозволяє додати новий ряд граней всередині обраної границі грані або поверхні.
4. Bevel - це інструмент, який змінює форму та розмір краю геометричного об'єкта. Цей інструмент створює округлення, що може покращити вигляд та реалістичність моделі.

Одним з важливих аспектів створення low poly моделей є вибір кольорів та текстур. Використання простих кольорів та текстур дозволяє зберегти мінімальну кількість даних та збільшити продуктивність гри. Текстури можливо створювати у blender, в розділі «Texture painting». Texture Painting - це один зі способів створення текстур в Blender, де ви можете розфарбувати 3D-модель безпосередньо на її поверхні.

Для того, щоб розпочати Texture Painting, вам потрібно мати модель, на яку ви хочете нанести текстуру. Ви також повинні мати створену текстуру, на яку ви

будете робити розмалювання. Ви можете створити нову текстуру в Blender, або імпортувати зображення з іншого джерела.

Після цього ви можете відкрити панель «Texture Paint» (перемикач знаходиться внизу екрану в режимі «3D Viewport»), щоб почати роботу з Texture Painting.

Під час роботи з Texture Painting ви можете використовувати різні інструменти, наприклад, кисть, гумка, маскування, текстурні кисті та інші. Ви можете змінювати розмір та жорсткість кисті, вибирати кольори та текстури для покладання.

Крім того, Texture Painting дає вам можливість перейти до інших шарів текстури, таких як normal map та рельєф, щоб налаштувати деталі вашої моделі.

Після того, як ви завершите Texture Painting, ви можете зберегти вашу текстуру та застосувати її на вашу модель.

Для всіх ворогів була створена анімація. Blender має вбудований редактор анімації, що дозволяє створювати анімацію за допомогою ключових кадрів. Ключовий кадр встановлює значення властивостей об'єкта (таких як положення, обертання, масштабування тощо) на певний момент часу. Потім Blender автоматично інтерполює значення між ключовими кадрами, щоб створити плавний перехід між ними.

Щоб анімувати об'єкт в Blender, спочатку потрібно створити арматуру або скелет, який визначає рух кісток в 3D-просторі. Потім потрібно прикріпити модель об'єкта до кісток за допомогою процесу, який називається rigging.

Rigging дозволяє створити ієрархічну структуру кісток, щоб керувати рухом моделі. Після того, як модель прикріплена до кісток, можна почати анімувати об'єкт, створюючи ключові кадри для кожної кістки, щоб встановити її положення, обертання та масштабування на певний момент часу

3.3 Оптимізація та розробка спільної текстури гри

Оптимізація - це процес вдосконалення продуктивності та ефективності вашого проєкту. Він містить в себе різні техніки та стратегії, які дозволяють зменшити витрати пам'яті, збільшити швидкість роботи, підвищити якість графіки та забезпечити кращу ігрову динаміку.

Оптимізація гри є важливим етапом в розробці будь-якої гри, оскільки вона дозволяє досягти більшої продуктивності та підвищити задоволення гравців від гри. Більш того, від оптимізації залежить розмір гри, що може бути критичним для мобільних пристроїв з обмеженою кількістю пам'яті. Оскільки мобільні пристрої мають обмежені обчислювальні ресурси та батарею, вони не здатні обробляти велику кількість складних графічних об'єктів та процесів.

Крім того, мобільні пристрої мають різні характеристики, такі як роздільна здатність екрана, процесор та пам'ять, що може впливати на продуктивність гри. Оптимізація дозволяє забезпечити максимальну продуктивність на різних мобільних пристроях, знизити навантаження на процесор та пам'ять, а також підвищити ефективність батареї.

Для досягнення цієї мети, розробники можуть використовувати різні техніки та інструменти, такі як шаблон проєктування «Object pool», інструмент склеювання кілька mesh в один «Meshbaker», створення спільної текстури за допомогою розташування трикутників та чотирикутників з граней об'єкта на мальованої текстури в «Blender» та використовуючи можливості ігрового двигуна Unity [12]. Крім того, оптимізація також може містити оптимізацію коду, використання правильних налаштувань в Unity та управління ресурсами.

3.3.1 Object Pool

Техніка оптимізації Object Pool [13] в Unity дозволяє зменшити кількість створених та знищених об'єктів у вашій грі. У розробці ігор, часто необхідно створювати та знищувати багато однакових об'єктів, наприклад, кулі, вороги, ефекти та інші.

Замість створення та знищення об'єктів знову та знову, Object Pooling зберігає зарезервовані об'єкти в пулі та перерозподіляє їх при необхідності. Коли об'єкт більше не потрібен, він не знищується, а повертається до пулу, де може бути використаний знову пізніше.

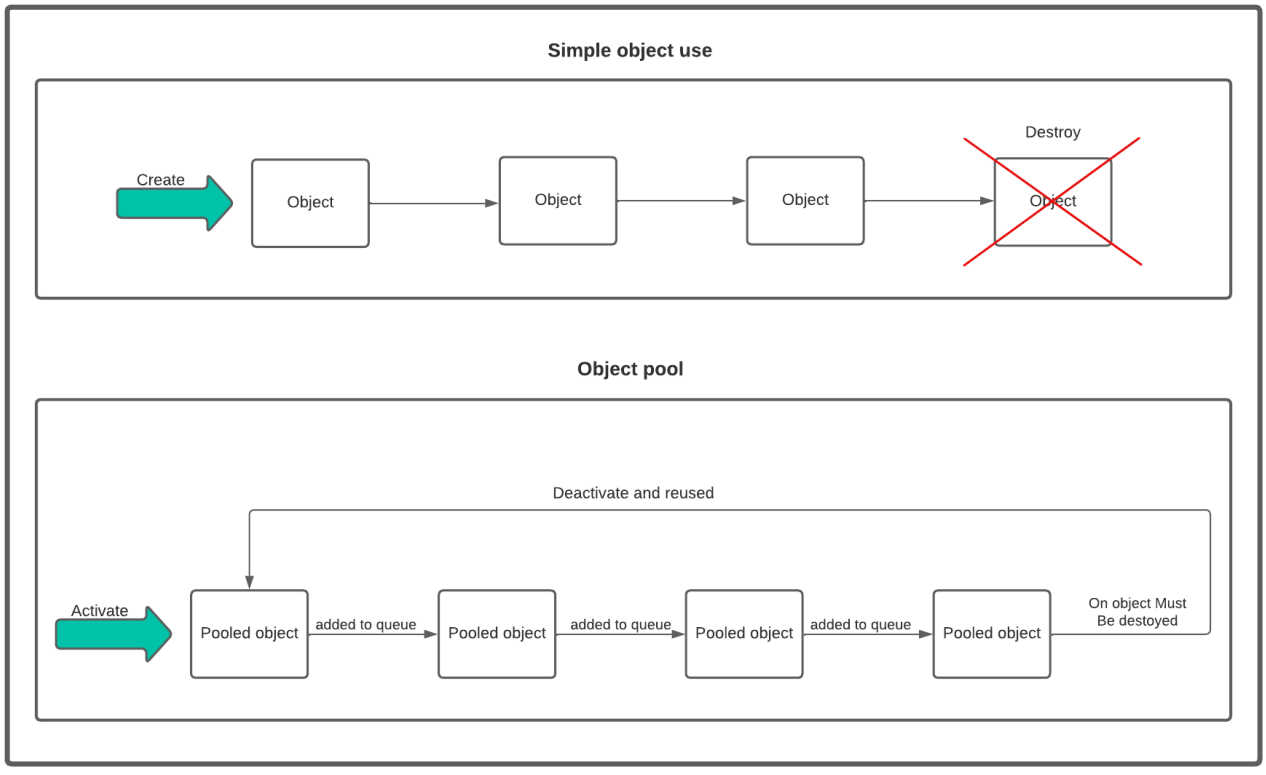


Рис. 3.12. діаграма роботи шаблону проєктування Object Pool

Зазвичай бажано тримати всі повторно використанні об'єкти, які в цю хвилину не використовуються, в одному пулі об'єктів, щоб ними можна було керувати за допомогою єдиної узгодженої політики.

В Unity, для реалізації Object Pooling, можна використовувати підхід з ручним створенням пулу об'єктів, або використовувати готові бібліотеки, які дозволяють створювати та управляти пулами об'єктів швидко та просто.

Для створення Object Pooler в Unity, було розроблено скрипт, який створюватиме пул та зберігатиме зарезервовані об'єкти у масиві. Після цього, ви можете використовувати методи, щоб взяти об'єкти з пулу, повернути їх назад, або додати нові об'єкти до пулу, якщо це необхідно.

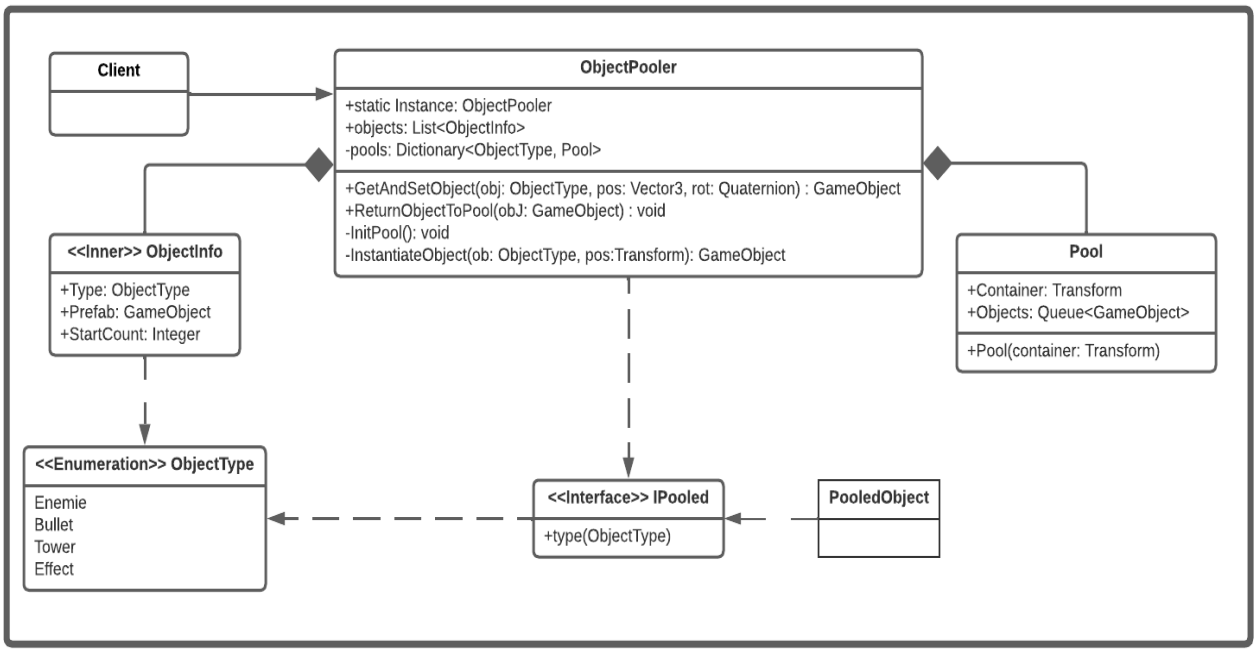


Рис. 3.13. UML діаграма класів шаблону проєктування Object Pool

Object Pooling є ефективним способом оптимізації вашої гри, оскільки дозволяє зменшити кількість створених та знищених об'єктів, що може покращити продуктивність та швидкість вашої гри. Використання Object Pooler дозволяє покращити геймплей, забезпечити кращий досвід користувача та зробити гру доступною для більшої аудиторії.

3.3.2 Mesh baker

Ассет для ігрового двигуна Unity, під назвою MeshBaker [14], дозволяє розробникам об'єднувати кілька mesh в один оптимізований mesh. Це може допомогти покращити продуктивність гри, зменшуючи кількість викликів методу «Draw» [15] і покращуючи ефективність рендерингу.

Метод «Draw» в Unity відповідає за відображення графічного контенту на екрані. Цей метод викликається під час рендерингу кадра, і він працює наступним чином:

1. Відбувається підготовка до рендерингу. Це означає, що Unity встановлює необхідні стани OpenGL або DirectX, такі як матриці проєкції та виду, налаштування освітлення, камеру, матеріали, текстури та інші параметри, щоб готувати сцену для відображення.
2. Unity розраховує, які об'єкти повинні бути видимими на екрані в залежності від позиції та орієнтації камери, рівнів деталізації та інших параметрів.
3. За допомогою алгоритмів обходу графу сцени Unity визначає, які об'єкти повинні бути відображені в якому порядку, щоб забезпечити правильне накладання об'єктів на екрані.
4. Unity перевіряє, які частини об'єктів видимі, а які - ні. Якщо частина об'єкта знаходиться за камерою або за іншим об'єктом, то вона не буде відображена.
5. Далі, Unity викликає метод Draw для кожного об'єкта, який потрібно відобразити. Цей метод збирає всі необхідні дані для відображення об'єкта, такі як вершини, індекси та матеріали, і надсилає їх на рендеринговий двигун.
6. Рендеринговий двигун, в свою чергу, відображає об'єкт на екрані, забезпечуючи правильне змішування кольорів, освітлення та текстури.
7. Коли всі об'єкти на сцені були відображені, Unity закінчує процес рендерингу та відображення кадра, і передає кадр на вивід.

Для кожного об’єкта та матеріалу на об’єкті викликається метод «Draw», тому об’єднуючи об’єкти та їх матеріали в один mesh, ми можемо зменшити кількість викликів цього методу. Адже тепер ми маємо один об’єкт, і всі повторно використані матеріали цих об’єктів об’єднуються в один, що дозволяє уникнути дублювання матеріалів і так само зменшити кількість «Draw». Саме для цього і потрібен Mesh baker, щоб зробити метод «Draw» найменш затратним.

Також Mesh baker надає різноманітні функції, включаючи підтримку скелетних meshes, генерацію атласів текстур та груп рівнів деталізації (LOD). Він також включає інструменти для оптимізації UV, генерації lightmap та налаштування процесу злиття, щоб він відповідав конкретним вимогам проєкту.

