ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ
ТЕХНОЛОГІЙ НАВЧАЛЬНО-НАУКОВИЙ
ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка гри жанру Tower Defence з елементами Real Time Strategy мовою C#»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Карєн ЧЕРКАСОВ
(підпис)

Виконав: здобувач вищої освіти групи ПД-43

Карєн ЧЕРКАСОВ

Керівник: Микита ТРЕНЬОВ
асистент кафедри ІПЗ

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____Ірина ЗАМРІЙ
« _____» _____2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Черкасову Карєну Андрійовичу

1. Тема кваліфікаційної роботи: «Розробка гри жанру Tower Defence з елементами Real Time Strategy»
керівник кваліфікаційної роботи асистент кафедри Микита ТРЕНЬОВ,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56.

2. Строк подання кваліфікаційної роботи «02» червня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: Науково-технічна література на тему розробки ігор на Unity, алгоритми роботи основних елементів гри.

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

1. Огляд гри жанру Tower Defence з елементами Real Time Strategy та аналіз існуючих ігор жанру Tower Defence.
2. Проектування гри жанру Tower Defence з елементами Real Time Strategy.
3. Програмна реалізація та опис функціонування гри жанру Tower Defence з елементами Real Time Strategy.

4. Тестування гри.

5. Перелік графічного матеріалу: *презентація*
   1. Аналіз аналогів.
   2. Концепт гри.
   3. Вимоги до програмного забезпечення.
   4. Програмні засоби реалізації.
   5. Діаграма варіантів використання
   6. Діаграма класів
   7. Екранні форми.
   8. Демонстрація роботи застосунку
   9.Апробація результатів дослідження

6. Дата видачі завдання«25» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	25.02.-04.03.2025	
2	Аналіз та дослідження існуючих аналогів	05.03-13.03.2025	
3	Огляд та аналіз гри жанру Tower Defence з елементами Real Time Strategy	14.03-15.03.2025	
4	Огляд та аналіз засобів реалізації Tower Defence з елементами Real Time Strategy	16.03-21.03.2025	
5	Проектування архітектури гри жанру Tower Defence з елементами Real Time Strategy	22.03-03.04.2025	
6	Реалізація та тестування гри жанру Tower Defence з елементами Real Time Strategy	04.04-28.04.2025	
7	Оформлення роботи: вступ, висновки, реферат	29.04-11.05.2025	
8	Розробка демонстраційних матеріалів	12.05-18.05.2025	
9	Попередній захист роботи	19.05-30.05.2025	

Здобувач(ка) вищої освіти _____ Карєн ЧЕРКАСОВ
(підпис)

Керівник
кваліфікаційної роботи _____ Микита ТРЕНЬОВ
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 46 стор., 2 табл., 34 рис., 11 джерел.

Мета роботи – розширення геймплею жанру Tower Defence з елементами Real Time Strategy завдяки імплементації комбінації нововведень, що поглибшують геймплей, зберігаючи сильні сторони жанру Tower Defence.

Об'єкт дослідження – геймплей гри жанру Tower Defence з елементами Real Time Strategy.

Предмет дослідження – засоби реалізації гри у жанрі Tower Defence з елементами Real Time Strategy.

Короткий зміст роботи: В роботі проаналізовано предметну галузь та засоби релізації гри жанру Tower Defence з елементами Real Time Strategy. Проаналізовано вже випушені ігри жанру Tower Defense : Plants vs. Zombies, Defense Grid: The Awakening, Kingdom Rush. Розроблено гру та програмно реалізовані ключові функціональні можливості, зокрема: будівля економічних та захисних споруд, добування ресурсів за допомогою економічних споруд, ремонт існуючих споруд, демонтаж існуючих споруд, знищення ігрових ворогів за допомогою захисних споруд. Проведено тестування геймплею гри. В роботі використано мову програмування C# для програмування логіки гри, середу програмування Visual Studio у якості текстового редактору для написання скриптів гри, ігровий рушій Unity для компіляції коду, доступу до бібліотек Unity, створення ігрового інтерфейсу та дизайну рівнів.

Сферою використання гри є мультимедія та розваги, рекреаційні активність, відпочинок.

КЛЮЧОВІ СЛОВА: ГЕЙМПЛЕЙ, UI, GUI, PREFAB, C#, UNITY, ASSET STORE, ASSET

ЗМІСТ

ВСТУП

Актуальність: Відеоігри за останні десятиліття стали невід'ємною частиною людської культури, на даний момент в світі приблизно три з половиною мільярди гравців в відеоігри, що майже половина всього людського населення і ця цифра тільки росте [1]. Деяки ігри стали кіберспортивними дисциплінами(Starcraft, Counter Strike, Dota 2 тощо), soundtrack деяких ігор стає культовим, по мотивам ігор знімають фільми. Відеоігри закріпили себе як один з найпопулярніших видів рекреаційної активності на найближчі десятиліття.

Відеогра - це розважально-інтерактивне програмне забезпечення, створене для відтворення через екран звуків, зображень або анімації, які контролюються гравцем чи гравцями. Гравець взаємодіє з грою, керуючи персонажами чи об'єктами на екрані за допомогою спеціальних контролерів, клавіатури, миші або інших пристроїв введення. Відеогри можуть мати різні жанри, такі як шутер, стратегія, пригоди, головоломки тощо, і можуть бути розроблені для різних платформ.

Об’єкт дослідження – розширення геймплею жанру Tower Defence з елементами Real Time Strategy завдяки імплементації комбінації нововведень, що поглибшують геймплей, зберігаючи сильні сторони жанру Tower Defence.

Предмет дослідження – відеогра жанру Tower Defence з елементами Real Time Strategy.

Мета роботи – підвищення популярності жанру Tower Defence з елементами Real Time Strategy.

Методи дослідження: Спершу було проалізовано вже випущені аналогічні ігри жанру Tower Defence, їхні переваги і недоліки. На основі цього аналізу були сформовані функціональні та нефункціональні вимоги до гри. Після цього були підібрані технології та засоби реалізації, для реалізації гри були обрані мова програмування C# для програмування логіки гри, середа програмування Visual Studio у якості текстового редактору для написання скриптів гри, ігровий рушій

Unity для керуванням компіляцією коду, доступу до бібліотек Unity, створення ігрового інтерфейсу та дизайну рівнів.

Наукова новизна роботи визначається наступними функціональними

складовими: суміщення сильних компонентів жанрів Tower Defence та Real Time Strategy: вільне розміщення будівель на карті, добування ресурсів через економічні будівлі, розширена взаємодія між ігровими ворогами та будівлями гравця, простий але захопливий геймплей.

Практична значущість результатів полягая в можливості захопливо проводити рекреаційні активності у вільний час.

1 АНАЛІЗ ГРИ ЖАНРУ TOWER DEFENCE З ЕЛЕМЕНТАМИ REAL TIME STRATEGY

1.1 Гра жанру Tower Defence з елементами Real Time Strategy

Tower Defence – це піджанр стратегічних ігор, який став популярним завдяки простій, але захопливій механіці. Гравець має у своєму розпорядженні різні види оборонних споруд, які можуть бути вдосконалені та модифіковані для більш ефективного знищення ворогів. Вороги рухаються по заздалегідь визначених шляхах або мають свободу пересування залежно від правил гри. Основними характеристиками ігор цього жанру є система хвиль ворогів, можливість покращєння веж та управління ресурсами.

Головні переваги ігор жанру Tower Defence:

- Проста ігрова механіка забезпечує низкий вхідний поріг для нових гравців
- Відсутність потреби в швидкій реакції, підходить для гравців, які надають перевагу розміреному темпу.
- розвиток логічного та стратегічного мислення
- Поступовий ріст складності - гравець поступово навчається та стикається з дедалі серйознішими викликами.
- Захоплення для широкої аудиторії - Tower Defence підходять як для казуальних гравців, так і для хардкорних стратегів.

Але як і будь який ігровий жанр, Tower Defence не позбавлений недоліків:

- Монотонність геймплею - повторювані дії можуть швидко набридати.

Обмежена взаємодія між ворогами та вежами - іноді гравець просто спостерігає без активної участі.

- Обмеження креативності гравця— зазвичай немає свободи переміщення або взаємодії з ігровим світом поза встановленими межами.

Аби нівелювати недоліки жанру Tower Defence, в нагоді можуть стати деякі

елементи жанру Real Time Strategy:

- Можливість вільно розміщувати споруди на карті, що дозволить більший спектр можливих стратегій
- Розширена взаємодія між ігровими ворогами та спорудами дозволить урізноманітнити ігрові ситуації та
- Наявність декількох ігрових ресурсів, що дає можливість різними методами оптимізувати ігрову економіку

1.2 Цільова аудиторія

На перший погляд може здатися, що такий відносно нескладний жанр ігр як Tower Defence розрахований на дітей, але статистика показує, що більшість любителів стратегічних ігр на мобільних пристроях - це чоловіки від 22 до 45 років, такі люди грають одночасно в 2-3 такі гри поміж роботою або навчанням для зняття стресу. [2]

1.3 Ознайомлення з іграми-аналогами

1.3.1 Plants vs. Zombies

Plants vs. Zombies(рис. 1.1, 1.2) - одна з найвідоміших ігор жанру Tower Defence, розроблена PopCap Games на рушії PopCap Framework.

Концепт гри полягає в тому, що гравець захищає домівку від орд зомбі завдяки висадці рослин, котрі є головною обороною силою.

Графіка виконана у 2D-мультяшному стилі з плавною анімацією, а звуковий супровід включає динамічні ефекти та адаптивний саундтрек.

Завдяки простій, але глибокій тактичній механіці, Plants vs. Zombies залишається одним із найуспішніших Tower Defence-проєктів.

Вона використовує лінійну сіткову систему розташування оборонних юнітів - рослин на ігровому полі, поділеному на ряди, за якими рухаються хвилі зомбі. Основна механіка базується на збиранні ресурсів - сонця - за допомогою

сонячників для висадки різноманитних рослин, кожна з яких має унікальні характеристики: атакуючі (горохостріли), сповільнюючі (заморожувальні гриби), вибухові (вишні-бомби) та захисні (горіхи) тощо.