Ще один важливий аспект MeshBaker - це його можливість оптимізувати кількість вершин, які обробляються при рендерингу, що покращує продуктивність гри. У іграх, особливо з великою кількістю об'єктів, кожна вершина може значно вплинути на продуктивність, особливо при рендерингу на слабших пристроях. Тому об'єднання кількох meshes у один оптимізований mesh може бути дуже корисним для оптимізації продуктивності гри.

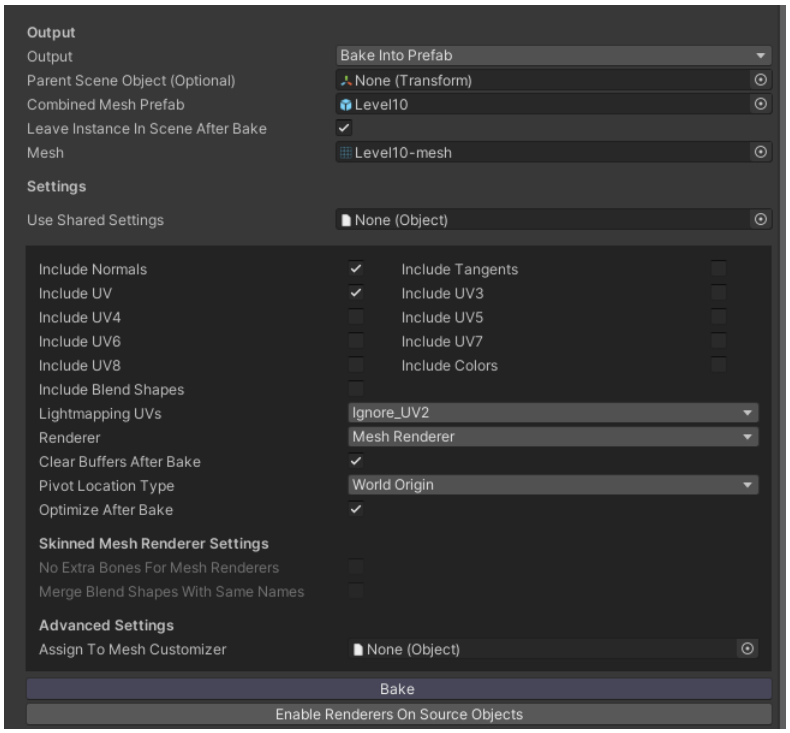


Рис. 3.14. Основні налаштування Mesh baker

3.3.3 Створення спільної текстури

Для того, щоб створити спільну текстуру спочатку необхідно записати усі кольори, які використовують наші об’єкти, після чого розмістити ці кольори на нашій текстурі. Ця текстура була продемонстрована у 2.3 пункті. Після цього необхідно для усіх об’єктів створити розгортку, використовуючи інструменти UV редагування у програмі blender та розташувати вершини об’єкта на необхідному нам місці у текстурі.

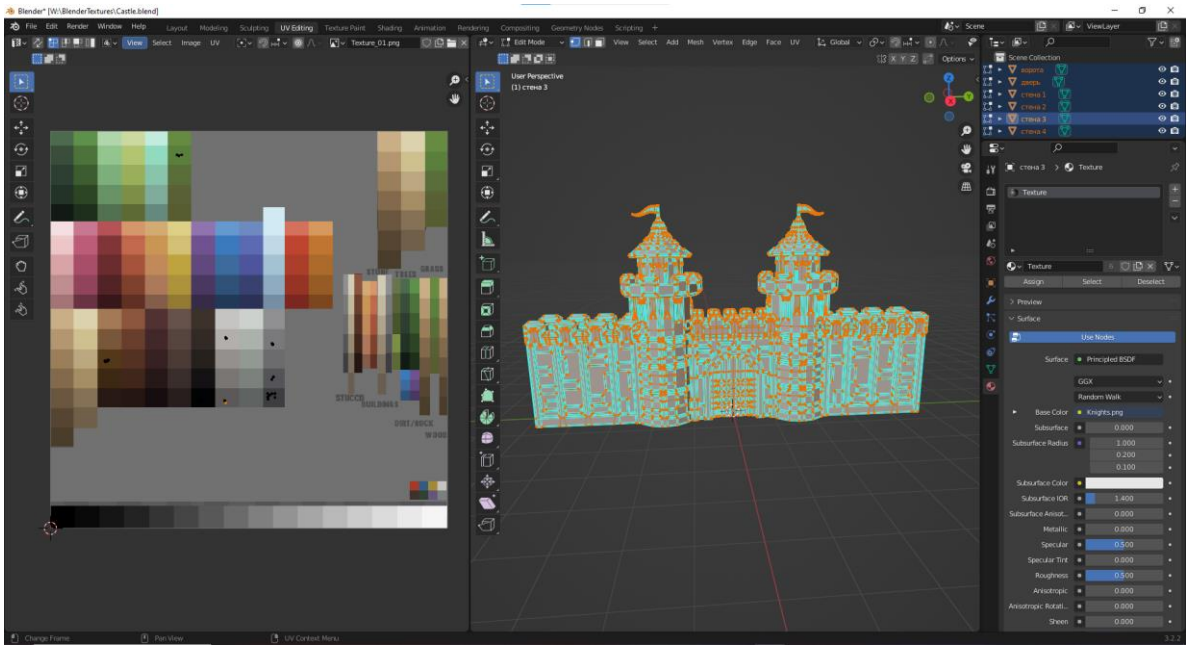


Рис. 3.15. розташування вершин об’єкту на текстурі використовуючи UV

Коли усі вершини були розміщені, необхідно видалити усі матеріали, імпортувати об’єкт, та додати йому матеріал, який використовує необхідно текстуру. Відтепер метод Draw, буде викликатись 1 раз для кожного об’єкта, поки вони не будуть об’єднані в один mesh завдяки Mesh baker.

Після додавання усіх об’єктів, та об’єднання їх в один mesh, unity буде викликати лише 1 метод Draw, адже у всіх об’єктів один матеріал, та зберігаються вони в одному mesh.

Також для елементів інтерфейсу, можливо зробити текстурний atlas, тобто велику текстуру, що містить безліч маленьких зображень, які називають

текстурними під регіонами. Замість того, щоб завантажувати безліч окремих текстур для кожного об'єкта в грі, можливо завантажити одну текстуру-атлас, яка буде містити усі необхідні текстури, а потім використовувати під регіони цієї текстури для кожного об'єкта.

Використання атласів текстур має кілька переваг. По-перше, це дозволяє істотно зменшити кількість текстур, що завантажуються, і прискорити завантаження гри. По-друге, це дозволяє зменшити кількість викликів до відеокарти для зміни текстур, оскільки всі необхідні текстури вже завантажені на згадку. Нарешті, використання атласів текстур може зменшити використання оперативної пам'яті, оскільки кожна текстура займає певний обсяг пам'яті, а використання атласу дозволяє об'єднати кілька текстур однією.

Unity має вбудований інструмент для створення текстурних атласів [16], який дозволяє об'єднувати кілька текстур в одну велику текстуру. Цей інструмент називається Sprite Packer (пакувальник спрайтів). Щоб використовувати його, вам необхідно позначити ваші текстури як спрайти та додати їх до списку упаковування спрайтів, після чого Unity автоматично збере їх у єдиний текстурний атлас.

Рис. 3.16. Демонстрація текстурного Atlas

3.4 Тестування розробленої гри

Тестування гри на ігровому двигуні Unity - це важлива частина розробки будь-якої гри. Тестування допомагає виявити й виправити помилки та недоліки гри, що можуть вплинути на її геймплей та роботу.

У Unity тестування гри зазвичай проводиться на ранніх стадіях розробки, коли гра не повністю готова, але вже містить базовий функціонал. Тестування можна проводити на декількох рівнях:

1. Тестування функціонала: на цьому етапі тестується робота окремих елементів гри, таких як рух персонажів, їх анімації, робота інтерфейсу тощо.
2. Тестування ігрового процесу: на цьому етапі тестується геймплей гри в цілому, включаючи взаємодію гравця з оточенням, бойову систему, місії та завдання, рівень складності тощо.
3. Тестування продуктивності: на цьому етапі тестується продуктивність гри, включаючи швидкість завантаження, кадрову частоту та оптимізацію роботи гри на різних платформах.

Для тестування гри на Unity зазвичай використовуються спеціальні інструменти, такі як Unity Test Runner, Unity Profiler та Unity Remote. Unity Test Runner дозволяє створювати та запускати тести для перевірки роботи окремих елементів гри. Unity Profiler допомагає виявляти та виправляти помилки, що впливають на продуктивність гри. Unity Remote дозволяє тестувати гру на різних пристроях без необхідності її кожен раз перекомпілювати та встановлювати.

Також для тестування гри на Unity можна використовувати різноманітні зовнішні інструменти, такі як тестувальні фреймворки та бібліотеки. Наприклад, для автоматизації тестування можна використовувати фреймворки, такі як Unity Test Runner, NUnit або Xunit. Ці фреймворки дозволяють створювати тести, які можуть бути виконані автоматично під час розробки гри.

Також існують спеціальні інструменти для аналізу продуктивності гри, які допомагають виявити можливі проблеми з продуктивністю та оптимізувати гру. Одним з таких інструментів є Unity Profiler, який дозволяє аналізувати роботу гри та знаходити проблемні місця. Він дає можливість відстежувати та аналізувати фреймворк, CPU, GPU та інші види завантаження, що допомагає виявити буття в продуктивності гри та вирішити їх. Profiler дає можливість відстежувати багато параметрів, таких як кількість виконаних рендерингів, кількість виконаних скриптів, зайняте CPU та GPU, час виконання фреймів, час виконання окремих методів та багато іншого.

Разом з Unity profiler іноді використовується Frame Debugger. Frame Debugger - це інструмент, що дозволяє аналізувати кожен кадр гри, відстежувати всі зміни, які відбуваються в грі, та визначати, які процеси займають більше ресурсів. Цей інструмент доступний в Unity і дозволяє розглядати гру в режимі реального часу.

Frame Debugger дозволяє відстежувати наступні елементи гри:

1. Положення і розмір об'єктів.
2. Кількість вершин, примітивів, матеріалів та текстур, що використовуються.
3. Кількість draw call та наскільки вони впливають на продуктивність.
4. Кількість рендер-текстур, які створюються та використовуються.
5. Кількість shadow casters та shadow receivers, які впливають на продуктивність.
6. Використання освітлення та ефектів.

За допомогою Frame Debugger можна визначити, які частини гри вимагають найбільше ресурсів, що можна покращити, щоб гра працювала більш ефективно та без затримок. Також цей інструмент дозволяє виявити помилки та неполадки, які можуть виникнути в процесі розробки гри.

Для тестування гри на різних пристроях можна використовувати Unity Remote, який дозволяє відображати зображення з пристрою на комп'ютері та взаємодіяти з грою за допомогою комп'ютера.

Крім того, для забезпечення якості гри можна використовувати сервіси для автоматичного тестування, такі як Unity Cloud Build, який дозволяє автоматично збирати та тестувати гру на різних пристроях та платформах.

Для тестування гри «Tower Dimension» було використано емулятор для Android «BlueStacks 5». BlueStacks 5 є популярним емулятором Android, який може бути використаний для запуску та тестування ігор на комп'ютері.

Рис. 3.17. Демонстрація емулятора «BlueStacks 5»

Цей емулятор дозволяє запустити більшість ігор, які доступні для Android-платформи, на вашому комп'ютері з використанням клавіатури та миші. BlueStacks 5 підтримує широкий спектр графічних налаштувань, таких як висока роздільна здатність, максимальна кількість FPS і підтримка різних типів графічних карт.

Цей емулятор має простий інтерфейс, який дозволяє легко налаштувати його для тестування конкретної гри. Він також підтримує багатокористувацький режим, що дозволяє грати у гру разом з друзями або іншими користувачами зі всього світу.

Крім того, BlueStacks 5 має додаткові функції, такі як можливість запису гри, швидкі клавіші, мульти-інстанс, що дозволяє запустити кілька екземплярів

BlueStacks на одному комп'ютері, та інші. Ці функції можуть бути корисними для розробників, які хочуть протестувати свої ігри в різних умовах або використовують додаткові програми для запису чи трансляції своїх ігор.

Ось ще деякі ключові можливості BlueStacks 5 для тестування ігор:

1. Інтуїтивний та легкий у використанні інтерфейс, який дозволяє швидко налаштувати емулятор та керувати його параметрами. Ви можете встановлювати роздільність екрана, кількість ядер процесора, обсяг оперативної пам'яті та інші налаштування, щоб переконатися, що гра працює оптимально.
2. Зручні інструменти розробника: BlueStacks 5 надає ряд корисних інструментів для розробників, які спрощують процес тестування. Це включає можливість збереження знімків екрана, запису відео з екрана, симуляцію різних режимів сенсорного введення (таких як дотик, жести, сенсорний екран) та інші.
3. Синхронізація з мобільними пристроями: BlueStacks 5 дозволяє синхронізувати ваші ігрові дані з мобільними пристроями. Це означає, що ви можете синхронізувати ваші ігрові дані з мобільними пристроями, що дозволяє вам продовжувати гру на BlueStacks 5 там, де ви закінчили на мобільному пристрої й навпаки. Це дозволяє зручно перемикатись між різними платформами та зберігати прогрес у ваших іграх.

У загальному, BlueStacks 5 може бути корисним інструментом для тестування ігор, особливо якщо ви хочете випробувати гру на більшому екрані та з використанням клавіатури та миші.

Загалом, тестування гри на Unity - це важливий етап розробки, який допомагає виявити та виправити помилки, забезпечити якість та оптимізувати гру для різних пристроїв та платформ.

ВИСНОВКИ

Отже, після проведення аналізу додатків та ігор, можливо зробити висновок, що актуальність ігор ніколи не зникне, адже додатки стали частиною сучасного життя людини. Ігри є не лише засобом розваги, але й можуть забезпечити людей задоволенням, емоційним досвідом та навчити їх нового, або змусити задуматися про щось глибше. Саме тому сфера розробки ігрових додатків є однією з найбільш прибуткових галузей сучасної економіки, які швидко зростають.

Під час написання дипломної роботи було ретельно досліджено процес створення гри та проаналізовано всі його складові етапи. Особлива увага була приділена до реалізації ігрових можливостей завдяки мові програмування «C#». Середовище «Unity» стало надійним та доступним інструментом для реалізації проєкту та усіх задуманих функцій. Всі графічні елементи було створено за допомогою «Blender» та були оптимізовані для мобільної гри.

Підсумовуючи вищесказане, можна зробити висновок, що Blender, як потужний інструмент для моделювання та створення 3D-об’єктів, дозволив створити дивовижну графіку та анімацію для нашого проєкту. Спільно з Unity, що є високоякісним движком для розробки ігор, було реалізовано всі задумані функції та створено професійний геймплей, за допомогою мови програмування C#, що є основною мовою розробки на платформі Unity.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. *TETRIS and Other Videogames' Effects on PTSD [Електронний ресурс] // CHC Game Studies. – 2013. – Режим доступу до ресурсу: https://chcgamestudies.wordpress.com/2013/03/18/2973/.*
2. *Plants vs Zombies™ 2 [Електронний ресурс] // PopCap Games. – 2013. – Режим доступу до ресурсу: https://play.google.com/store/apps/details?id=com.ea.game.pvz2_row&hl=ru&gl=US*
3. *TowerMadness 2 [Електронний ресурс] // Limbic Software. – 2014. – Режим доступу до ресурсу: https://play.google.com/store/apps/details?id=com.limbic.towermadness2&hl=ru&gl=US*
4. *C# documentation [Електронний ресурс] // Microsoft. – 2023. – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/dotnet/csharp/.*
5. *Mark J. P. C# 8.0 and .NET Core 3.0 – Modern Cross-Platform Development / Price Mark J.., 2019. – 800 с*
6. *Unity Overview [Електронний ресурс] // Unity – Режим доступу до ресурсу: https://docs.unity3d.com/510/Documentation/Manual/UnityOverview.html.*
7. *Unity Asset Store [Електронний ресурс] // Unity – Режим доступу до ресурсу: https://assetstore.unity.com.*
8. *Blender 3.5 Reference Manual [Електронний ресурс] // Blender – Режим доступу до ресурсу: https://docs.blender.org/manual/en/latest/.*
9. *Blender Guru [Електронний ресурс]. – 2009. – Режим доступу до ресурсу: https://www.blenderguru.com/.*
10. *Visitor [Електронний ресурс] // Refactoring Guru – Режим доступу до ресурсу: https://refactoring.guru/design-patterns/visitor*
11. *Audio [Електронний ресурс] // Unity Manual – Режим доступу до ресурсу: https://docs.unity3d.com/Manual/class-AudioListener.html.*
12. *Georgios K. Improving Mobile Game Performance with Basic Optimization Techniques in Unity [Електронний ресурс] / K. Georgios, X. Stelios // MDPI. – 2022. – Режим доступу до ресурсу: https://www.mdpi.com/2673-3951/3/2/14.*
13. *Object Pool Design Pattern [Електронний ресурс] // Source Making. – Режим доступу до ресурсу: https://sourcemaking.com/design_patterns/object_pool*
14. *Mesh Baker Manual [Електронний ресурс] // Digital Opus – Режим доступу до ресурсу: https://digitalopus.ca/site/mesh-baker-3-manual/.*

15. *Manjil T. Optimizing Game using Unity Draw Call Management [Електронний ресурс] / Thapa Manjil // Yarsa Labs DevBlog. – 2021. – Режим доступу до ресурсу: https://blog.yarsalabs.com/optimize-game-using-draw-call-management/.*
16. *Prerak G. Game Optimization Using Unity Sprite Atlas [Електронний ресурс] / Giri Prerak // Yarsa Labs DevBlog. – 2022. – Режим доступу до ресурсу: https://blog.yarsalabs.com/game-optimization-using-unity-sprite-atlas/.*
17. *Head First Design Patterns / F.Eric, R. Elisabeth, B. Bert, S. Kathy., 2004. – 688 с. – (O'Reilly Media, Inc.). – ISBN: 9780596007126*
18. *Steve M. Code Complete 2nd Edition / McConnell Steve., 2004. – 960 с. – (Cisco Press). – (ISBN: 9780735619678; кн. 2)*

Додаток А

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО- НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

РОЗРОБКА ГРИ В ЖАНРІ «TOWER DEFENSE» З ВИКОРИСТАННЯМ RPG МЕХАНІК МОВОЮ C#

Виконав студент 4 курсу
групиПД-43
Бойко Микита Сергійович
Керівник роботи
доктор філософії, доцент кафедри ІПЗДібрівний Олесь Андрійович

Київ – 2023

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи** – підвищення зацікавленості геймплеєм гри в жанрі «Tower Defense» гравців на мобільних пристроях за рахунок впровадження механік «Дерево діалогів » та «Дерево пасивних умінь ».

- **Об’єкт дослідження** – геймплей гри жанру «Tower Defense».

- **Предмет дослідження** – гра жанру «Tower Defense» з елементами RPG для мобільних пристроїв .

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Підбір науково-технічної літератури.
2. Аналіз основних концепцій Tower Defense.
3. Розробка структури та компонентів гри.
4. Моделювання об’єктів гри.
5. Створення текстур для ігрових компонентів та розробка дизайну рівнів.
6. Оптимізування гри під мобільні пристрої.
7. Тестування та документація гри.

3

АНАЛОГИ

Особливість	 Plants vs Zombies 2	 Tower Madness 2	 Infinitode2	 Tower Dimension
Різноманітні вороги	+	-	-	+
Різноманітні вежі	+	-	+	+
Велика кількість рівнів	+	+	-	-
Опис ворогів	-	-	+	+
Нескінчений режим	-	-	+	+
Сюжетний режим	+	-	-	+
3D графіка	-	+	-	+
Професійний UI	+	-	-	-
Унікальні механіки жанру	-	-	-	+

4

ВИМОГИ ДО ІГРОВОГО КОНТЕНТУ

1. Наявність сюжетного та нескінченного ігрового режиму.
2. Представлення внутрішньоігрового магазину з покращеннями веж та щоденних нагород у головному меню гри.
3. Використання різноманітних веж зі своїми унікальними типами атаки.
4. Використання різноманітних ворогів з унікальними здібностями.
5. Використання ігрової механіки «Дерево діалогів» в сюжетному режимі.
6. Представлення вибору рівнів та скасування наслідків виборів під час діалогів в сюжетному режимі завдяки завантаженню рівня.

5

КОНЦЕПТ ГРИ

1. У магічному світі до планети людей вторглись орки, які намагаються захопити світ людей. Користувач буде грати за молодого командира, який має вирішувати, як діяти, кому вірити та як боротись з орками.
2. Основний геймплей містить побудову оборонних веж та використання пасток для захисту свого замку.
3. Геймплей у сюжетному режимі додатково має систему виборів під час діалогів, які будуть впливати на геймплей та кінець історії. Якщо користувач буде незадоволений наслідками вибору, він зможе завантажити необхідний рівень та змінити свій вибір.
4. Гра містить внутрішньоігровий магазин, в якому гравець може покращувати свої вежі. Це допоможе гравцю набирати рекордні бали у нескінченному режимі та полегшить сюжетний режим.
5. Кожен день гравець може отримувати золото з щоденних нагород, та отримувати золото за пройдений рівень. Користувач зможе витрати це золото на покращення веж у магазині.

6

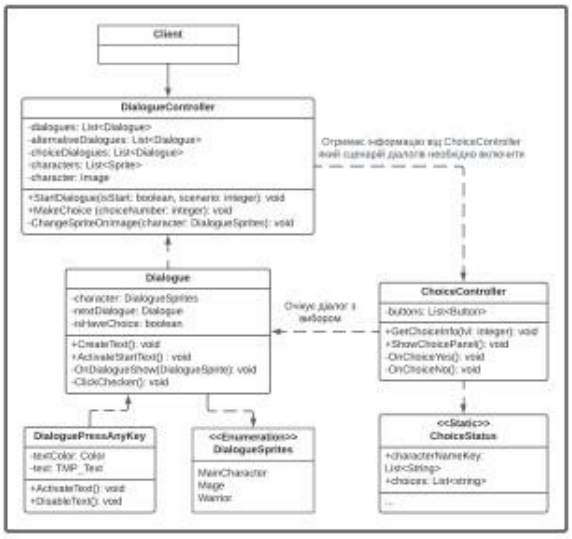
ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ





7

ОПИС ІГРОВОЇ МЕХАНІКИ «ДЕРЕВО ДІАЛОГІВ»

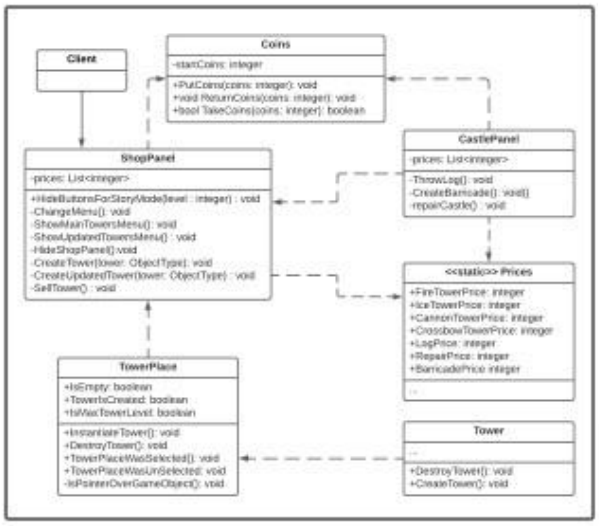


Діаграма класів механіки «Дерево діалогів»

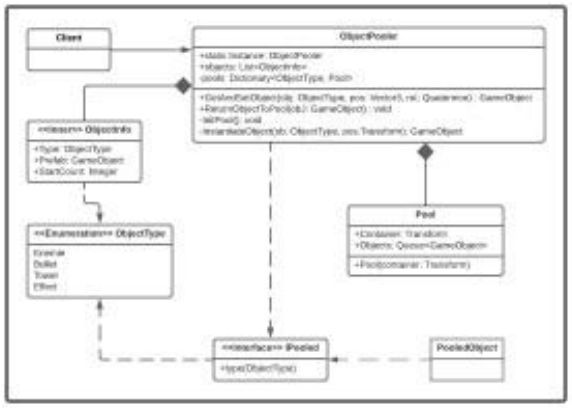
Блок-схема роботи алгоритму механіки «Дерево діалогів»

8

ДІАГРАМИ КЛАСІВ СТВОРЕННЯ ВЕЖ



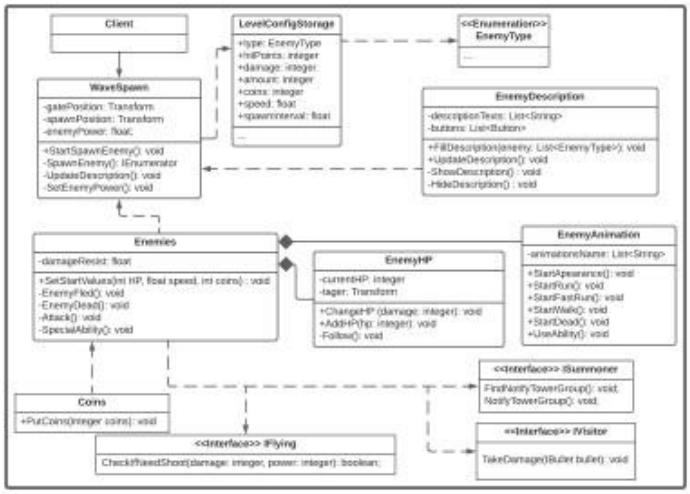
Механізм створення ігрових веж



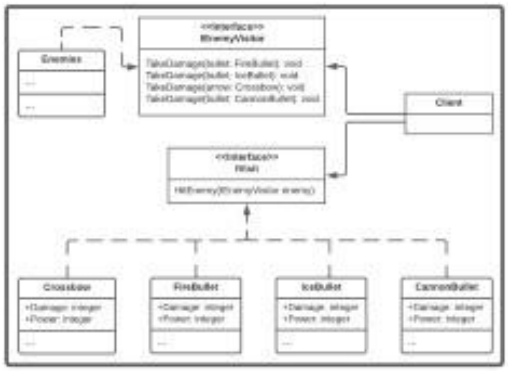
Реалізація шаблону проєктування «Object Pool» для створення ігрових об’єктів

9

ДІАГРАМИ КЛАСІВ ВОРОГІВ



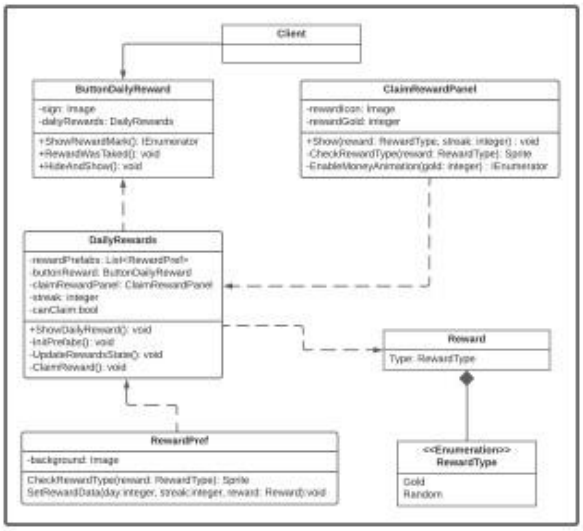
Алгоритм створення та поведінки ворогів



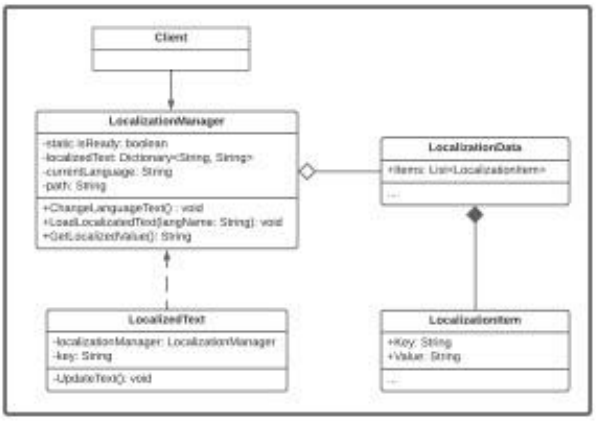
Реалізація шаблону проєктування «Visitor» для взаємодії ворогів з різними джерелами пошкоджень