Рис. 1.1 Plants vs. Zombies

Основною задачею є вибудувати економіку та оборону, щоб не пропустити зомбі до кінця ряду. Присутні багато видів зомбі, де кожен тип має особливу поведінку (наприклад, звичайні зомбі просто йдуть вперед, тоді як танцюючі викликають підмогу, а зомбі з драбинами можуть перелазити перешкоди). Гра має поступове ускладнення рівнів, різні режими гри (Survival, Puzzle, Mini-Games) та систему апгрейдів.

Рис. 1.2 геймплей Plants vs. Zombies

Однією з цікавих особливостей гри є також взаємодія зомбі з рослинами - зомбі їдять рослини, а не проходять повз веж як у інших іграх жанру Tower Defence, а також добування ресурсів шляхом посадки сонячників а не знищенням ворогів.

Отже у підсумку перевагами Plants vs. Zombies є:

- Велика різноматність рослин, що забезпечує реіграбільність
- унікальний сеттинг та стиль гри
- розширена взаємодія між рослинами та зомбі

Недоліками же є відносна одноманітсь структури мап - п'ять рядів та фіксовані місця для споруди будівель(в нашому випадку рослин). В Defensor TD ці недоліки вирішені можливістю вільно будувати споруди на мапі, що відкриває багатші можливоті для дизайну мап.

1.3.2 Defense Grid: The Awakening

Defense Grid: The Awakening(Рис. 1.3, 1.4) - це класика жанру Tower Defence, яка була розроблена студією Hidden Path Entertainment у 2008 році. Основним концептом є захист від інопланетних загарбників за допомогою за допомогою будівлі веж.

Першою особливістю цієї гри є те, що маршрутів може бути декілька одночасно, штучний інтелект обирає маршрут для ворогів в залежності від дистанції та кількості веж на кожному маршруті.

Другою особливістю є система ресурсів, основним ресурсом є енергія, яка використовується для будівництва та покращення веж. Гравець отримує енергію за знищення ворогів, а також пасивно з часом.

Головні недоліками Defense Grid: The Awakening є типовими для ігор жанру Tower Defence:

- одноманітність геймплею
- обмежена взаємодія між ворогами та гравцем.

В Defensor TW ці недоліки вирішуються за допомогою ворогів, котрі можуть атакувати будівлі, що дозволить різнообразити геймплей.

Рис. 1.3 Defense Grid: The Awakening

Рис. 1.4 геймплей Defense Grid: The Awakening

1.3.3 Kingdom Rush

Kingdom Rush(Рис. 1.5, 1.6) - це 2D-гра жанру Tower Defence, розроблена студією Ironhide Game Studio на платформі Adobe Flash, а пізніше портована на мобільні пристрої та ПК.

Kingdom Rush має досить стандартний геймплей для ігор жанру Tower Defence. Гра використовує тайлову систему розташування оборонних веж, які будуються в спеціально відведених точках на карті. Геймплей зосереджений на стратегічному плануванні захисту шляхів пересування ворогів за допомогою чотирьох основних типів веж: лучників, магів, артилерії та казарм. Кожна вежа має дві унікальні гілки покращень із додатковими здібностями, такими як отруєння, заморожування або вибуховий урон.

Штучний інтелект ворогів запрограмований на рух за фіксованими шляхами, але різні типи супротивників мають унікальні поведінкові патерни, наприклад, телепортація, бронювання від певних атак або роздвоєння після смерті.

Успіх Kingdom Rush показує, що для створення успішної гри не завжди потрібні радикальні інновації, а гарно виконана перевірена формула може працювати не один раз.

Рис. 1.5 Kingdom Rush

Рис. 1.6 геймплей Kingdom Rush

Недоліки Kingdom Rush є типовими для ігор жанру Tower Defense:

- одноманітність геймплею
- обмежена взаємодія між ворогами та вежами
- Фіксовані позиції для веж

Defensor TW вирішує ці проблеми впровадженням елементів Real Time Strategy.

Таблиця 1.1

Порівняння аналогів до Defensor TD

Показник	**Plants vs. Zombies**	**Defense Grid: The Awakening**	**Kingdom Rush**	Defesor TD
Платформи	Android, Microsoft Windows	Microsoft Windows, Xbox3 60	Browser game, Android, Microsoft Windows	Microsoft Windows, Android
Наявністть більш ніж одного будівельного ресурсу	Ні	Так	Ні	Так
Вільне розміщення споруд на карті	Ні	Ні	Ні	Так
Наявність безкінечного режиму гри	Так	Ні	Так	Так
Наявність ворогів, що атакують будь-які споруди	Так	Ні	Ні	Так
Спосіб добування будівельних ресурсів окрім знищення ворогів	Так	Ні	Ні	Так

1.4 Концепт гри

Головна задача гри - відбити максимальну кількість хвиль ігрових ворогів за допомогою споруд. Гравець може будувати будувати споруди, використовуючі ресурси для будівництва: дерево, каміння та золото. Споруди використовуються для добування ресурсів та знищення ворогів.

Починає гру з Головною Спорудою(HQ) та невеликою кількістю ресурсів. Втрата HQ є умовою поразки гравця.

Вороги генеруються хвилями у заздалігдь визначених точках та рухаються по замовчуванню до HQ, атакуючі зустрічні споруди. Кожна наступна хвиля генерує більшу кількість ворогів.

1.5 Формування вимог до гри

На основі наявних даних можна сформувати наступні вимоги до гри жанру Tower Defence з елементами Real Time Strategy.

Нефункціональні вимоги:

1. Гра має бути доступною на ПК та Android
2. Гра має поєднувати жанр Tower Defence з елементами Real Time Strategy
3. Гра має мати музику стилізований під фентезі саундтрек.
4. Гра має мати 2D графіку
5. Вид на ігровий світ має бути "зверху"
6. Гра має бути помірно складною

Функціональні вимоги:

1. Гравець має мати можливість будувати нові будівлі.
2. Гравець має мати можливість ремонтувати існуючі будівлі.
3. Гравець має мати можливість демонтуати існуючі будівлі.
4. Гравець має мати можливість налаштовувати гучність музики.
5. Гравець має мати можливість налаштовувати гучність ігрових ефектів.
6. Гравець має мати можливість вмикати або вимикати переміщення головної камери, при торканні курсором краю екрану

2 ЗАСОБИ РЕАЛІЗАЦІЇ

2.1 Visual Studio

У якості інтегрованого середовища розробкибуло обрано Visual Studio 2022(рис. 2.1). Visual Studio є середовищем розробки, випущеним корпорацією Microsoft в 1997 році, яке підходить для створення широкого спектру програмних застосунків: Windows додатки, Web додатки, мобільні і хмарні платформи. Воно надає повний набір інструментів і функцій для розробки, тестування і розгортання програм, таких як підтримка багатьох програмних мов : C#, Python Visual Basic, F#, C++ тощо. [3]

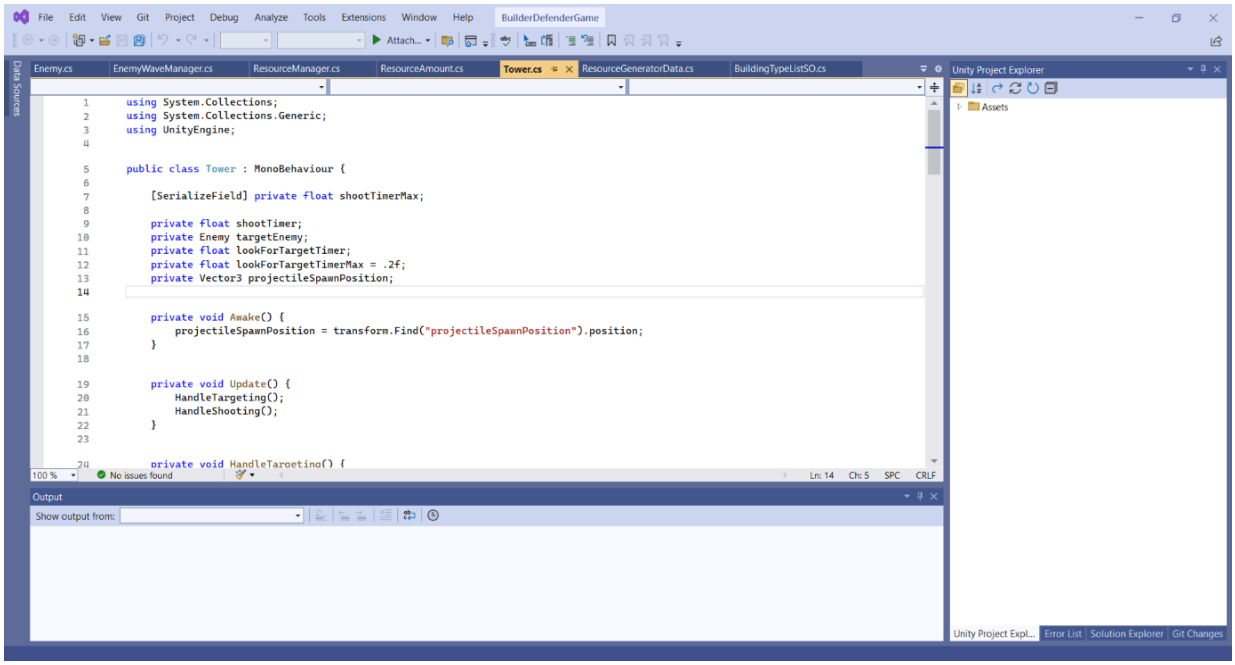


Рис. 2.1 Приклад інтерфейсу Visual Studio

Одна з головних переваг Visual Studio - це самодостатність: неймовірна кількість можлиіих розширень та інструментів, майже будь яку платформу або мову програмування можна інтегрувати в Visual Studio, завдяки чому користувачу майже немає необхідності встановлювати інші середовища програмування.

Visual Studio також зазвичай може виконувати функцію компілятора коду, але у випадку гри жанру Tower Defence з елементами Real Time Strategy

компіляцією коду керує ігровий рушій Unity.

У розробці гри жанру Tower Defence з елементами Real Time Strategy Visual Studio використовується як текстовий редактор для написання коду. В теорії скрипти для Unity технічно можна писати в будь-якому текстовому редакторі, включаючи навіть Notepad, оскільки це звичайні текстові файли з розширенням, яке Unity розпізнає. Однак Visual Studio потрібна для зручності розробки: вона забезпечує підсвічування синтаксису, автодоповнення, інтеграцію з Unity API, швидкий перехід між помилками, налагодження коду та кращу навігацію в проєкті, що значно пришвидшує й полегшує роботу зі скриптами.