10

ДІАГРАМИ КЛАСІВ ГОЛОВНОГО МЕНЮ



Діаграма класів механіки «Щоденна нагорода»



Діаграма класів відповідальних за локалізацію

11

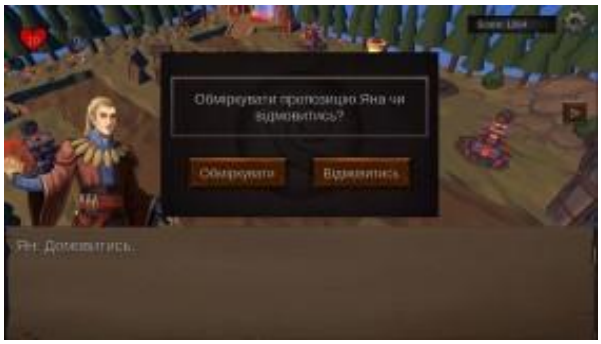
ЕКРАННІ ФОРМИ

Головне меню гри

Геймплей гри

12

ЕКРАННІ ФОРМИ

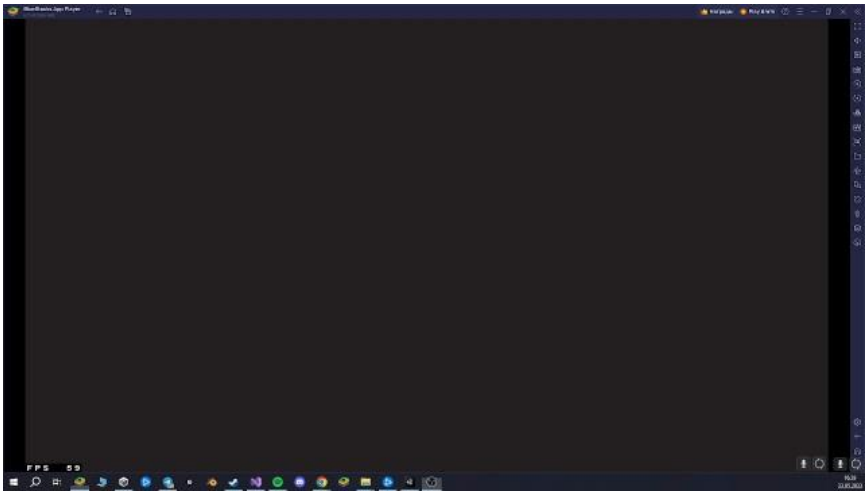


Діалогова гілка

Внутрішньоігровий магазин

13

ЕКРАННІ ФОРМИ



Відеодемонстрація гри в емуляторі «Bluestacks 5»

14

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Бойко М.С Застосування шаблону проектування visitor на двигуні Unity для обробки взаємодії об’єктів гри // Застосування програмного забезпечення в ІКТ. Збірник тез. 20.04.2023, ДУТ, м. Київ – К.: ДУТ, 2023. – С. 114

2. Бойко М.С Використання алгоритму A* для штучного інтелекту мобів у грі на Unity // ІІІ Всеукраїнська Науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті». Збірник тез 16.05.2023, ДУТ, м. Київ К.: ДУТ, 2023. – подано до друку.

ВИСНОВКИ

1. Підібрано науково-технічну літературу.
2. Проведено аналіз основних концепцій жанру «Tower Defense».
3. Досліджено весь процес створення гри від алгоритмів до створення об’єктів. Розроблено архітектуру та основні алгоритми гри жанру «Tower Defense» з додатковими механіками «Дерево діалогів» та «Дерево пасивних умінь».
4. Створено основні 3D об’єкти гри, такі як вежі, вороги та елементи декору завдяки програмного забезпечення «Blender».
5. Створено основний дизайн рівнів та ігрові текстури для об’єктів гри. Було розташовано розгортки об’єктів на необхідному кольорі спільної текстури, завдяки чому було зменшено кількість матеріалів в грі.
6. Гру оптимізовано завдяки загальнодоступних інструментів, таких як текстурний атлас, спільна текстура та склеювання об’єктів завдяки ассету «MeshBaker».
7. Проведено тестування гри завдяки мобільному емулятору «Bluestacks 5».

ДЯКУЮ ЗА УВАГУ!