2.2 Мова програмування C#

На Unity можна створювати прості додатки без глибоких знань програмування, використовуючи візуальні інструменти, шаблони, готові ассети та системи візуального скриптингу, як Bolt або Unity Visual Scripting, але незнання програмування в Unity суттєво обмежує можливості. Хоча за допомогою візуального скриптингу та готових інструментів можна створити просту логіку чи базову гру, повноцінне керування поведінкою об'єктів, взаємодією, оптимізацією, нестандартними механіками або розширенням функціоналу майже завжди вимагає написання коду. Без програмування складно реалізувати унікальні ідеї або адаптувати проєкт під конкретні потреби. Це призводить до необхідності обрати мову програмування для реалізації проекту. У якості мови програмування було обрано мову C#.

C# — сучасна, об’єктно-орієнтована типобезпечна мова програмування розроблена А́ндерсом Ге́йлсберґом із Microsoft в 2000-му році. C# дозволяє розробникам створювати багато видів безпечних та надійних програм, які працюють на платформі .NET. [4] Мова C# відповідно до ресурсу dou.ua, займає четверте місце по популярності комерційного використання в Україні з часткою 13,3%, поступаючись Python(13,4%), Java(14%) та JavaScript(19,1%). [5]

Головними перевагами мови C# вважаються універсальність - цю мову

використовують для розробки найрізноманітніших додатків: веб-додатків, desktop -додатків, database-додатків, ігор тощо, багатоплатформність- можливість розробляти додатки для різних операційних систем та апаратних платформ, відносно висока швидкість виконання коду, автоматичний збір сміття(automatic garbage collection), Language Integrated Query - інтегрований синтаксис для запитів схожий на SQL.

Мова C# постійно оновлюється та вдосконавлюється, так нариклад в версії C# 11 було покращена скорость виконання программи та оптимізован автоматичний збір сміття, вдосконалено перетворення групи методів а також додатно багато інших дрібних фукцій. [6]

В грі жанру Tower Defence з елементами Real Time Strategy C# використовується для написання скриптів для Unity - частин коду, які дозволяють керувати поведінкою об'єктів у грі, вони зазвичай прикріплюються до об'єктів у сцені. Через скрипти можна задавати логіку взаємодії гравця та інших ігрових елементів з оточенням:

- рух об'єктів
- анімації
- фізику
- реакції на події
- зміну станів тощо.

Коли гра запускається, Unity автоматично виконує ці скрипти відповідно до життєвого циклу об'єктів.

2.3 Ігровий рушій Unity

Unity(Рис. 2.2, 2.3) — це сучасний кросплатформовий ігровий рушій, розроблений компанією Unity Technologies. Він широко застосовується у створенні 2D- і 3D-ігор, інтерактивних додатків, віртуальної та доповненої реальності, а також симуляцій і візуалізацій. Його архітектура, гнучкість та широкий набір інструментів роблять Unity універсальним рішенням як для інді-

розробників, так і для великих студій.[7]

Основною мовою програмування в Unity є C#, що забезпечує високу продуктивність, чистий синтаксис та інтеграцію з потужними середовищами розробки, зокрема Visual Studio, JetBrains Rider та Visual Studio Code. Також підтримується написання шейдерів на ShaderLab та HLSL, а для деяких інструментів доступна інтеграція з Python.

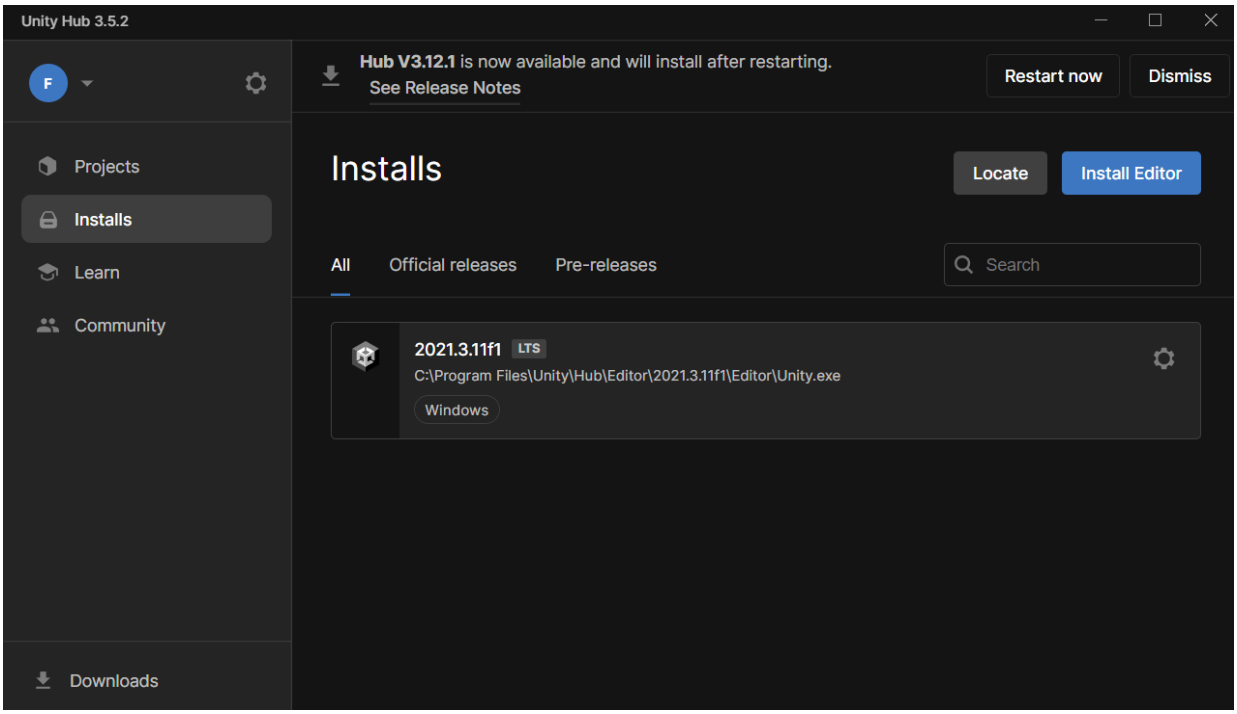


Рис. 2.2 Інтерфейс Unity Hub - лаунчер двигуна Unity

Рушій підтримує велику кількість платформ, зокрема Windows, macOS, Linux, Android, iOS, WebGL, PlayStation, Xbox, Nintendo Switch, Meta Quest, Apple Vision Pro тощо. Unity дозволяє створювати універсальний код з мінімальними змінами для експорту на різні цільові платформи.

Графічна система включає три основні рендер-пайплайни: Built-in Render Pipeline, Universal Render Pipeline — оптимізований для кросплатформенних проєктів, і High Definition Render Pipeline — для високоякісного графічного контенту, з підтримкою фізично коректного рендерингу, постобробки, освітлення та тіней.

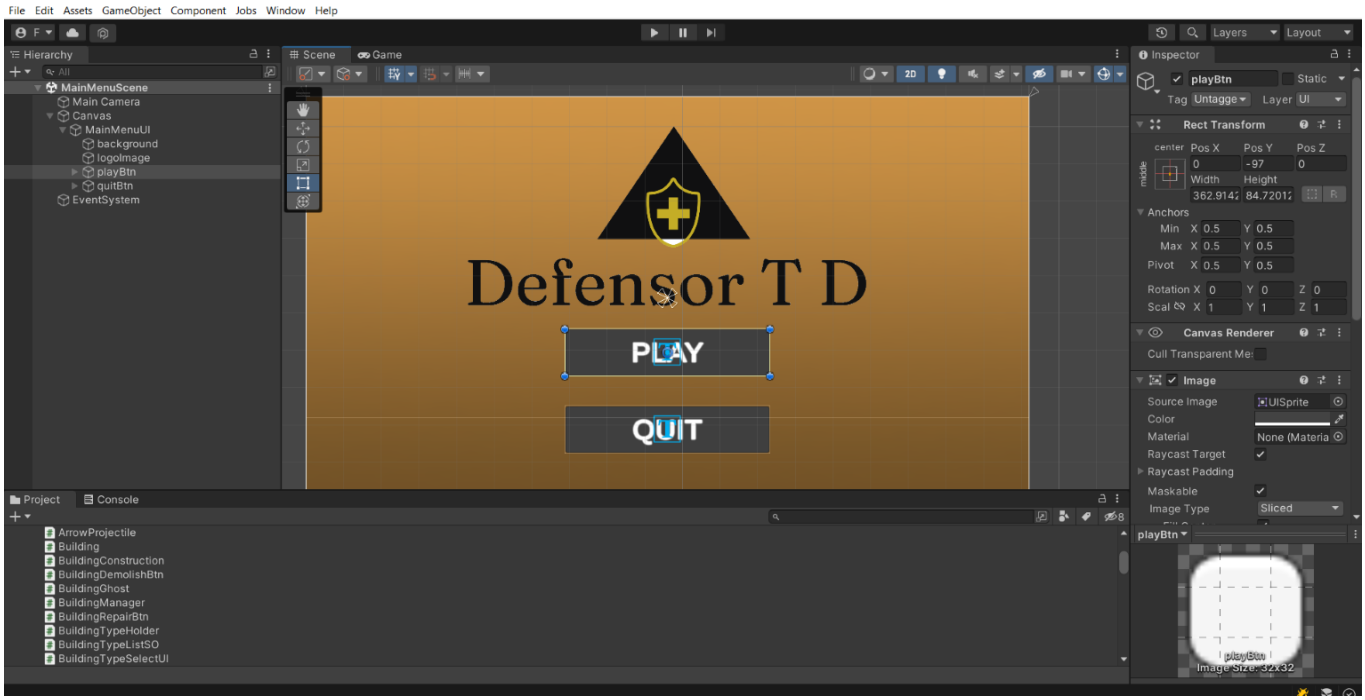


Рис. 2.3 Приклад інтерфейсу Unity

Окрім цього Unity підтримує розширення функціональності за допомогою сторонніх плагінів, які можна додавати через Unity Asset Store або вручну. Також доступна можливість підключення нативних бібліотек і Dynamic-link library через P/Invoke. Завдяки цьому рушій може бути інтегрований у великі, промислові проєкти або розширений власними інструментами.

Також Unity забезпечує підтримку візуального редактора сцен, у якому розробник може працювати зі світлом, камерами, фізичними колайдерами, аудіо, анімацією та іншими системами. Також доступні розширені інструменти профілювання, дебагу, тестування та аналітики.

В реалізації гри жанру Tower Defence з елементами Real Time Strategy ігровий рушій Unity виконує ключову роль.Через Unity реалізуються візуальні складові гри:

- рендеринг графіки,
- створення та відображення візуального інтерфейсу користувача,
- створення та налаштування анімацій,
- створення шейдерів, масок,
- налаштування освітлення тощо.

Окрім візуальної складової Unity також необхідний для реалізації логіки та

інших складових гри:

- Стовення та редагування ігрових сцен
- Створення та редагування ігрових об'єктів
- Керування компіляцією коду гри
- Надання бібліотек класів для C#
- Керування взаємодією ігрових об'єктів на сцені
- Виконання ролі тестового майданчика для контролю якості
- Обробка аудіо складових гри таких як музика та звукові ефекти
- Надання доступу до ігрових Asset'ів через вбудований Asset Store
- Конвертація проекту у готовий build - готовий додаток, який можна публікувати

2.4 Система контролю версій Github

GitHub – це хмарна платформа для розробки програмного забезпечення, побудована на принципах розподіленого контролю версій і мікросервісної архітектури. GitHub використовується для зберігання, керування та спільної розробки програмного коду за допомогою системи контролю версій Git, дозволяючи розробникам ефективно відстежувати зміни, працювати в команді, автоматизувати робочі процеси та забезпечувати безпечний доступ до проєктів. Також GitHub є дуже популярний серед розробників Open Source додатків дозволяє публікувати вихідний код. [8]

У командних проектах Github зазвичай виконує роль майданчика, на якому різні розробники працювати над обним проєктом, обговорютувати тавносити власні зміни через pull request, відслідковувати історію змін до найменших деталей.

У реалізації гри жанру Tower Defence з елементами Real Time Strategy система котролю версій Github виконує наступні функції:

- Консервація старих версій, на випадок, якщо проєкт буде втрачено або пошкоджено
- Завантаження вихідного коду гри в вільний доступ

3 ПРОЄКТУВАННЯ

3.1 Огляд класів Unity Monobehaviour та Scriptable Object

Перед тим, як приступити до проєктування гри, необхідно зробити огляд двух ключових класів Unity - Monobehaviour та Scriptable Object, оскільки більша частина класів гри наслідують ці класи і без розуміння контексту того, як вони працюють і для чого використовуються, опис архітектури може бути не зрозумілим.

3.1.1 Класс Monobehaviour

Monobehaviour (Рис. 3.1) - базовий клас Unity, який є основою для більшості скритів Unity, більша частина скриптів успадковують цей клас.

Основна особливість Monobehaviour є в тому, що його та класи-спадкоємці можна прикріпляти до ігрових об'єктів в інспекторі Unity і завдяки цьому керувати їхньою поведінкою завдяки великій кількості гнучких методів, котрі можуть редагувати інформацію ігрового об'єкта на старті сцени, кожний фіксований проміжок часу, дає можливість отримати доступ до інших об'єктів на сцені тощо.

Одним з недоліків класу Monobehaviour є те, що він не зберігає зміну інформації між різними сценами, тому щоб наприклад зберегти кількість грошей або здоровья між рівнями потрібно використовувати інші класи.

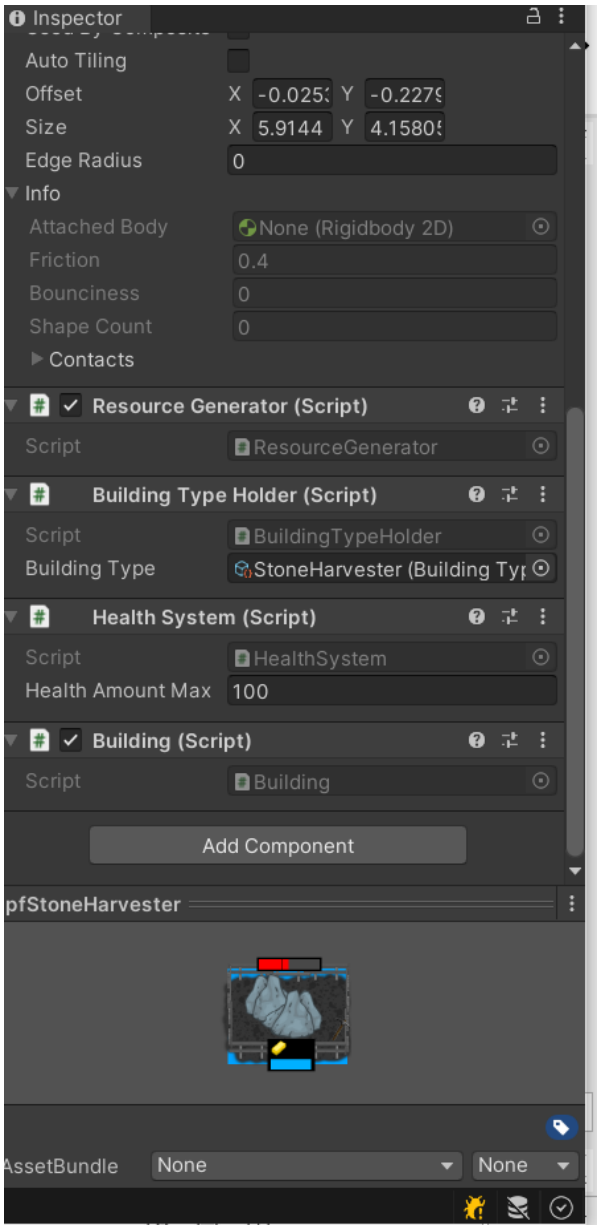


Рис. 3.1 Приклад декількох скритів Monobehaviour прикріплених до ігрового об'єкта

3.1.2 Класс Scriptable Object

Scriptable Object(рис. 3.2, 3.3) - це базовий клас Unity, котрий слугує як контейнер інформації між сесіями в редакторі. Основна різниця між Monobehaviour та Scriptable Object полягая в тому, що Scriptable Object не можна прикріпити до ігрового об'єкта напряму, але об'єкти класу Scriptable Object можна зберігати як asset'ти у провіднику та редагувти їх як файли.

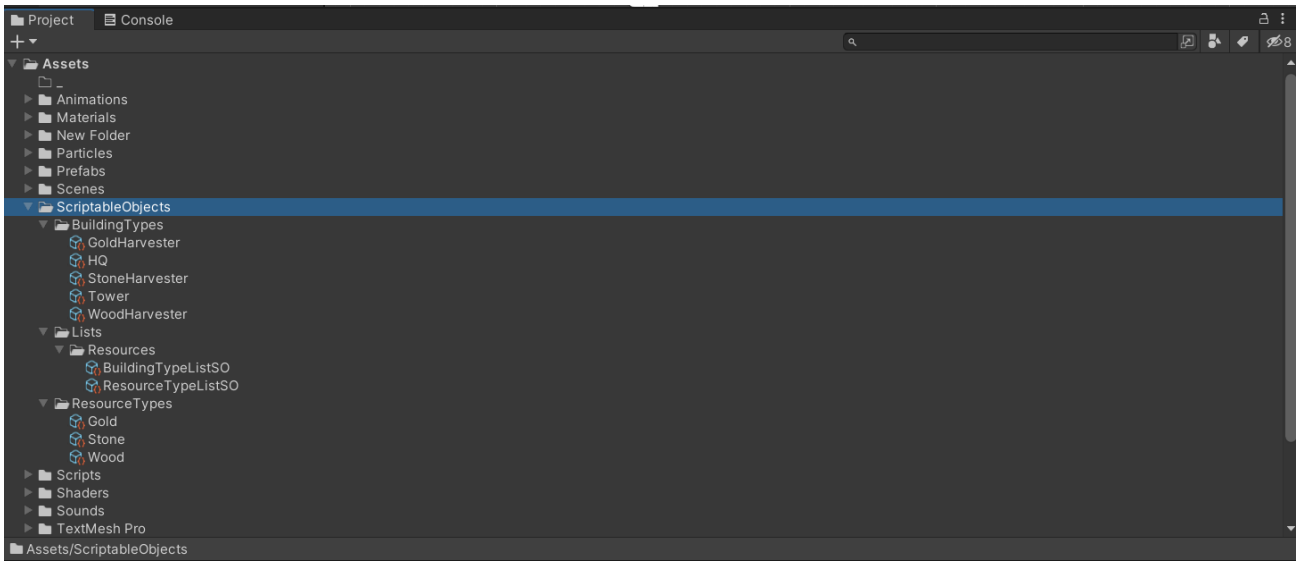


Рис. 3.2 Scriptable Object'ти у папках проєкту

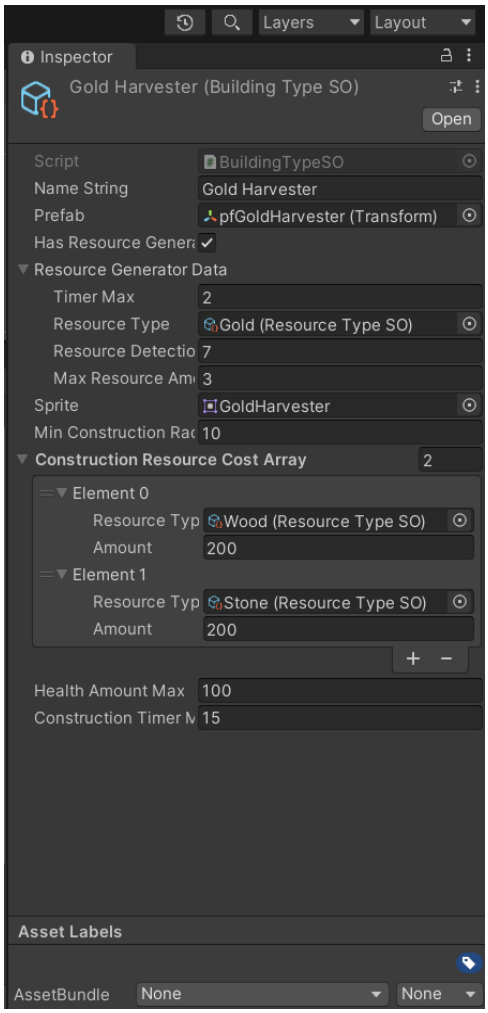


Рис. 3.3 Приклад редагування даних об'єкта Scriptable Object через інспектор Unity

Основне призначення Scriptable Object - зберігання інформації. Якщо один

ігровий об'єкт використовує декілька скриптів клвсу Monobehaviour, то більшу частиеу необхідної інформації для всіх скриптів можна зберегти в відповідному об'єкті класу Scriptable Object, це набагато зручніше, ніж редагувати декілька скриптів Monobehaviour індивідуально, особливо у великих проєктах.