Додаток Б

```
1.
   public interface IEnemyVisitor
2. {
3. void TakeDamage(FireBullet bullet);
4. void TakeDamage(IceBullet bullet);
5. void TakeDamage(CannonBullet bullet);
6. void TakeDamage(CrossbowBullet bullet);
7. void TakeDamage(CastleLogic.Barricade barricade);
8. void TakeDamage(UI.LevelController levelController);
9. void TakeDamage(BomberExplosion bomberExplosion);

10.     void TakeDamage(CastleLogic.LogWithSpikes logWithSpikes);
11.     }

12.     public abstract class Enemies : MonoBehaviour, IEnemyPooled
13.     {
14.     [SerializeField] protected
      EnemyPooler.ObjectInfo.EnemyObjectType _enemyName;
  [SerializeField] protected NavMeshAgent _navMeshAgent;
  protected CastleLogic.Castle _gate;
  protected EnemyHitPoints _hitPoints;
  protected const float _minDistance = 0.5f;
  protected float _freezeImmunity = 0.8f;
  protected float _frezeTime = 5f;
  protected float _unFrezeeTime;
  protected float _startSpeed;
  protected int _fireDamageDifference = 2;
  protected int _coins;
  protected int _maxHP;
  protected bool _isBoss;
  protected bool _sizeWasChanged = false;
  protected Action OnEnemyDied;
  static protected Transform _defenseGate;
  static protected Coins _coinsContainer;
  static protected bool _levelIsEndless;
  static public float AdditionalSize;

  public EnemyPooler.ObjectInfo.EnemyObjectType
  Type => _enemyName;

  protected virtual void Awake()
  {
      OnEnemyDied += SimpleKillEnemy;
  }

public static void SetGeneralEnemySettings(Transform
defenseCastle,    Coins coinsContainer, bool levelIsEndless)
```

```
{
    _defenseGate = defenseCastle;
    _coinsContainer = coinsContainer;
    _levelIsEndless = levelIsEndless;
}

protected virtual void SimpleKillEnemy()
{
    _coinsContainer.PutCoins(_coins);
    EnemyPooler.Instance.DestroyEnemyObject(gameObject);
         OtherPooler.Instance.DestroyObject(_hitPoints.gameObject);
    if (!_levelIsEndless) EnemyCounter.RemoveEnemy();
}
protected virtual IEnumerator UnFreeze()
{
    while (_unFrezeeTime > 0)
    {
        yield return new WaitForSeconds(1f);
        _unFrezeeTime--;
    }

    while (_navMeshAgent.speed < _startSpeed)
    {
        _navMeshAgent.speed /= _freezeImmunity;
    }
}

protected virtual void Freeze()
{
    if (!IsInvoking("UnFreeze"))
        StartCoroutine(UnFreeze());
    _unFrezeeTime = _frezeTime;

    if (_navMeshAgent.speed * _freezeImmunity > 0.2f)
        _navMeshAgent.speed *= _freezeImmunity;
}

protected virtual void TakeSimpleDamage(int damage)
{
    bool isAlive = _hitPoints.ChangeHP(damage);
    if (!isAlive) { OnEnemyDied?.Invoke(); }
}



protected virtual void TakeFreezeDamage(int damage, int
freezePower = 0)
{
     bool isAlive = _hitPoints.ChangeHP(CalculatePowerDamage(damage,
     freezePower));
```

```
        if (!isAlive) OnEnemyDied?.Invoke();
        else
        {
            for (int i = 0; i < freezePower; i++)
            {
                Freeze();
            }
        }
    }
        protected virtual IEnumerator TakeFireDamage(int damage,
        int power)
        {
            int fireDamage = CalculatePowerDamage(damage, power);
            while (fireDamage > 0)
            {
                if (gameObject.active)
                {
                    TakeSimpleDamage(fireDamage);
                    fireDamage -= _fireDamageDifference;
                    yield return new WaitForSeconds(0.15f);
                    if (fireDamage <= _fireDamageDifference)
                    {
                        for (int i = 0; i < 5; i++)
                        {
                            if (gameObject.active)
                            {
                                TakeSimpleDamage(fireDamage);
                                yield return new WaitForSeconds(0.15f);
                            }
                            else break;
                        }
                        break;
                    }
                }
                else break;
            }
        }

    protected int CalculatePowerDamage(int damage, int power)
    {
        int returnedDamage = damage;



        for (int i = 1; i < power; i++)
        {
            returnedDamage *= 2;
        }
        return returnedDamage;
    }
```

```
    public virtual void EnemyFled()
    {
        EnemyPooler.Instance.DestroyEnemyObject(gameObject);
        Destroy(_hitPoints.gameObject);
        if (!_levelIsEndless) EnemyCounter.RemoveEnemy();
    }

    public abstract void EnemyNearToGate(CastleLogic.Castle gate);
}


public abstract class EnemiesWithAnimation : Enemies
 {
     [SerializeField] protected EnemyAnimation _animation;
     [SerializeField] protected Collider _collider;
     [SerializeField] protected int _damage = 1;
     private bool _isFreezing = false;
     protected override void SimpleKillEnemy()
     {
         EnemyPooler.Instance.DestroyEnemyObject(gameObject);
     }

     protected virtual void PrepafeForDead()
     {
         OnEnemyDied = null;
         _collider.enabled = false;
         StopAllCoroutines();
         _animation.Dead();
         _navMeshAgent.speed = 0;
         _coinsContainer.PutCoins(_coins);
         OtherPooler.Instance.DestroyObject(_hitPoints.gameObject);
         if (!_levelIsEndless) EnemyCounter.RemoveEnemy();
         _navMeshAgent.enabled = false;
     }

     protected virtual void GeneralOnEnable()
     {
         _collider.enabled = true;
         _animation.SetDead(SimpleKillEnemy);
         OnEnemyDied = PrepafeForDead;
     }

      public virtual void SetStartValues(EnemyHitPoints hitPoins,     int
HP, float speed, int coins, bool isBoss = false)
      {
           _navMeshAgent.enabled = true;
           _navMeshAgent.SetDestination(_defenseGate.position);
      _hitPoints = hitPoins;
           _hitPoints.CreateHP(gameObject, HP);
```

```
            _startSpeed = speed; _navMeshAgent.speed = _startSpeed;
        _coins = coins; _isBoss = isBoss; _maxHP = HP;
            _animation.StartMove(speed);
        }
        protected override IEnumerator UnFreeze()
        {
           while (_unFrezeeTime > 0)
         {
             yield return new WaitForSeconds(1f);
             _unFrezeeTime--;
         }

        while (_navMeshAgent.speed < _startSpeed)
        {
            _navMeshAgent.speed /= _freezeImmunity;
        }

        _animation.StartMove(_navMeshAgent.speed);
        _isFreezing = false;
        }

        protected override void Freeze()
        {
            _unFrezeeTime = _frezeTime;
            if (!_isFreezing)
            {
            StartCoroutine(UnFreeze());
            _isFreezing = true;
            }
            if (_navMeshAgent.speed * _freezeImmunity > 0.2f)
            {
            _navMeshAgent.speed *= _freezeImmunity;
            _animation.StartMove(_navMeshAgent.speed);
            }

        }

        protected virtual void Attack()
        {
            if (!_gate.TakeHit(_damage))
            EnemyFled();
        }

           public override void EnemyNearToGate(CastleLogic.Castle
gate)
           {
               _gate = gate;
               _navMeshAgent.speed = 0f;
               _animation.StartAttack(Attack);
               _navMeshAgent.enabled = false;
```

```
        }
}

public interface IFlyEnemy
{
    bool CheckIfNeedShot(int damage, int power);
}

public interface ISummoner
{
    void FindNotifyTowerGroup();
    void NotifyTowerGroup();
}


public class EnemyAnimation : MonoBehaviour
{
    [SerializeField] private Animator _animator;
    [SerializeField] private string _appearance;
    [SerializeField] private string _walk;
    [SerializeField] private string _run;
    [SerializeField] private string _fastRun;
    [SerializeField] private string _dead;
    [SerializeField] private string _ability;
    [SerializeField] private string _attack;

    private Action _onAbilityCasted;
    private Action _onEnemyDead;
    private Action _onEnemyAppear;
    private Action _onEnemyAttack;

    private void AnimationEventDead()
    {
        _onEnemyDead?.Invoke();
    }
    private void AnimationEventUseAbility()
    {
        _onAbilityCasted?.Invoke();
    }

    private void AnimationEventAppear()
{
    _onEnemyAppear?.Invoke();
}

private void AnimationEventAttack()
{
    _onEnemyAttack?.Invoke();
}
```

```
public void Appear(Action appearEvent)
{
    _animator.SetTrigger(_appearance);
    _onEnemyAppear = appearEvent;
}
public void StartMove(float speed)
{
    _animator.ResetTrigger(_walk);
    _animator.ResetTrigger(_run);
    _animator.ResetTrigger(_fastRun);

    if (speed >= 3f)
    {
        if (_fastRun != "")
        {
            _animator.SetTrigger(_fastRun);
        }
        else
        {
            _animator.SetTrigger(_run);
        }
    }
    else if (speed >= 1.3f)
    {
        if (_run != "")
        {
            _animator.SetTrigger(_run);
        }
        else
        {
            _animator.SetTrigger(_walk);
        }
    }
    else
    {
        _animator.SetTrigger(_walk);
    }
}

public void SetDead(Action deadEvent)
    {
        _onEnemyDead = deadEvent;
    }

    public void AddDeadAction(Action deadEvent)
    {
        _onEnemyDead += deadEvent;
    }
    public void Dead()
    {
```

```
        _animator.SetTrigger(_dead);
    }

    public void UseAbility(Action ability)
    {
        _onAbilityCasted = ability;
        _animator.SetTrigger(_ability);
    }
    public void UseAbility()
    {
        _animator.SetTrigger(_ability);
    }

    public void SetAbility(Action ability)
    {
        _onAbilityCasted = ability;
    }

    public void StartAttack(Action attack)
    {
        _onEnemyAttack = attack;
        _animator.SetTrigger(_attack);
    }

}

public class EnemyHitPoints : MonoBehaviour, IOtherPooled
{
    [SerializeField] private
    OtherPooler.ObjectInfo.ObjectType _hitPointsType;
    private int _currentHitPoints;

    private static Vector3 _offset = new Vector3(0, 1.35f, 0);

    private GameObject _enemy;

    public static void UpdateHPOffset(float offset)
    {

       if(offset == 1.5f)
            _offset = new Vector3(0, 1.35f + offset / 3, 0);
        else
            _offset = new Vector3(0, 1.35f + offset / 1.5f, 0);
    }
    public static void SetDefaultHPOffset()
    {
        _offset = new Vector3(0, 1.35f, 0);
    }
    public OtherPooler.ObjectInfo.ObjectType
    Type => _hitPointsType;
```

```
    public void CreateHP(GameObject enemy, int maxHitPoints)
    {
        _enemy = enemy;
        _currentHitPoints = maxHitPoints;
    }
    private void Update()
    {
        GetComponent<RectTransform>().position =
        Camera.main.WorldToScreenPoint(_enemy.transform.position
        + _offset)
        GetComponent<TMP_Text>().text =
        _currentHitPoints.ToString();
    }

    public bool ChangeHP(int damage)
    {
        _currentHitPoints -= damage;
        if (_currentHitPoints <= 0) return false;
        return true;
    }

    public void AddHitPoins(int hitPoints)
    {
        _currentHitPoints = hitPoints;
    }

    public int GetHPValue() { return _currentHitPoints; }
}
```

```
public static class EnemyCounter
{
    public static int CurrentEnemyAmount = 0;
    public static int EnemyCount = 0;
    public static int CurrentWaveLevel = 1;

    public static Action<Enemies> OnEnemyWasCounted;
    public static void ResetWaveLevel()
    {
        CurrentWaveLevel = 1;
    }

    public static void AddEnemy(Enemies enemy)
```

```
    {
        CurrentEnemyAmount++;
        OnEnemyWasCounted?.Invoke(enemy);
    }

    public static void RemoveEnemy()
    {
        CurrentEnemyAmount--;
        EnemyCount--;
        if (EnemyCount == 0) WaveSpawn.NoEnemyOnLevel();
    }

    public static void CountEnemy(int enemy)
    {
        EnemyCount += enemy;
    }

    public static bool HasEnemyOnGame()
    {
        if (CurrentEnemyAmount == 0) return false;
        else return true;
    }

    public static bool HasEnemy()
    {
        if (EnemyCount == 0) return false;
        else return true;
    }

    public static void ResetEnemyCount()
    {
        CurrentEnemyAmount = 0;
        EnemyCount = 0;
    }
}

public class LocalizationManager : MonoBehaviour
{
    private Dictionary<string, string> _localizedText;
    private string _currentLanguage;
    private static bool _isReady = false;

    public delegate void ChangeLangText();
    public event ChangeLangText OnLanguageChanged;

    private void Awake()
    {
        if (!PlayerPrefs.HasKey(PlayerPrefsKeyContainer.Language))
        {
```

```
            if (Application.systemLanguage ==
                SystemLanguage.Russian ||
                Application.systemLanguage ==
                SystemLanguage.Belarusian)
                 PlayerPrefs.SetString(
                PlayerPrefsKeyContainer.Language, "ru_RU");
            else if (Application.systemLanguage ==
                SystemLanguage.Ukrainian)
                     PlayerPrefs.SetString(
                PlayerPrefsKeyContainer.Language, "uk_UK");
            else
                PlayerPrefs.SetString(
               PlayerPrefsKeyContainer.Language, "en_US");
        }
        _currentLanguage = PlayerPrefs.GetString
        (PlayerPrefsKeyContainer.Language);
        LoadLocalizedText(_currentLanguage);
    }

     public void LoadLocalizedText(string langName)
     {
         string path = Application.streamingAssetsPath +
"/Languages/" + langName + ".json";
         string dataAsJson;
         // HACK Iphone == RuntimePlatform.IPhonePlayer
         if (Application.platform == RuntimePlatform.Android)
         {
             #region MyTry
             path = Path.Combine(Application.streamingAssetsPath,
"Languages/" + langName + ".json");
             #endregion
             WWW reader = new WWW(path);
             while (!reader.isDone) { }

             dataAsJson = reader.text;
         }
            else
            {
                dataAsJson = File.ReadAllText(path);
            }

            LocalizationData loadedData =
JsonUtility.FromJson<LocalizationData>(dataAsJson);

            _localizedText = new Dictionary<string, string>();
            for (int i = 0; i < loadedData.Items.Length; i++)
            {
                _localizedText.Add(loadedData.Items[i].Key,
loadedData.Items[i].Value);
            }
```

```
            PlayerPrefs.SetString(PlayerPrefsKeyContainer.Language,
langName);
            _currentLanguage =
PlayerPrefs.GetString(PlayerPrefsKeyContainer.Language);
            _isReady = true;
            OnLanguageChanged?.Invoke();
        }
        public string GetLocalizedValue(string key)
        {
            if (_localizedText.ContainsKey(key)) return
_localizedText[key];
            else
                throw new Exception("Localized text with key \"" + key +
"\" not found");
        }
        public string CurrentLanguage
        {
            get
            {
                return _currentLanguage;
            }
            set
            {
                LoadLocalizedText(value);
            }
        }
        public bool IsReady
        {
            get
            {
                return _isReady;
            }
        }

}

public class LocalizedText : MonoBehaviour
{
    [SerializeField] private string _key;
    private LocalizationManager _localizationManager;
    private TMP_Text _text;

    void Awake()
    {
        if (_localizationManager == null)
            _localizationManager =
GameObject.FindGameObjectWithTag("LocalizationManager").GetComponent<Locali
zationManager>();
        if (_text == null)
```

```
            _text = GetComponent<TMP_Text>();
        _localizationManager.OnLanguageChanged += UpdateText;
    }

    void Start()
    {
        UpdateText();
    }

    private void OnDestroy()
    {
        _localizationManager.OnLanguageChanged -= UpdateText;
    }

    virtual protected void UpdateText()
    {
        if (gameObject == null) return;