Також Scriptable Object може зберігати зміну інформації між сценами, що дуже зручно для серіалізації даних.

Іншою перевагою Scriptable Object є більша ефективнісь використання ресурсів пристрою, бо дані Scriptable Object не дублюються для кожного екземпляру, що є перевагою при оптимізації гри для слабких пристроїв.

3.2 Проєктування гри

Перед тим як приступити до проєктування класів гри, необхідно розглянути основні сценарії використання гри.

3.2.1 Сценарії використання

Відповідно до сформованих вимог до гри можна спроєктувати діаграму прецедентів.(рис. 3.4)

Відповідно до діаграма прецедентів та вимог до гри, геймплей гри Tower Defence з елементами Real Time Strategy полягає в:

- будівництві будівель для добування ігрових ресурсів
- будівництві захисних споруд
- ремонті існуючих будівель
- демонтажу існуючих бідівель, частинна вартості будівлі повертається гравцю
- технічні функції(налаштування тощо)

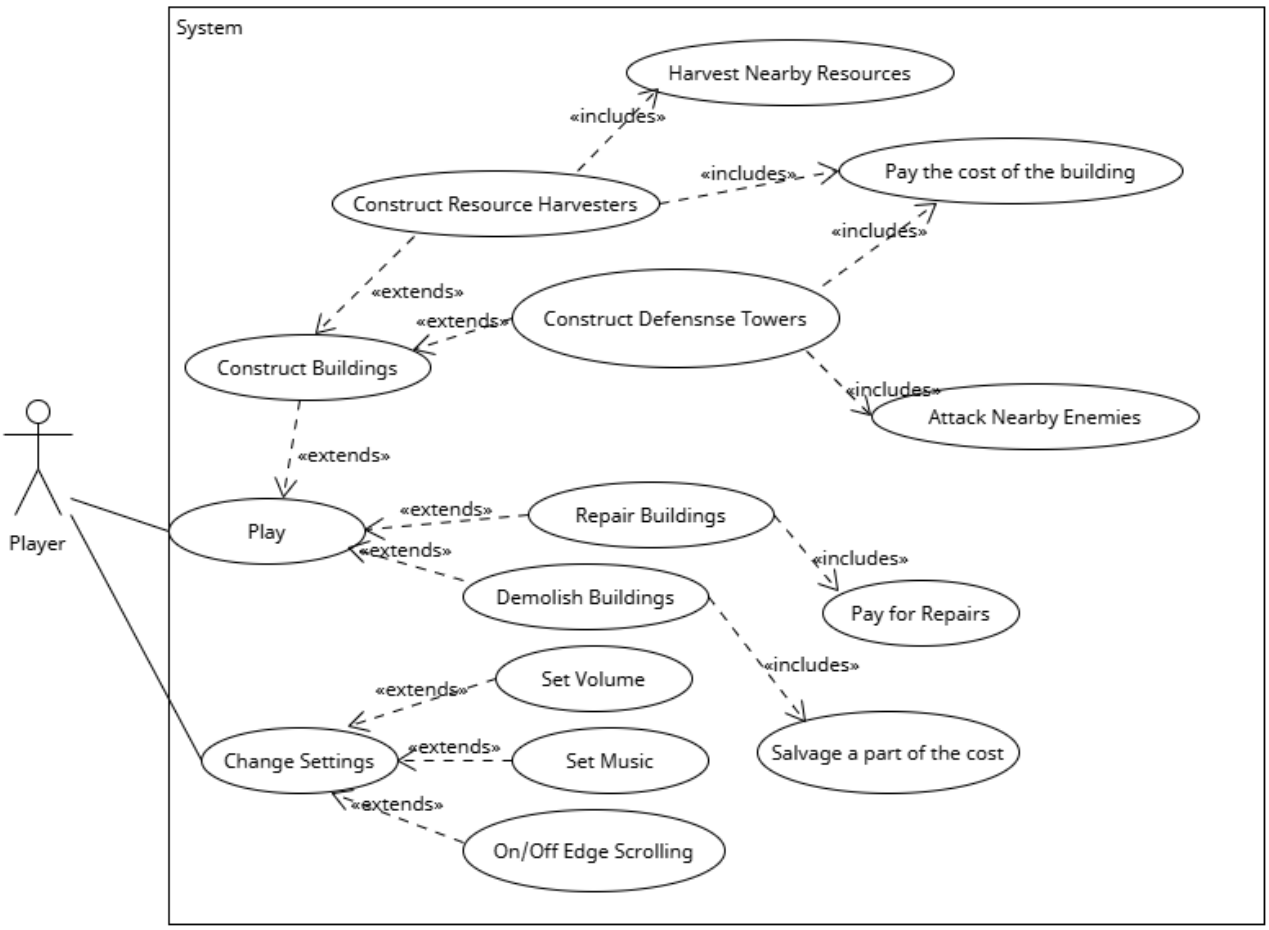


Рис. 3.4 Діаграма прецедентів

3.2.2 Принципи SOLID та KISS

Основою проєктування класів гри жанру Tower Defence з елементами Real Time Strategy є принципи KISS та SOLID.

SOLID - акронім з п'яти принципів Об'єктно Орієнованого Програмування, що допомагають формувати код:

- Single Responsibility Principle - кожний клас має мати відповідати за одну фунцію.
- Open/Closed Principle - кожний клас має бути відкритим для розширення але закритим для змін.
- Liskov Substitution Principle - використання підкласів замість маткринського класу не має викликати помилок.
- Interface Segregation Principle - використання маленьких спеціалізованих інтерфейсів замість одного великого.

- Dependency Inversion Principle - класи не мають бути надто залежні один від одного та використовувати абстракції. [9]

KISS - абревіатура "Keep It Simple, Stupid", це принцип дизайну систем, який передедбачає, що системи мають бути настільки простими, наскількі можливо. Складні рішення мають уникатися, коли можливо. Цей принцип використовується у розробці програмного забезперчення, дизайні інтерфейсів користувача, дизайні продуктів та навіть у військовій сфері.

У розробці програмного забезпечення принцип KISS сприяє створенню зрозумілих, легких у підтримці та масштабуванні рішень. Простий код зменшує ймовірність помилок, полегшує тестування, пришвидшує розробку.[10]

Принципи SOLID та KISS не є жорсткими правилами, скоріше рекомендаціями, вони допомагають писати легший для читання код, допомагають легше розширювати додатки та ігри з меншою кількістю помилок.

3.2.3 Шаблони проєктування

При розробці програмного забезпечення розробники часто зтикаються з аналогічними проблемами. Шаблони проєктування - це загальноприйняті способи проєктування структури програм, які дозволяють вирішувати проблеми архітектури, котрі були сформовані та оптимізовані розробниками. Програмісти навіть можуть використовувати шаблони проєктування неосвідомлено, не маючи теорітичного знання. Розуміння шаблонів проєктування є ключом до побудови зрозумілих та підтримуваних програмних систем.

При розробці гри жанру Tower Defence з елементами Real Time Strategy, були використані шаблони проєктування: Singleton, Factory, Observer.

Singleton - шаблон проєктування, який передбачає наявність лише одного екземпляру класу одночасно, оголошення статичного поля, котре є посиланням на власний екземпляр та методи доступу до нього, що дає глобальний доступ усім іншим класам до екземпляру цього класу. Шаблон Singleton добре підходить для проєктування менеджерів гри та налаштувань. Але треба бути обережним, бо зловживання шаблоном Singleton може сильно порушити принцип Single

Responsibility.

Factory - шаблон проєктування, котрий використовує фабричні класи або методи, котрі можуть створювати екземпляри класів без виклику конструктора класа у коді. У грі жанру Tower Defence з елементами Real Time Strategy Factory використовується для побудови будівель, генерації ворогів, деяких елементів інтерфейсу та снарядів веж.

Observer — шаблон проєктування, котрий визначає залежність "один-до-багатьох" між об’єктами таким чином, що при зміні стану одного об’єкта усі його залежні об’єкти (спостерігачі) автоматично отримують повідомлення та оновлюють свій стан. У грі жанру Tower Defence з елементами Real Time Strategy Observer використовуэться у менеджеры будівництва, котрий повідомляє про вибрану споруди для будівництва. [11]

3.2.4 Проєктування класів гри

На наступній діаграмі класів(рис. 3.5) зображено принцип роботи основних класів, котрі відповідальні за функіонування будівель гри жанру Tower Defence з елементами Real Time Strategy.

Кожна функція будівлі має окремий скрипт, відповідальний за цю функцію, згідно принципу Single Responsibility. Скрипти-підкласи Monobehaviour зазвичай відповідальні за ігрові механіки, в той час як Scriptable Object - за збереження інформації.

Клас HealthSystem - це універмальний клас, котрий відповідає за наявність здоров'я у ігрового об'єкту та має наступні функції

- Задання масимальної кідькості одиниць здоров'я
- Відстеження наявної кількісті одиниць здоров'я
- Маніпуляція кількістю одиниць здоров'я
- Підтвердження, що об'єкт "живий" або "мертвий"(залежить чи кількіть одниць здоро'я більша чи менша за 0).