        if (_localizationManager == null)
            _localizationManager =
GameObject.FindGameObjectWithTag("LocalizationManager").GetComponent<Locali
zationManager>();
        if (_text == null)
            _text = GetComponent<TMP_Text>();
    _text.text = _localizationManager.GetLocalizedValue(_key);
    }
}
```

```
[System.Serializable]
public class LocalizationData
{
    public LocalizationItem[] Items;
}

[System.Serializable]
public class LocalizationItem
{
    public string Key;
    public string Value;
}

public class ButtonLanguage : MonoBehaviour
{
```

```
    [SerializeField] private
    LocalizationManager _localizationManager;
    [SerializeField] private MainMenu.ShopLogic.Shop _shop;
    public void OnButtonClick()
    {
        _localizationManager.CurrentLanguage = name;
        _localizationManager.LoadLocalizedText(name);
        _shop.ChangeLanguage();
    }
}


public class ShopPanel : MonoBehaviour
{

    [Space(5)]
    [Header("Ice")]
    [SerializeField] private GameObject _iceButtonContainer;
    [SerializeField] private Button _iceTowerButton;
    [SerializeField] private TMP_Text _iceTowerPriceText;

    [Space(5)]
    [Header("Cannon")]
    [SerializeField] private GameObject _cannonButtonContainer;
    [SerializeField] private Button _cannonTowerButton;
    [SerializeField] private TMP_Text _cannonTowerPriceText;

    [Space(5)]
    [Header("Crossbow")]
    [SerializeField] private GameObject _crossbowButtonContainer;
    [SerializeField] private Button _crossbowTowerButton;
    [SerializeField] private TMP_Text _crossbowTowerPriceText;

    [Space(5)]
   [Header("Fire")]
   [SerializeField] private GameObject _fireButtonContainer;
   [SerializeField] private Button _fireTowerButton;
   [SerializeField] private TMP_Text _fireTowerPriceText;

   [SerializeField] private GameObject _buyPanel;
   [SerializeField] private GameObject _updatePanel;
   [SerializeField] private Button _sellTowerButton;
   [SerializeField] private Button _updateButton;

   [Space(5)]
   [Header("Other")]
   [SerializeField] private GameObject _shopPanelContainer;
   [SerializeField] private WarningMessages _warnings;
   [SerializeField] private Coins _coinsContainer;
   [SerializeField] private TMP_Text _updateTowerText;
```

```
    [SerializeField] private Sprite _fireUpdateSprite;
    [SerializeField] private Sprite _iceUpdateSprite;
    [SerializeField] private EffectPooler.ObjectInfo.EffectObjectType
    _createTowerEffect;
    [SerializeField] private EffectPooler.ObjectInfo.EffectObjectType
    _sellTowerEffect;
    [SerializeField] private AudioUITypes _buySound;

     private int _fireTowerPrice, _iceTowerPrice, _cannonTowerPrice,
     _crossbowTowerPrice;


     private const string AUDIO_MANAGER_TAG_NAME = "AudioManager";
     private int _maxIceAmount;
     private int _maxCrossbowAmount;
     private Vector3 _distanceToCreateTower = new
     Vector3(0.3f, 1f, 0.3f);
     private Vector3 _offset = new Vector3(0f, -1.05f, 0f);
     private TowerPlace _towerPlace;
     private AudioManager _audioManager;
     private int _iceAmount = 0;
     private int _crossbowAmount = 0;
     private bool _mainMenuIsCreated;
     private bool _updateMenuIsCreated = false;
     private bool _finalMenuIsCreated = false;

     private void Awake()
     {
            _mainMenuIsCreated = true;
            GameObject[] audio =
GameObject.FindGameObjectsWithTag(AUDIO_MANAGER_TAG_NAME);
            _audioManager = audio[0].GetComponent<AudioManager>();

              Ground.OnGroundClick = HideShopPanel;
              #region SetTowerPrice
              _maxIceAmount =
PlayerPrefs.GetInt(PlayerPrefsTowerCharacterictics.IceAmount);
              _maxCrossbowAmount =
PlayerPrefs.GetInt(PlayerPrefsTowerCharacterictics.CrossbowAmount);
              _fireTowerPrice =
PlayerPrefs.GetInt(PlayerPrefsTowerCharacterictics.FirePrice);
              _iceTowerPrice =
PlayerPrefs.GetInt(PlayerPrefsTowerCharacterictics.IcePrice);
              _cannonTowerPrice =
PlayerPrefs.GetInt(PlayerPrefsTowerCharacterictics.CannonPrice);
              _crossbowTowerPrice =
PlayerPrefs.GetInt(PlayerPrefsTowerCharacterictics.CrossbowPrice);
              SetTowerPriceText();
              #endregion
         }
```

```
        private void Start()
        {
            #region ButtonAddListener
            _fireTowerButton.onClick.AddListener(CreateFireTower);
            _iceTowerButton.onClick.AddListener(CreateIceTower);
            _cannonTowerButton.onClick.AddListener(CreateCannonTower);
            _crossbowTowerButton.onClick.AddListener(CreateCrossbowTower);
            _sellTowerButton.onClick.AddListener(SellTower);
            _updateButton.onClick.AddListener(UpdateTower);
            #endregion
        }
        private void ChangeMenu()
        {
            if (_towerPlace != null)
            {
                if (_towerPlace.IsEmpty && !_mainMenuIsCreated &&
!_towerPlace.IsMaxLevel)
                {
                    SetVisibleUpdateButton();
                    ShowMainTowersMenu();
                }
                else if (!_towerPlace.IsEmpty && !_towerPlace.IsMaxLevel)
                {
                    if (!_updateMenuIsCreated)
                    {
                        if (_finalMenuIsCreated)
                        {
                            SetVisibleUpdateButton();
                            _finalMenuIsCreated = false;
                        }
                        if (_mainMenuIsCreated)
                            ShowUpdateTowersMenu();
                        else
                            _updateMenuIsCreated = true;
                    }
                }
                else if (_towerPlace.IsMaxLevel && !_finalMenuIsCreated)
                {
                    SetVisibleObjects();
                    _mainMenuIsCreated = false;
                    _updateMenuIsCreated = false;
                    _finalMenuIsCreated = true;

                }
                _updateTowerText.text =
                _towerPlace.TowerPrice.ToString();
            }
        }
```

```
    private void CreateUpdatedTower(TowerPooler.ObjectInfo.TowerObjectType
tower)
    {
        if (_coinsContainer.TakeCoins(_towerPlace.TowerPrice))
        {
            _towerPlace.InstantiateTower(tower, _offset,
            _createTowerEffect, _distanceToCreateTower);
            _audioManager.PlayUISound(_buySound);
        }
        else { _warnings.NotEnoughCoins(); }
    }


    private void ShowMainTowersMenu()
    {
        SetVisibleObjects();
        _mainMenuIsCreated = true;
        _updateMenuIsCreated = false;
        _finalMenuIsCreated = false;
    }

    private void ShowUpdateTowersMenu()
    {
        SetVisibleObjects();
        _mainMenuIsCreated = false;
        _finalMenuIsCreated = false;
       // ChangeButtonSprite();
    }

  private void SetVisibleUpdateButton()
  {
      _updateButton.gameObject.SetActive(true);
      _updateTowerText.gameObject.SetActive(true);
  }

  private void SetVisibleObjects()
  {
      if (_towerPlace.CurrentTower != null )
      {
          _buyPanel.SetActive(false);
          _updatePanel.SetActive(true);
          if (!_towerPlace.IsMaxLevel)
          {
              _updateButton.gameObject.SetActive(true);
              _updateTowerText.gameObject.SetActive(true);
          }
          else
          {
              _updateButton.gameObject.SetActive(false);
              _updateTowerText.gameObject.SetActive(false);
```

```
            }
        }
        else
        {
            _buyPanel.SetActive(true);
            _updatePanel.SetActive(false);
        }
    }

    private void SetTowerPriceText()
    {
        _fireTowerPriceText.text = _fireTowerPrice.ToString();
        _iceTowerPriceText.text = _iceTowerPrice.ToString();
        _cannonTowerPriceText.text = _cannonTowerPrice.ToString();
        _crossbowTowerPriceText.text = _crossbowTowerPrice.ToString();
    }

    private void HideShopPanel()
    {
        if (_towerPlace != null)
        {
            _towerPlace.TowerPlaceWasUnSelected();
            _towerPlace = null;
        }
        _shopPanelContainer.SetActive(false);
    }

private void CreateTower(TowerPooler.ObjectInfo.TowerObjectType tower)
{
    _towerPlace.InstantiateTower(tower, _offset, _
     createTowerEffect, _distanceToCreateTower);
}

private void CreateIceTower()
{
     if (_towerPlace.IsEmpty && _
     coinsContainer.TakeCoins(_iceTowerPrice))
     {
          _iceAmount++;
          CreateTower(TowerPooler.ObjectInfo.TowerObjectType.
          Frozen_Tower1);
          ShowUpdateTowersMenu();
          ChangeMenu();
          _audioManager.PlayUISound(_buySound);
          if (_iceAmount >= _maxIceAmount)
           _iceButtonContainer.SetActive(false);
    }
    else { _warnings.NotEnoughCoins(); }
}
```

```
private void CreateCrossbowTower()
{
    if (_towerPlace.IsEmpty &&
     _coinsContainer.TakeCoins(_crossbowTowerPrice))
    {
        _crossbowAmount++;
   CreateTower(TowerPooler.ObjectInfo.TowerObjectType.Crossbow_Tower1);
        ShowUpdateTowersMenu();
        ChangeMenu();
        _audioManager.PlayUISound(_buySound);
        if (_crossbowAmount >= _maxCrossbowAmount)
            _crossbowButtonContainer.SetActive(false);
    }
    else { _warnings.NotEnoughCoins(); }
}

private void CreateFireTower()
{
    if (_towerPlace.IsEmpty &&  _coinsContainer.TakeCoins(_fireTowerPrice))
    {
        CreateTower(TowerPooler.ObjectInfo.TowerObjectType.Fire_Tower1);
                ShowUpdateTowersMenu();
        ChangeMenu();
        _audioManager.PlayUISound(_buySound);
    }
    else { _warnings.NotEnoughCoins(); }
}

private void CreateCannonTower()
{
    if (_towerPlace.IsEmpty &&
   _coinsContainer.TakeCoins(_cannonTowerPrice))
    {

CreateTower(TowerPooler.ObjectInfo.TowerObjectType.Cannon_Tower1);
        ShowUpdateTowersMenu();
        ChangeMenu();
        _audioManager.PlayUISound(_buySound);
    }
    else { _warnings.NotEnoughCoins(); }
}

private void SellTower()
{
    if (_towerPlace.CurrentTower.GetComponent<IceTower>())
    {
        _iceAmount--;
        _coinsContainer.ReturnCoins(_towerPlace.DestroyTower
       (_sellTowerEffect, _distanceToCreateTower));
        ChangeMenu();
```

```
            _iceButtonContainer.SetActive(true);
            return;
        }
        else if (_towerPlace.CurrentTower.GetComponent<CrossbowTower>())
        {
            _crossbowAmount--;
            _coinsContainer.ReturnCoins(_towerPlace.DestroyTower
           (_sellTowerEffect, _distanceToCreateTower));
            ChangeMenu();
            _crossbowButtonContainer.SetActive(true);
            return;
       }
      _coinsContainer.PutCoins(_towerPlace.DestroyTower
      (_sellTowerEffect, _distanceToCreateTower));
       ChangeMenu();
      }

    private void UpdateTower()
    {
        if (_towerPlace.TowerIsCreated)
        {
        switch(_towerPlace.TowerType)
        {
            case TowerPooler.ObjectInfo.TowerObjectType.Fire_Tower1:
            CreateUpdatedTower(TowerPooler.ObjectInfo.TowerObjectType.
            Fire_Tower2);
             break;

            case TowerPooler.ObjectInfo.TowerObjectType.Fire_Tower2:
            CreateUpdatedTower(TowerPooler.ObjectInfo.TowerObjectType.
            Fire_Tower3);
            break;

            case TowerPooler.ObjectInfo.TowerObjectType.Frozen_Tower1:
            CreateUpdatedTower(TowerPooler.ObjectInfo.TowerObjectType.
            Frozen_Tower2);
            break;

            case TowerPooler.ObjectInfo.TowerObjectType.Frozen_Tower2:
            CreateUpdatedTower(TowerPooler.ObjectInfo.TowerObjectType.
            Frozen_Tower3);
            break;

            case TowerPooler.ObjectInfo.TowerObjectType.Crossbow_Tower1:
            CreateUpdatedTower(TowerPooler.ObjectInfo.TowerObjectType.
            Crossbow_Tower2);
            break;
```

```
            case TowerPooler.ObjectInfo.TowerObjectType.Crossbow_Tower2:
            CreateUpdatedTower(TowerPooler.ObjectInfo.TowerObjectType.
            Crossbow_Tower3);
            break;

            case TowerPooler.ObjectInfo.TowerObjectType.Cannon_Tower1:
            CreateUpdatedTower(TowerPooler.ObjectInfo.TowerObjectType.
            Cannon_Tower2);
            break;

            case TowerPooler.ObjectInfo.TowerObjectType.Cannon_Tower2:
            CreateUpdatedTower(TowerPooler.ObjectInfo.TowerObjectType.
            Cannon_Tower3);
            break;

            default:
            Debug.LogError("ShopPanel strangeTowerType");
            break;
        }
    }
    else { _warnings.TowerIsBuilding(); }
    ChangeMenu();
    }

    public void SetTowerPlace(TowerPlace towerPlace)
    {
        if (_towerPlace != null)
            _towerPlace.TowerPlaceWasUnSelected();
        else
        {
            _shopPanelContainer.SetActive(true);
            _warnings.HideStartWarning();
        }
        _towerPlace = towerPlace;
        _towerPlace.TowerPlaceWasSelected();
        _audioManager.OnBackButtonClick();
        ChangeMenu();
    }

    public void CheckTowerPlaceTypeForSale(TowerPlace towerPlace)
    {
        if (towerPlace.CurrentTower.GetComponent<IceTower>())
        {
            _iceAmount--;
             _coinsContainer.ReturnCoins(towerPlace.DestroyTower
           (_sellTowerEffect, _distanceToCreateTower));
           _iceButtonContainer.SetActive(true);
        }
        else if(towerPlace.CurrentTower.
          GetComponent<CrossbowTower>())
```

```
    {
         _crossbowAmount--;
           _coinsContainer.ReturnCoins(towerPlace.DestroyTower
           (_sellTowerEffect, _distanceToCreateTower));
         _crossbowButtonContainer.SetActive(true);
     }

     towerPlace.IsEmpty = true;
     towerPlace.CurrentTower = null;
     towerPlace.IsMaxLevel = false;
     if (_towerPlace == towerPlace)
     {
         ChangeMenu();
     }
 }

public void HideButtonsForStoryMode(int level)
{
    switch(level)
    {
        case 0:
                _crossbowButtonContainer.SetActive(false);
                _iceButtonContainer.SetActive(false);
                _cannonButtonContainer.SetActive(false);
                break;
            case 1:
                _crossbowButtonContainer.SetActive(false);
                _iceButtonContainer.SetActive(false);
                break;
            case 2:
                _iceButtonContainer.SetActive(false);
                break;
            case 3:
                _iceButtonContainer.SetActive(false);
                break;
            default:
                break;
        }
    }
}


public class CastlePanel : MonoBehaviour
{
    [Header("LogWithSpikes")]
    [SerializeField] private Button _logButton;
    [SerializeField] private TMP_Text _logPriceText;

    [Header("Castle")]
    [SerializeField] private Button _castleUpdateButton;
```

```
    [SerializeField] private TMP_Text _castleUpdatePriceText;

    [Header("Other")]
    [SerializeField] private Audio.AudioUITypes _buySound;
    [SerializeField] private WarningMessages _warnings;
    [SerializeField] private Coins _coinsContainer;

    [Header("Settable Objects in Scene")]
    [SerializeField] private Castle _castle;
    private Audio.AudioManager _audioManager;
    private const string AUDIO_MANAGER_TAG_NAME = "AudioManager";

    private void Awake()
    {
        GameObject[] audio = GameObject.FindGameObjectsWithTag
        (AUDIO_MANAGER_TAG_NAME);
        _audioManager = audio[0].GetComponent<Audio.AudioManager>();
    }


private void Start()
   {
       _castleUpdatePriceText.text = _castle.CatlePrice.ToString();
       _logPriceText.text = _castle.LogPrice.ToString();
       _logButton.onClick.AddListener(OnLogButtonClick);
       _castleUpdateButton.onClick.AddListener(UpdateCastle);
   }

private void OnLogButtonClick()
{
    if (_coinsContainer.TakeCoins(_castle.LogPrice))
    {
        _castle.UseTrap();
        _logButton.interactable = false;
        _logPriceText.text = "";
        _audioManager.PlayUISound(_buySound);
    }
    else _warnings.NotEnoughCoins();
}

private void UpdateCastle()
{
    if (_coinsContainer.TakeCoins(_castle.CatlePrice))
    {
     _castle.ChangeMaxHitPoints();
     _castleUpdatePriceText.text = _castle.CatlePrice.ToString();
     _audioManager.PlayUISound(_buySound);
    }
    Else _warnings.NotEnoughCoins();
}
```

```
public void SetLogActive()
{
    _logButton.interactable = true;
    _logPriceText.text = _castle.LogPrice.ToString();
}
}
```

```
public class Coins : MonoBehaviour
{
    [SerializeField] private TMP_Text _coinsAmountText;
    [SerializeField] private Score _score;
    [SerializeField] private int _startCoins;

    private const int EARNED_DURIN_LEVEL_DIVISION = 10;
    private int _coinsContainer;
    private int _coinsMultiply = 1;

    public static int EarnedDuringLevelCoins;

    private void Awake()
    {
        EarnedDuringLevelCoins = 0;
        _coinsContainer = _startCoins;
        _coinsAmountText.text = _startCoins.ToString();
    }

    public void PutCoins(int coins)
    {
        _coinsContainer += coins;
        _coinsAmountText.text = _coinsContainer.ToString();
        EarnedDuringLevelCoins += coins /EARNED_DURIN_LEVEL_DIVISION
       * EnemyCounter.CurrentWaveLevel * _coinsMultiply;
        _score.AddScores(coins * EnemyCounter.CurrentWaveLevel);
    }

    public void ReturnCoins(int coins)
    {
        _coinsContainer += coins;
        _coinsAmountText.text = _coinsContainer.ToString();
```

```
    }
    public bool TakeCoins(int price)
    {
        if (price <= _coinsContainer)
        {
            _coinsContainer -= price;
            _coinsAmountText.text = _coinsContainer.ToString();
            return true;
        }
        else { return false; }
    }
    public void IncreaseGold()
    {
        _coinsMultiply += 1;
    }
}

public class Score : MonoBehaviour
{
    [SerializeField] private TMP_Text _scoreAmountText;
    private static int _scoreAmount = 0;

    private void Awake()
    {
        _scoreAmount = 0;
    }
    public void AddScores(int scores)
    {
        _scoreAmount += scores;

        _scoreAmountText.text = _scoreAmount.ToString();
    }

    public static int GetScore()
    {
        return _scoreAmount;
    }
}


public class DailyRewards : MonoBehaviour
{
   [SerializeField] private GameObject _background;
   [SerializeField] private TMP_Text _status;
   [SerializeField] private Button _claimButton;
   [SerializeField] private Button _closeButton;
   [SerializeField] private ButtonDailyReward _dailyRewardButton;

   [SerializeField] private RewardPref _rewardPrefab;
   [SerializeField] private Transform _rewardsGrid;
```

```
[SerializeField] private ClaimRewardPanel _claimRewardPanel;
[SerializeField] private List<Reward> _reward;

private List<RewardPref> _rewardPrefabs;
private string _claimText;
private string _comeBackText;
private string _forNextRewardText;
private float _claimCoolDown = 24f; /// 24 / 60 / 6 / 2;
private float _claimDeadLine = 48f; /// 24 / 60 / 6 / 2;
private int _maxVisibleStreakCount = 12;
private int _minStreak = 0;
private bool _canClaimReward;


private int _currentStreak
 {
     get => PlayerPrefs.GetInt(PlayerPrefsKeyContainer.
          CurrentDayStreak, 0);
     set => PlayerPrefs.SetInt(PlayerPrefsKeyContainer.
         CurrentDayStreak, value);
 }


 private DateTime? _lastClaimTime
 {
     get
     {
         string data = PlayerPrefs.GetString(
        PlayerPrefsKeyContainer. LastDayStreakClaimedTime, null);
         if (!string.IsNullOrEmpty(data))
         return DateTime.Parse(data);
         return null;
     }
     set
     {
         if (value != null)
         PlayerPrefs.SetString(PlayerPrefsKeyContainer.
         LastDayStreakClaimedTime, value.ToString());
         else PlayerPrefs.DeleteKey(PlayerPrefsKeyContainer.
          LastDayStreakClaimedTime);
     }
 }


 private void Awake()
 {
     _claimButton.onClick.AddListener(ClaimReward);
     _closeButton.onClick.AddListener(ClosePanel);
     _dailyRewardButton.SetButtonAction(ShowPanel);
```

```
        _background.LeanScale(Vector2.zero, 0.5f).setEaseInBack();
            SetLocalization();
    }

    private void Start()
    {
        InitPrefabs();
        StartCoroutine(RewardStateUpdater());
    }

    private void InitPrefabs()
    {
        _rewardPrefabs = new List<RewardPref>();


     for (int i = 0; i < _maxVisibleStreakCount; i++)
     {
          _rewardPrefabs.Add(Instantiate(_rewardPrefab,
          _rewardsGrid, false));
     }
}

private IEnumerator RewardStateUpdater()
{
    while (true)
    {
        UpdateRewardsState();
        yield return new WaitForSeconds(1);
    }
}

private void UpdateRewardsState()
{
    _canClaimReward = true;
    if (_lastClaimTime.HasValue)
    {
      var timeSpan = DateTime.UtcNow - _lastClaimTime.Value;

            if (timeSpan.TotalHours > _claimDeadLine)
            {
                _lastClaimTime = null;
                _currentStreak = 0;
                _minStreak = 0;
            }
            else if (timeSpan.TotalHours < _claimCoolDown)
            {
                _canClaimReward = false;
            }
        }
        UpdateRewardsUI();
```

```
    }

 private void UpdateRewardsUI()
 {
     _claimButton.interactable = _canClaimReward;

     if (_canClaimReward)
     {
         if(!_dailyRewardButton.MarkShowed
         && _dailyRewardButton.gameObject.active)
        _dailyRewardButton.StartCoroutine(
        _dailyRewardButton.ShowRewardMark());
        _status.text = _claimText;
      }
    else
    {
       var nextClaimTime = _lastClaimTime.Value.AddHours
       (_claimCoolDown);
       var currentClaimCooldown = nextClaimTime - DateTime.UtcNow;
       string cooldown = $"{currentClaimCooldown.Hours:D2}:
       {currentClaimCooldown.Minutes:D2}:
       {currentClaimCooldown.Seconds:D2}";
       _status.text = $"{_comeBackText} {cooldown}
       {_forNextRewardText}";
    }

    for (int i = 0, streak = _minStreak;
      i < _rewardPrefabs.Count; i++, streak++)
    {
        _rewardPrefabs[i].SetRewardData(streak,
        _currentStreak + _minStreak, _reward[i]);
    }
}

private void ClaimReward()
{
    if (!_canClaimReward) return;

    var reward = _reward[_currentStreak];

    switch (reward.Type)
    {
        case Reward.RewardType.Gold:
            break;
        case Reward.RewardType.Random:
            break;
        case Reward.RewardType.Chest:
            break;
        default:
             throw new Exception("DailyRewars, ClaimReward error");
```

```
        }
        _lastClaimTime = DateTime.UtcNow;
        _currentStreak++;
        if(_currentStreak == _maxVisibleStreakCount)
        {
            _minStreak += _currentStreak;
            _currentStreak = 0;
        }
        UpdateRewardsState();
        _dailyRewardButton.RewardWasTaked();
        _claimRewardPanel.Show(reward.Type, ClosePanel,
        _currentStreak + _minStreak);
    }

     private void ClosePanel()
     {
         _background.LeanScale(Vector2.zero, 0.5f).setEaseInBack();
    }
    private void ShowPanel()
    {
        _background.SetActive(true);
        _background.LeanScale(Vector2.one, 0.5f).setEaseOutBack();
    }
    public void SetLocalization()
    {
        if (PlayerPrefs.GetString("Language") == "ru_RU")
        {
            _claimText = "Заберите награду!";
            _comeBackText = "Вернитесь через";
            _forNextRewardText = "для вашей следующей награды";
        }
        else if (PlayerPrefs.GetString("Language") == "uk_UK")
        {
            _claimText = "Заберіть нагороду!";
            _comeBackText = "Поверніться через";
            _forNextRewardText = "для вашої наступної винагороди";
        }
        else // (PlayerPrefs.GetString("Language") == "en_US")
        {
            _claimText = "Claim Your reward!";
            _comeBackText = "Come back in";
            _forNextRewardText = "for your next reward";
        }
        _claimRewardPanel.SetLocalization();
    }
}

[Serializable]
public class Reward
{
```

```
    public enum RewardType
    {
        Gold,
        Chest,
        Random
    }

    public RewardType Type;
}



public class RewardPref : MonoBehaviour
{
    [SerializeField] private Image _background;
    [SerializeField] private Color _defaultColor;
    [SerializeField] private Color _currentColor;
    [SerializeField] private Color _selectedColor;

    [SerializeField] private TMP_Text _dayText;

    [SerializeField] private Image _rewardIcon;
    [SerializeField] private Sprite _rewardGold;
    [SerializeField] private Sprite _rewardRandom;
    [SerializeField] private Sprite _rewardChest;


    private string _startText;

    private void Awake()
    {
        SetLocalization();
    }


    private void SetLocalization()
    {
        if (PlayerPrefs.GetString("Language") == "ru_RU")
       _   startText = "День";
        else if (PlayerPrefs.GetString("Language") == "uk_UK")
       _   startText = "День";
        else if (PlayerPrefs.GetString("Language") == "en_US")
           _startText = "Day";
            else _startText = "Day";
     }

     private Sprite CheckRewardType(Reward.RewardType reward)
     {
         switch (reward)
         {
```

```
            case Reward.RewardType.Gold:
                return _rewardGold;
            case Reward.RewardType.Random:
                return _rewardRandom;
            case Reward.RewardType.Chest:
                return _rewardChest;
            default:
                throw new Exception("RewardPref incorrectType");
        }
    }

public void SetRewardData(int day, int currentStreak, Reward reward)
{
    SetLocalization();
    _dayText.text = $"{_startText} {day + 1}";
    _rewardIcon.sprite = CheckRewardType(reward.Type);

    if (day > currentStreak)
        _background.color = _defaultColor;
    else if (day < currentStreak)
        _background.color = _selectedColor;
    else
        _background.color = _currentColor;
    }
}

public class ClaimRewardPanel : MonoBehaviour
{
    [SerializeField] private GameObject _claimRewardContainer;
    [SerializeField] private Image _rewardIcon;
    [SerializeField] private TMP_Text _rewardValue;
    [SerializeField] private Button _claimButton;
    [SerializeField] private CoinsPanel _coins;
    [SerializeField] private Audio.AudioManager _audioManager;

    [Space(5)]
    [SerializeField] private Sprite _rewardGold;
    [SerializeField] private Sprite _rewardRandom;
    [SerializeField] private Sprite _rewardChest;

    private Action OnRewardClaimed;
    private string _goldText;
    private const int GOLD_PRIZE = 100;
    private int _tempGoldContainer = 0;

    private void Start()
    {
        _claimButton.onClick.AddListener(Hide);
        Hide();
    }
```

```
    private Sprite CheckRewardType(Reward.RewardType reward)
    {
        switch (reward)
        {
            case Reward.RewardType.Gold:
                return _rewardGold;
            case Reward.RewardType.Random:
                return _rewardRandom;
            case Reward.RewardType.Chest:
            return _rewardChest;
        default:
            throw new Exception("RewardPref incorrectType");
    }
}

private void Hide()
{
_claimRewardContainer.LeanScale(Vector2.zero, 0.5f).setEaseInBack();
     _claimRewardContainer.SetActive(false);
     OnRewardClaimed?.Invoke();
}
private IEnumerator EnableMoneyAnimation(int gold)
{
    audioManager.PlayUISound(Audio.AudioUITypes.TickSound);
    _tempGoldContainer = 0;
    int _oneTenthEarnedGold = gold / 60;
    while (gold > 0)
    {
            if (gold - _oneTenthEarnedGold >= _oneTenthEarnedGold)
            {
                _coins.AddDailyCoins(_oneTenthEarnedGold);
                gold -= _oneTenthEarnedGold;
                _tempGoldContainer += _oneTenthEarnedGold;
            }
            else
            {
                _coins.AddDailyCoins(gold, true);
                gold -= gold;
                _tempGoldContainer += _oneTenthEarnedGold;
            }
            _rewardValue.text = $"{_tempGoldContainer}";
            yield return new WaitForSeconds(0.015f);
     }
     _audioManager.PlayUISound(Audio.AudioUITypes.TakeDaily);
     _rewardValue.text = $"{_tempGoldContainer} {_goldText}";
 }

 private IEnumerator EnableRandomAnimation(int gold, int streak)
 {
```

```
        _audioManager.PlayUISound(Audio.AudioUITypes.TickSound);
        int rand = 0;
        for(int i = 0; i < 60; i++)
        {
            rand = UnityEngine.Random.Range(-95, 400);
            _rewardValue.text = $"{(GOLD_PRIZE + rand) * streak}";
                   yield return new WaitForSeconds(0.015f);
        }
      _rewardValue.text = $"{(GOLD_PRIZE + rand) * streak} {_goldText}";
        coins.AddDailyCoins((GOLD_PRIZE + rand)
        * streak, true);
         _audioManager.PlayUISound(Audio.AudioUITypes.TakeDaily);
}

public void Show(Reward.RewardType reward, Action rewardClaimed,
 int streak)
{
        _rewardIcon.sprite = CheckRewardType(reward);

        switch(reward)
        {
              case Reward.RewardType.Gold:
              int additionalGold = UnityEngine.Random.Range(25, 50);
              StartCoroutine(EnableMoneyAnimation((GOLD_PRIZE
             + additionalGold) * streak));
              break;

              case Reward.RewardType.Random:
              StartCoroutine(EnableRandomAnimation(GOLD_PRIZE,streak));
              break;

              case Reward.RewardType.Chest:
              _rewardValue.text = $" 1 {reward}";
              break;
         }

 _claimRewardContainer.SetActive(true);
 _claimRewardContainer.LeanScale(Vector2.one,0.5f).setEaseOutBack();
  OnRewardClaimed = rewardClaimed;
}

 public void SetLocalization()
 {
        if (PlayerPrefs.GetString("Language") == "ru_RU")
        {
            _goldText = "золота";
        }
        else if (PlayerPrefs.GetString("Language") == "uk_UK")
        {
            _goldText = "золота";
```

```
        }
        else // (PlayerPrefs.GetString("Language") == "en_US")
        {
            _goldText = "gold";
        }
    }
}

public class ButtonDailyReward : MonoBehaviour
{
    [SerializeField] private Image _mark;
    [SerializeField] private Button _button;
    [SerializeField] private GameObject _buttonContainer;
    [SerializeField] private float _markDelay = 0.1f;
    [SerializeField] private Audio.AudioManager _audioManager;

    private Color _color;
    private Action OnButtonClick;
    private bool _markShowed;
    public bool MarkShowed => _markShowed;

    private void Awake()
    {
    _button.onClick.AddListener(() => OnButtonClick?.Invoke());
    }

    public void SetButtonAction(Action action)
    {
        OnButtonClick = action;OnButtonClick += () =>
       _audioManager.PlayUISound(Audio.AudioUITypes.ShowDaily);
    }

    public IEnumerator ShowRewardMark()
    {
        _color = _mark.color;
        _color.a = 0f;
        _mark.color = _color;
        _markShowed = true;
        _mark.gameObject.SetActive(true);
        while(true)
        {
            while(_color.a > 0.1f)
            {
                _color.a -= 0.05f;
                _mark.color = _color;
                yield return new WaitForSeconds(_markDelay);
            }
            while (_color.a < 1f)
            {
                _color.a += 0.05f;
```

```
                _mark.color = _color;
                yield return new WaitForSeconds(_markDelay);
            }
        }
    }

    public void RewardWasTaked()
    {
        StopAllCoroutines();
        _mark.gameObject.SetActive(false);
        _markShowed = false;
    }

    public void HideAndShow()
    {
        if (_buttonContainer.active)
        {
            _buttonContainer.LeanScale(Vector2.zero,
           0.5f).setEaseInBack();
            _buttonContainer.SetActive(false);
        }
        else
        {
            _color = _mark.color;