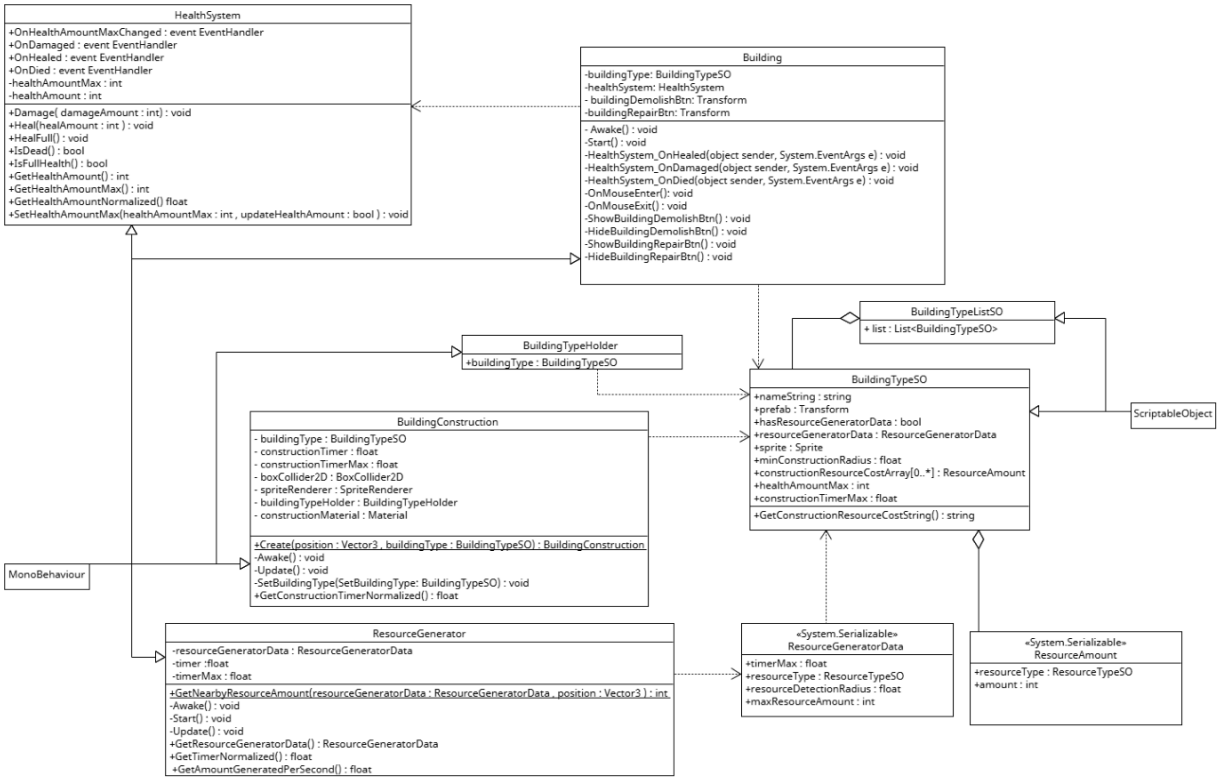


Рис. 3.5 Діаграма основних класів будівель

Клас ResourceGenerator - клас економічних споруд, котрий відповідальний за пошук родовищ ресурсів відповідних до типу споруди та власне добування тих рксурсів. Але щоб зробити процес добування ресурсів цікавішим, будівлі зі скриптом ResourceGenerator мають наступні обмеження:

- Не можна розміщувати дві ідентичні споруди занадто близько
- Чим більша кількість родовищ поруч з добуваючою спорудою, тим ефективніша добича.

Якби не було цих обмежень, то гравець міг би просто побудувати велику кількіть однакових споруд вокруг одного родовища, щоб дуже обмежило стратегічну складову гри.

Побудова будівель не моментальна, перед тим як будівля почне функціонувати, створюється клас BuildingConstruction. Об'єкт класу BuildingConstruction створюэться з таймером, під час його існування грається анімація побудови, по завершенню таймера об'єкт знищується та на його місці створюється готова будівля.

BuildingTypeSO - підклас ScriptableObject, об'єкти котрого містять усі вхідні характеристики будівлі, котрі потім передаються витягаються та передаються в інші класи: вартіть побудови, дані для добування ресурсів, дані захисних споруд, кількість одиниць здоровья, швидкість побудови, посилання на префаб тощо. Коригувати та відслідковувати усі ці дані в одному місці набагато зручніше, ніж в декількох різних класах.

BuildingTypeListSO використовується для зберігання списку об'єктів класу BuildingTypeSO, на основі цього списку генерується інтерфейс користувача, а саме список споруд для побудови.

BuildingTypeHolder використовується для посилання на об'єкт BuildingTypeSO, оскільки підкласи ScriptableObject не можна прикріпляти до об'єктів Unity. Завдяки скрипту BuildingTypeHolder інші класи ігрового об'єкту можуть отримати доступ до даних об'єкту BuildingTypeSO.

ResourceGeneratorData містить дані, який ресурс добуває ResourceGenerator, у якому радіусі та в якій кідькості. Цю інформацію виділено у окремий клас, щоб її можна було редагувати у класі BuildingTypeSO.

Клас Building відповідає за витягування даних з скрипту BuildingTypeHolder, та передачу цих даних до скрипту HealthSystem, наявність елементів інтерфейсу кнопок для ремонту та демонажу будівлі, та передачу даних про ремонт тп пошкодження до HealthSystem.

На рис. 3.6 зображені класи, пов'язані з основними класами-менеджерами гри.

ResourceManager - клас, який відповідає за типи, кількісь та стартову кількість ресурсів. Також цей скрипт контролює зміну кількості рескрсів перевіряє, чи може гравець дозволити собі побудову нової будівлі, або ремонт існуючої. будівлі.

Клас EnemyWaveManager відповідає за генерацію хвиль ворогів:

- кількість ворогів,
- локацію створення порогів,
- частоту генерації хвиль.
- відстеження номеру хвилі.

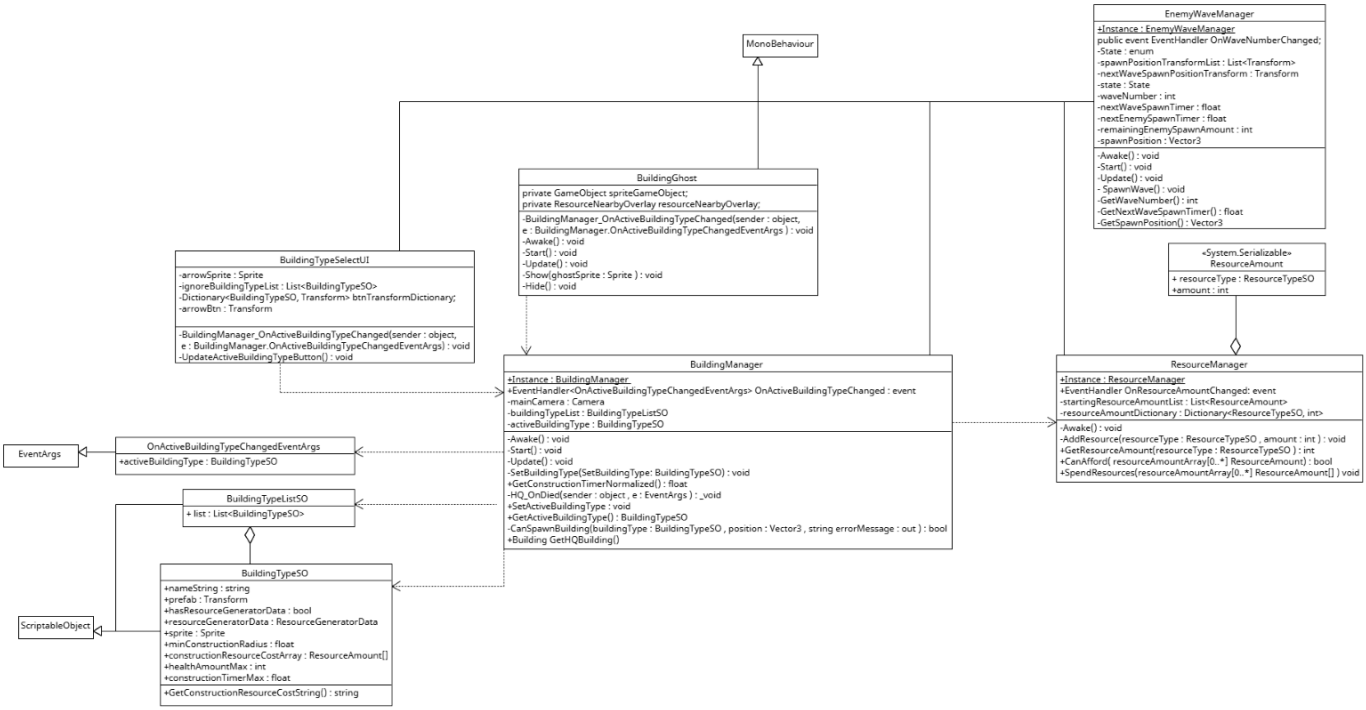


Рис. 3.6 Діаграма класів менеджерів гри

BuildingManager - клас, що відповідає за побудову споруд:

- завантаження даних списку споруд з BuildingTypeListSO
- збереження виду активної споруди для побудови
- розсилку події про зміну вибору активної споруди

Клас BuildingGhost відображає проекцію обраної споруди перед побудовою, що дозволяє краще планувати місце розташування будівлі.

Клас BuildingTypeSelectUI відповідає за елементи користувацького інтерфейсу, через які гравець може обирати тип споруди, який він хоче побудувати, іконки для кожної споруди генеруються у відповідності з даними з BuildingTypeListSO.

На рис. 3.7 зображено повний список класів гри жанру Tower Defence з елементами Real Time Strategy.

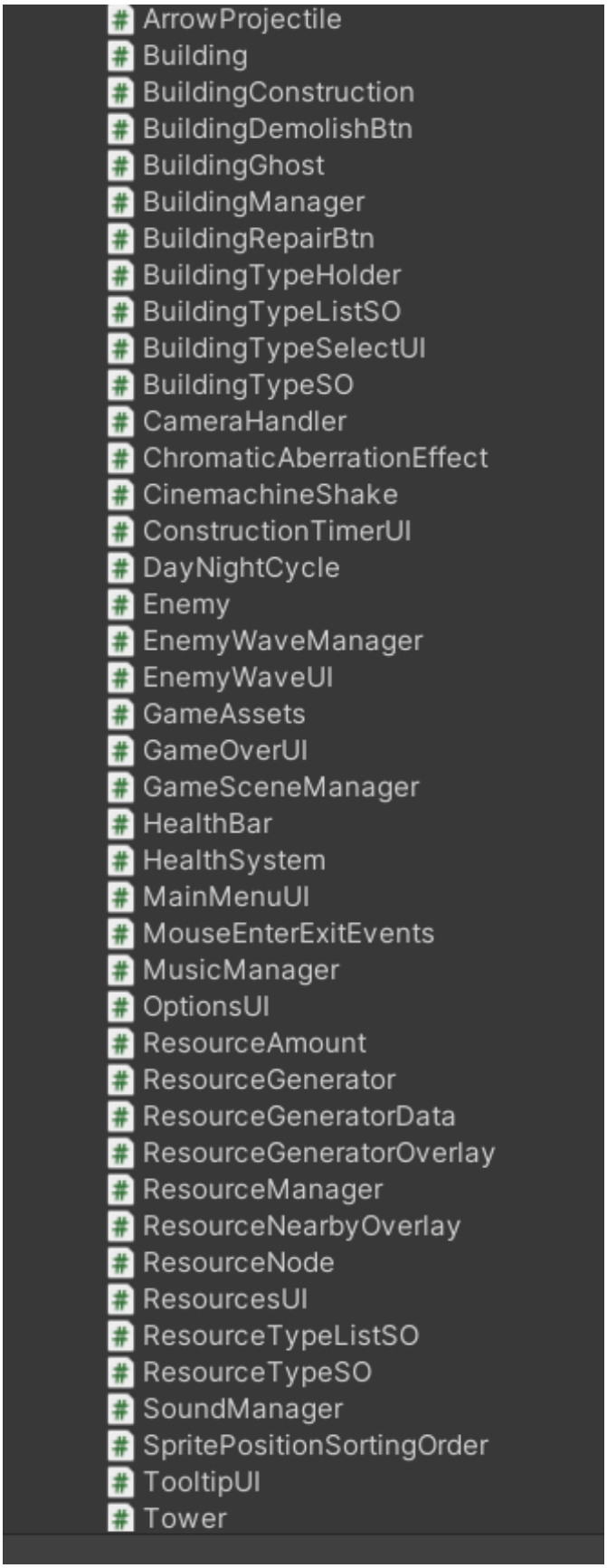


Рис. 3.7 Повний список класів Defensor TD

4 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

4.1 Покроковий опис процесу реалізації гри

Покроковий опис процесу процесу реалізації гри жанру Tower Defence x елементами Real Time Strategy:

1. Першим кроком є створення нового 2D проєкту Unity(Рис. 4.1), новий проєкт містить майже пусту сцену сцену гри з об'єктом Main Camera.

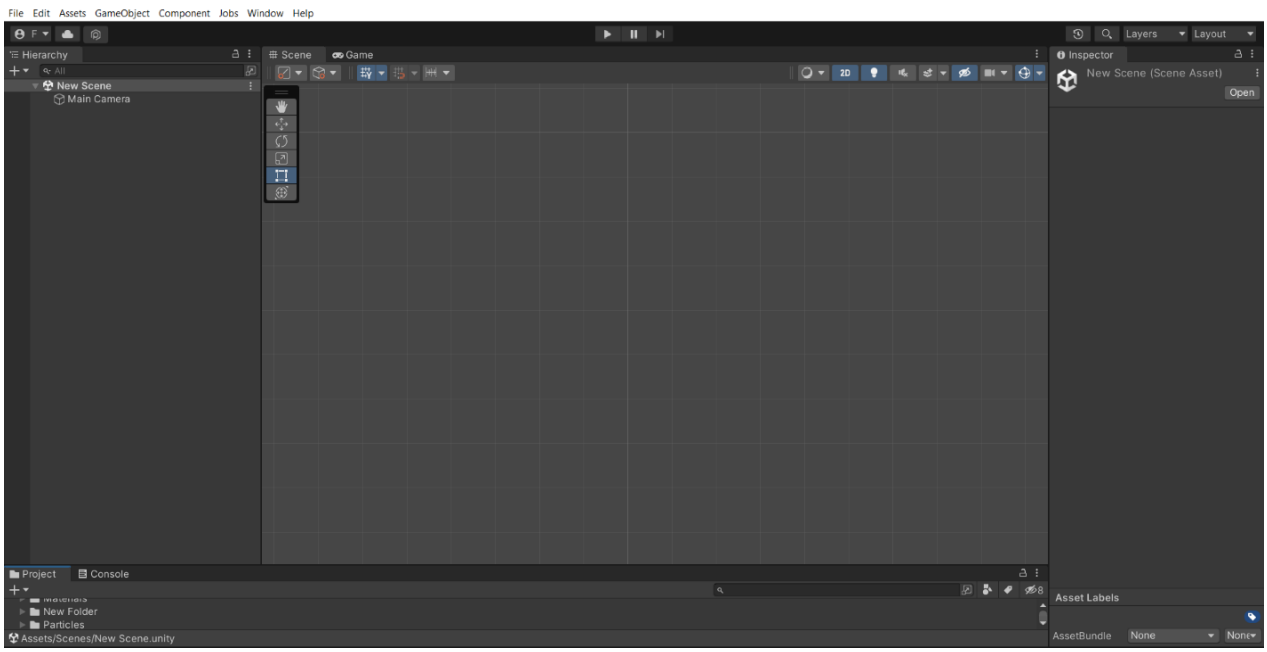


Рис. 4.1 нова сцена Unity

2. Другим кроком було налаштування проєкту: імпорт Universal Render Pipeline та Text Mesh Pro. Импортовані асети з Asset Store: спрайти, аудіофайли, анімації.

Universal Render Pipeline - частина системи рендерингу Unity, яка основана на базі Scriptable Render Pipeline. Universal Render Pipeline налаштувується через об'єкти, що є підкласами ScriptableObject. Та відповідає за керування рендерингом графіки проєкту.

Text Mesh Pro - розширена текстова система в Unity, яка забезпечує

високоякісне відображення тексту з гнучкими можливостями стилізації. Text Mesh Pro було розпочато як асет розроблений користувачами, але набув такої популярності, що де-факто став вибором текстової системи за замовчуванням

3. Початок розробки системи побудови будівель - при натискані курсором генерується вибраний об'єкт кдасу Building зі списку списку BuildingTypeListSO. Обирання будівлі оскільки інтерфейс поки не реалізований відбувається через натискання клавіши, а вибрана відстежується через консоль. Створено класи Building, BuildingManager, BuildingTypeSO, BuildingTypeListSO та BuildingTypeHolder. Створено префаби для основних видів споруд(рис. 4.2): HQ, Wood Harvester, Stone Harvester, Gold Harvester, Tower.

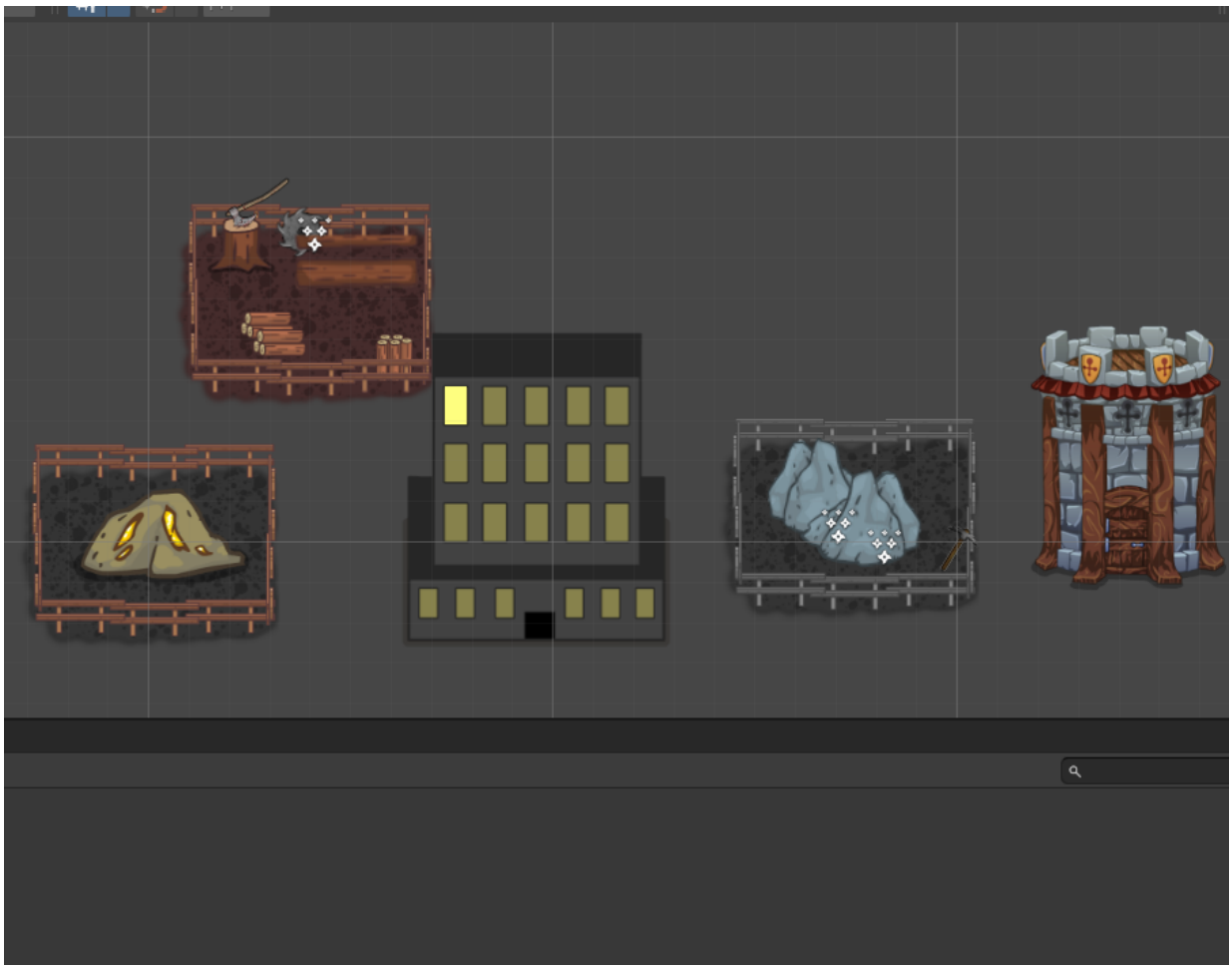


Рис. 4.2 Спрайти головних споруд гри

4. Початок розробки системи системи ресурсів. Створені класи ResourceManager, ResourceTypeSO, ResourceTypeListSO, ResourceAmount. Створені