            _color.a = 0f;
            _mark.color = _color;
            _markShowed = false;
            _buttonContainer.SetActive(true);
            _buttonContainer.LeanScale(Vector2.one,
              0.5f).setEaseOutBack();
        }
    }
}


public class DialogueController : MonoBehaviour
{
    [SerializeField] protected List<Dialogue> _startDialogues;
    [SerializeField] protected List<Dialogue>
    _startAlternativeDialogues;
    [SerializeField] protected List<Dialogue> _choiceDialogues; // n
    [SerializeField] protected List<Dialogue> _endDialogue;
    [SerializeField] private List<Sprite> _characterSprites;
    [SerializeField] private DialoguePressAnyKey _pressAnyKeyText;
    [SerializeField] private GameObject _dialogueBorder;
    [SerializeField] private GameObject _pausePanel;
    [SerializeField] private Image _characterImage;
    protected Action _onDialogueEnded;
```

```
  private IEnumerator ChangeLanguageDialogue
  (List<Dialogue> dialogues)
  {
     foreach (var dialogue in dialogues)
     {
         dialogue.ActivateText(ChangeSpriteOnImage,
        _pausePanel, _pressAnyKeyText);
    yield return new WaitForSeconds(0.01f);
    dialogue.GetComponent<LocalizedText>().enabled = false;
    dialogue.gameObject.SetActive(false);
   }
}
 private void ChangeSpriteOnImage(DialogueSprites character)
 {
     switch (character)
     {
         case DialogueSprites.MainCharacter:
             _characterImage.sprite = _characterSprites[0];
             break;
         case DialogueSprites.Mage:
             _characterImage.sprite = _characterSprites[1];
             break;
         case DialogueSprites.Yan:
             _characterImage.sprite = _characterSprites[2];
             break;
         case DialogueSprites.Warrior:
             _characterImage.sprite = _characterSprites[3];
             break;
         case DialogueSprites.Engineer:
             _characterImage.sprite = _characterSprites[4];
             break;
         default:
             break;
     }
 }

public virtual void StartDialogue(bool isEnded, int scenario = 1)
 {
     _dialogueBorder.SetActive(true);
     if (!isEnded)
     {
         _characterImage.gameObject.SetActive(true);
         switch (scenario)
         {
             case 1: _startDialogues[0].
              CreateText(_onDialogueEnded);
              break;
```

```
                case 2: _startAlternativeDialogues[0].
                 CreateText(_onDialogueEnded);
                 break;

                case 3: _startAlternativeDialogues[1].
                 CreateText(_onDialogueEnded);
                 break;

      case 4: _startAlternativeDialogues[2].
      CreateText(_onDialogueEnded);
      break;
    }
}
    else
    {
        _characterImage.gameObject.SetActive(true);
        _endDialogue[0].gameObject.SetActive(true);
        _endDialogue[0].CreateText(_onDialogueEnded);
    }
}
public void SetDialogueLanguage()
{
    gameObject.SetActive(true);
    StartCoroutine(ChangeLanguageDialogue(_startDialogues));
    StartCoroutine(ChangeLanguageDialogue(_endDialogue));

    if(_startAlternativeDialogues != null)
 StartCoroutine(ChangeLanguageDialogue(_startAlternativeDialogues));

    if(_choiceDialogues != null)
      StartCoroutine(ChangeLanguageDialogue(_choiceDialogues));
}

public void MakeChoice(int choice)
{
    switch(choice)
    {
        case 1: _choiceDialogues[0].CreateText(_onDialogueEnded);
break;

        case 2: _choiceDialogues[1].CreateText(_onDialogueEnded);
break;

        case 3: _choiceDialogues[2].CreateText(_onDialogueEnded);
break;
        case 4: _choiceDialogues[3].CreateText(_onDialogueEnded);
break;
    }
}
```

```
public virtual void SetEventOnDialogueEnd(Action onDialogueEnd)
{
      _onDialogueEnded = onDialogueEnd;
      _onDialogueEnded +=
      () => { _characterImage.gameObject.SetActive(false); };
}}



public class DialogueController : MonoBehaviour
{
    [SerializeField] protected List<Dialogue> _startDialogues;
    [SerializeField] protected List<Dialogue>
    _startAlternativeDialogues;
    [SerializeField] protected List<Dialogue> _choiceDialogues;
    [SerializeField] protected List<Dialogue> _endDialogue;
    [SerializeField] private List<Sprite> _characterSprites;
    [SerializeField] private DialoguePressAnyKey _pressAnyKeyText;
    [SerializeField] private GameObject _dialogueBorder;
    [SerializeField] private GameObject _pausePanel;
    [SerializeField] private Image _characterImage;
    protected Action _onDialogueEnded;

    private IEnumerator ChangeLanguageDialogue
     (List<Dialogue> dialogues)
    {
        foreach (var dialogue in dialogues)
        {
            dialogue.ActivateText(ChangeSpriteOnImage,
            _pausePanel, _pressAnyKeyText);
            yield return new WaitForSeconds(0.01f);
            dialogue.GetComponent<LocalizedText>().enabled = false;
            dialogue.gameObject.SetActive(false);
        }
    }
    private void ChangeSpriteOnImage(DialogueSprites character)
    {
        switch (character)
        {
            case DialogueSprites.MainCharacter:
                _characterImage.sprite = _characterSprites[0];
                break;
            case DialogueSprites.Mage:
                _characterImage.sprite = _characterSprites[1];
                break;
            case DialogueSprites.Yan:
                _characterImage.sprite = _characterSprites[2];
                break;
            case DialogueSprites.Warrior:
                _characterImage.sprite = _characterSprites[3];
```

```
                break;
            case DialogueSprites.Engineer:
                _characterImage.sprite = _characterSprites[4];
                break;
            default:
                break;
    }
}

 public virtual void StartDialogue(bool isEnded, int scenario = 1)

    _dialogueBorder.SetActive(true);
    if (!isEnded)
    {
        _characterImage.gameObject.SetActive(true);
        switch (scenario)
        {
            case 1: _startDialogues[0].
             CreateText(_onDialogueEnded); break;
            case 2: _startAlternativeDialogues[0].
             CreateText(_onDialogueEnded); break;
            case 3: _startAlternativeDialogues[1].
             CreateText(_onDialogueEnded); break;
            case 4: _startAlternativeDialogues[2].
             CreateText(_onDialogueEnded); break;
        }
    }
    else
    {
        _characterImage.gameObject.SetActive(true);
        _endDialogue[0].gameObject.SetActive(true);
        _endDialogue[0].CreateText(_onDialogueEnded);
    }
}

public void SetDialogueLanguage()
{
    gameObject.SetActive(true);
    StartCoroutine(ChangeLanguageDialogue(_startDialogues));
    StartCoroutine(ChangeLanguageDialogue(_endDialogue));

    if(_startAlternativeDialogues != null)
StartCoroutine(ChangeLanguageDialogue(_startAlternativeDialogues));

    if(_choiceDialogues != null)
     StartCoroutine(ChangeLanguageDialogue(_choiceDialogues));
}
```

```
public void MakeChoice(int choice)
{
    switch(choice)
    {
        case 1: _choiceDialogues[0].CreateText(_onDialogueEnded);
     break;
        case 2: _choiceDialogues[1].CreateText(_onDialogueEnded);
     break;
        case 3: _choiceDialogues[2].CreateText(_onDialogueEnded);
     break;
         case 4: _choiceDialogues[3].CreateText(_onDialogueEnded);
     break;
    }
}

public virtual void SetEventOnDialogueEnd(Action onDialogueEnd)
{
     _onDialogueEnded = onDialogueEnd;
     _onDialogueEnded += () => {
_characterImage.gameObject.SetActive(false); };
}}


public enum DialogueSprites
{
    MainCharacter,
    Mage,
    Yan,
    Warrior,
    Engineer
}


public class DialoguePressAnyKey : MonoBehaviour
{
    [SerializeField] private TMP_Text _text;
    private float _showSpeed = 0.075f;
    private float _showTime = 1f;
    private float _hideSpeed = 0.05f;
    private float _hideTime = 1f;
    private const float COLOR_VALUE = 0.05f;
    private Color _textColor;

    private IEnumerator ShowAndHideText()
    {
```

```
        _textColor = _text.color;



        while (_textColor.a < 1)
        {
            _textColor.a += COLOR_VALUE;
            yield return new WaitForSeconds(_showSpeed);
            _text.color = _textColor;
        }
        yield return new WaitForSeconds(_showTime);

        while (_textColor.a > 0)
        {
            _textColor.a -= COLOR_VALUE;
            yield return new WaitForSeconds(_hideSpeed);
            _text.color = _textColor;
        }
        yield return new WaitForSeconds(_hideTime);
        StartCoroutine(ShowAndHideText());
    }

    public void ActivateText()
    {
        _text.gameObject.SetActive(true);
        StartCoroutine(ShowAndHideText());
    }
    public void DisableText()
    {
        StopAllCoroutines();
        _text.gameObject.SetActive(false);
        _textColor = _text.color;
        _textColor.a = 0;
        _text.color = _textColor;
    }
}


public static class ChoiceStatus
{
    public static string MainCharacter = "MainCharacter"; yes/no
    public static string Mage = "Mage"; // yes/no
    public static string Yan  = "Yan"; // yes/no
    public static string Engineer = "Engineer"; // yes/no
    public static string Warrior  = "Warrior"; // yes/no

    // End Level 1
    public static string StayLevel1 = "StayLevel1"; // stay or sneak

    // Start Level2
```

```
    // Звинуватити мага в тому, що він не туди телепортував нас
    public static string ConflictWithMageLevel2 =
      "ConflictWithMageLevel2"; // yes/no Mage Sad

    // End Level2
    // Погодитись з перемир'ям між орками
    public static string NegotiateOrcsLevel2 =  "NegotiateOrcsLevel2"; //
yes/no Warrior sad

    // Start Level3
    // Звинуватити мага в тому, що через його телепорт ми не встигли
допомогти людям.
    public static string ConflictWithMageLevel3 =
      "ConflictWithMageLevel3"; // yes/no Warrior sad

    // End Level3
    // Вирішувати телепортуватися до міста чи до місця сили
    public static string ChoiceToTeleportLevel3 =
      "ChoiceToTeleportLevel3"; // power / Town Mage Sad

    // Start Level4
    // Звинуватити мага в тому, що він навмисно телепортував нас до
місця сили
    public static string ConflictWithMageLevel4 =
      "ConflictWithMageLevel4"; // Mage Sad
}

public class ChoiceController : MonoBehaviour
{
    [SerializeField] private List<GameObject> _choicePanels;
    [Space(10)]
    [SerializeField] private Button _choiceYes1;
    [SerializeField] private Button _choiceYes2;
    [Space(5)]
    [SerializeField] private Button _choiceNo1;
    [SerializeField] private Button _choiceNo2;
    [Space(5)]
    [SerializeField] private DialogueController _dialogueController;
    [SerializeField] private Coins _coins;

    private int _levelNumber;
    private int _choiceNumber = 1;
    private GameObject _dialogue;

    private void Awake()
    {
        _choiceYes1.onClick.AddListener(OnChoiceYes1);
        _choiceYes2.onClick.AddListener(OnChoiceYes2);
        _choiceNo1.onClick.AddListener(OnChoiceNo1);
        _choiceNo2.onClick.AddListener(OnChoiceNo2);
```

```
}

private void OnChoiceYes1()
{
    _choiceYes1.interactable = false;
    ChoiceYes(_choiceNumber);
    _choiceNumber++;
    _choiceYes1.interactable = true;
}
private void OnChoiceYes2()
{
    _choiceYes2.interactable = false;
    ChoiceYes(_choiceNumber);
    _choiceNumber++;
    _choiceYes2.interactable = true;
}

private void OnChoiceNo1()
{
    _choiceNo1.interactable = false;
     ChoiceNo(_choiceNumber);
    _choiceNo1.interactable = true;
    _choiceNumber++;
}

private void OnChoiceNo2()
{
    _choiceNo2.interactable = false;
    ChoiceNo(_choiceNumber);
    _choiceNo2.interactable = true;
    _choiceNumber++;
}

private void ChoiceYes(int numb)
{
    if(numb == 1)
    {
        switch(_levelNumber)
        {
            case 1: PlayerPrefs.SetString(
            ChoiceStatus.StayLevel1, "stay");
                  _dialogueController.MakeChoice(1);
                   break;
            case 2: PlayerPrefs.SetString(
            ChoiceStatus.ConflictWithMageLevel2, "yes");
      _dialogueController.MakeChoice(1);
                  break;
            case 3: PlayerPrefs.SetString(
            ChoiceStatus.ConflictWithMageLevel3, "yes");
      _dialogueController.MakeChoice(1);
```

```
                    break;
                case 4: PlayerPrefs.SetString(
                ChoiceStatus.ConflictWithMageLevel4, "yes");
           _dialogueController.MakeChoice(1);
                    break;
            }
        }
        else if ( numb == 2)
        {
            switch (_levelNumber)
            {
                case 2: PlayerPrefs.SetString(
                ChoiceStatus.NegotiateOrcsLevel2, "yes"); _
           dialogueController.MakeChoice(3);
                    break;
                case 3: PlayerPrefs.SetString(
                ChoiceStatus.ChoiceToTeleportLevel3, "power");
           _dialogueController.MakeChoice(3);
                    break;
            }
        }
        _choicePanels[_choiceNumber-1].SetActive(false);
        _dialogue.SetActive(false);
    }
    private void ChoiceNo(int numb)
    {
        if (numb == 1)
        {
            switch (_levelNumber)
            {
                case 1: PlayerPrefs.SetString(
                ChoiceStatus.StayLevel1, "sneak");
           _dialogueController.MakeChoice(2);
                    break;
                case 2: PlayerPrefs.SetString(
                ChoiceStatus.ConflictWithMageLevel2, "no");
           _dialogueController.MakeChoice(2);
                    break;
                case 3: PlayerPrefs.SetString(
                ChoiceStatus.ConflictWithMageLevel3, "no");
           _dialogueController.MakeChoice(2);
                     break;
                case 4: PlayerPrefs.SetString(
                ChoiceStatus.ConflictWithMageLevel4, "no");
           _dialogueController.MakeChoice(2);
                    break;

            }
        }
```

```
    else if (numb == 2)
    {
        switch (_levelNumber)
        {
            case 2: PlayerPrefs.SetString(
            ChoiceStatus.NegotiateOrcsLevel2, "no");
       _dialogueController.MakeChoice(4);
                 break;
            case 3: PlayerPrefs.SetString(
            ChoiceStatus.ChoiceToTeleportLevel3, "town");
       _dialogueController.MakeChoice(4);
                 break;
        }
    }
    _choicePanels[_choiceNumber-1].SetActive(false);
    _dialogue.SetActive(false);
}
public int GetChoiceInfo(int level)
{
    _levelNumber = level - 1;
    switch (_levelNumber)
    {
        case 1: return 1;
        case 2:
            if (PlayerPrefs.GetString(
            ChoiceStatus.StayLevel1) == "yes")
            {
                _coins.TakeCoins(100);
                return 2;
            }
            else return 1;

        case 3:
            if (PlayerPrefs.GetString(ChoiceStatus.
            ConflictWithMageLevel2) == "yes"
            || PlayerPrefs.GetString(
            ChoiceStatus.NegotiateOrcsLevel2) == "yes")
                 return 2;
            else return 1;
        case 4:
            if (PlayerPrefs.GetString(
            ChoiceStatus.ChoiceToTeleportLevel3) == "town")
            {
                if (PlayerPrefs.GetString(
                ChoiceStatus.ConflictWithMageLevel2) == "yes"
                && PlayerPrefs.GetString(
                ChoiceStatus.ConflictWithMageLevel3) == "yes")
                {return 4; }
            }
```

```
            else if (PlayerPrefs.GetString(
            ChoiceStatus.ConflictWithMageLevel2) == "yes"
            ||PlayerPrefs.GetString(
            ChoiceStatus.ConflictWithMageLevel3) == "yes")
            {
                 return 3;
            }
            else return 2;
            }
            else return 1;

        default:
            return 1;
    }
}
public void ShowChoicePanel(GameObject dialogue)
{
    _dialogue = dialogue;
    _choicePanels[_choiceNumber-1].SetActive(true);
}
```