асети Scriptable Object для зберігання інформації щодо кількості та типів ресусів(рис. 4.3) Створені механізми додавання та віднімання ресурсів, вони тестуються через виклики відповідних методів через натискання класів та відстеження змін значення змінних через консоль

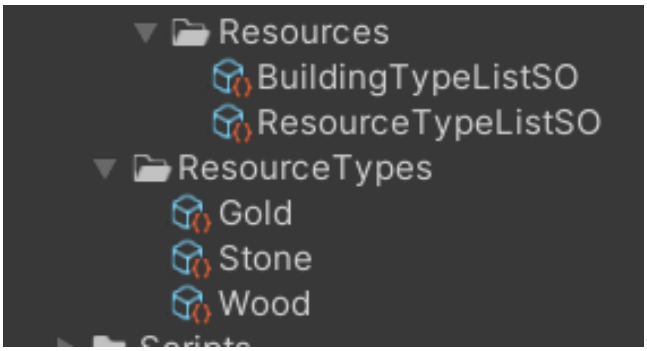


Рис. 4.3 Створені типи ресурсів

5. Релізації системи добування ресурсів за допомогою будівель, створені класи ResourceGenerator, ResourceGeneratorData. Будівлі з прикріпленим скриптом ResourceGenerator викликають метод додання відповідного ресурсу через кожний відведений проміжок часу.

Від кількості родовищ ресурсів залежить швидкість добичі ресурсів екстрактором

6. Початок реалізації інтерфейсу гравця: інтерфейс, котрий відображає кількість ресурсів гравця(рис. 4.4) та інтерфейс вибору споруди, створено класи ResourcesUI та BuildingTypeSelectUI(Рис. 4.5)

Інтерфейс ресурсів являє собою не інтерактивний список з іконок ресурсів та їх відповідних значень.

Інтерфейс вибору споруди є списком кнопок з іконками будівель та кнопкою з зображенням курсора, який аннулює вибір. Вибрана кнопка відображається білою рамкою навколо.

Для реалізації інтерфесу використовується елемент Unity Canvas. Це ключовий компонент системи інтерфейсу користувача Unity, котрий визначає, як користувацький інтерфейс відображається на екрані. Canvas виступає кореневим

елементом для всіх об'єктів інтерфейсу користувача та відповідає за їхнє розташуванняя, масштабування та порядкок рендерингу.

Рис. 4.4 Інтерфейс ресурсів

Рис. 4.5 Інтерфейс вибору споруди

7. Допрацьовано систему добування ресурсів: створені місценароженя різних типів ресурсів класу ResourceNode(рис. 4.6). Ресурси тепер добуваються тільки, якщо поруч з екстрактором ресурсів є відповідне місценародження ресурсів.

Рис. 4.6 Спрайти місценароджень ресурсів

8. Створення дизайну рівня, створення фону, розміщення родовищ ресурсів.(рис. 4.7)

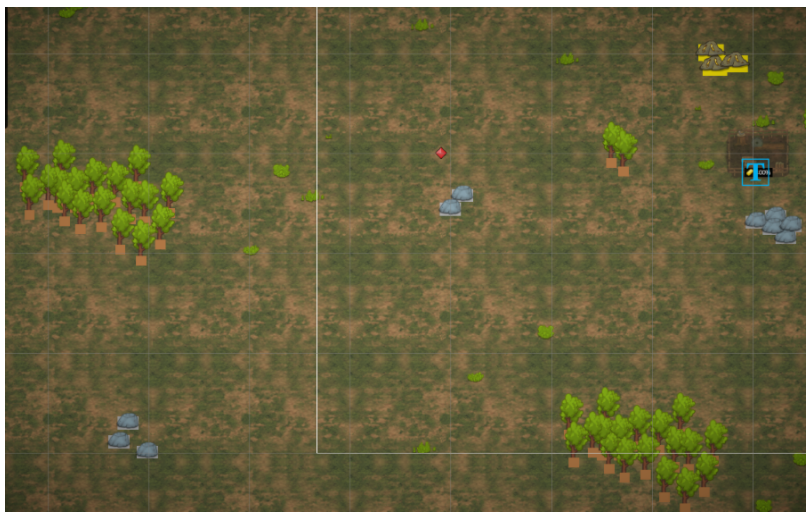


Рис. 4.7 фрагмент дизайну рівня

9. Створення проєкції вибраної будівлі перед розташуванням, створено клас BuildingGhost.(Рис. 4.8) . Принцип роботи: при виборі будівлі в інтерфесі, за курсором починає слідувати полузпрозора версія спрайту відповідної будівлі, що дозволяє легше вибирати місце для побудови споруди.

Рис. 4.8 Building Ghost, проєкція будівлі

10. Допрацьовано систему будівлі споруд, додано умови розташування будівель:

- В гравця має бути достатньо ресурсів
- Місце розташування має бути вільним
- Не можна розташовувати споруди одного типу поруч один з одним

11. Додано елемент інтерфейсу відповідальний за підказки для гравця - TooltipUI. За певних умов, TooltipUI викликається з відповідним текстом біля курсору гравця.

Він відображає такі елементи як:

- Ціну будівлі котру хоче побудувати гравець(Рис. 4.9), умова виклику: курсор парить над відповідною кнопкою.
- Причину, чому гравець не може побудувати вибрану будівлю((Рис. 4.10), умова виклику: гравець намагається побудувати споруду не невиконавши умови розташування, викликається відповідне повідомлення.
- Еффективність добування ресурсів при розташування економічних споркуд.

Рис. 4.9 Tooltip UI - відображення ціни споруди

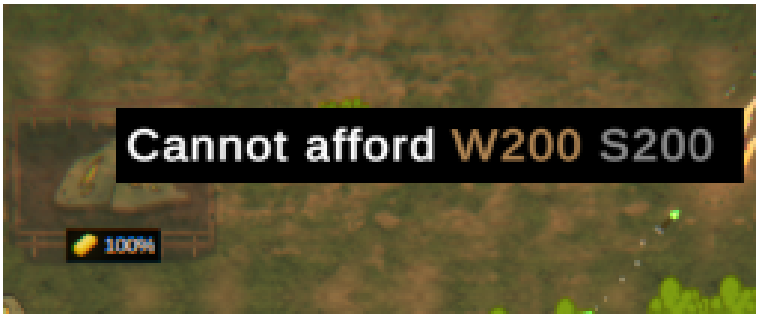


Рис. 4.10 Tooltip UI - відображення причини неможливості побудови споруди

12. Створення системи здоров'я HealthSystem, котра регулює стартову та максимальну кількість одиниць здоров'я, має методи додавання та віднімання одиниць здоров'я, метож знищення об'єкту, якщо одиниць здоров'я стає нуль або менше. Система здоров'я була додана до всіх будівель.

13. Створення елементів інтерфейсу, пов'язаних з системою з системою здоров'я - полоску здоров'я класу HealthBar(рис. 4.11). Наповнення полоси здоров'я пропорціональна відношенню наявної кількості здоров'я до максимальної

Рис. 4.11 Приклад інтерфейсу полоси здоров'я

14. Був створений клас ворогів Enemy(Рис. 4.12). Поведінка ворогів передбачає рух до головної будівлі та атаку усіх будівель, що зустрічаються на шляху. При контакті будівлі з ворогом, ворог знищується, а будівля втрачає певну частину одиниць здоров'я.

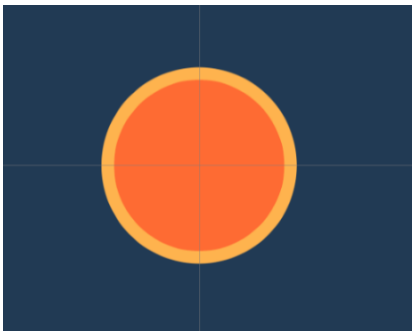


Рис. 4.12 Спрайт ворога

15. Розроблено менеджер хвиль класу EnemyWaveManager, який регулює генерацію ворогів:

- Частоту генерації хвиль
- Кількість згенерованих ворогів
- Локацію генерації ворогів

16. Розроблено елементи інтерфейсу користувача класу EnemyWaveUI відповідальний за:

17. Зображення кількості хвиль, котрі витримав гравець.(Рис. 4.13)

Рис. 4.13 Зображення кількості хвиль, котрі витримав гравець.

- Таймер відрахування наступної хвилі.(Рис. 4.14)

Рис. 4.14 Таймер відрахування наступної хвилі.

- Індикатор локації генерації наступної хвилі.(Рис. 4.15)

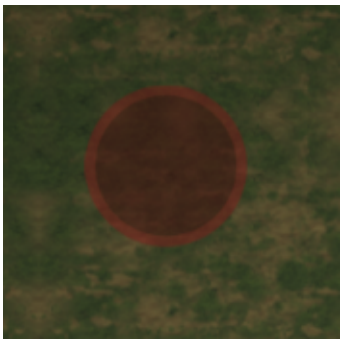


Рис. 4.15 Індикатор локації генерації наступної хвилі

- Вказівник, котрий вказує направлення локації генерації наступної хвилі, коли вона поза екраном. (Рис. 4.16)

Рис. 4.16 Вказівник, котрий вказує направлення локації генерації наступної хвилі, коли вона поза екраном.

- Вказівник, що вказує на ворогів поза екраном. (Рис. 4.17)

Рис. 4.17 Вказівник, що вказує на ворогів поза екраном.

18. Створен клас веж - Tower та їхніх стріл - ArrowProjectile(Рис. 4.18), котріє основою оборони гравця. Вежи випускають стріли по найближчих ворогах в радіусі ураження(рис. 4.19), стріли при колізії з ворогом віднімають в ворога певну кількість одиниць здоров'я. Скрипт Tower був також доданий до Головної Будівлі.

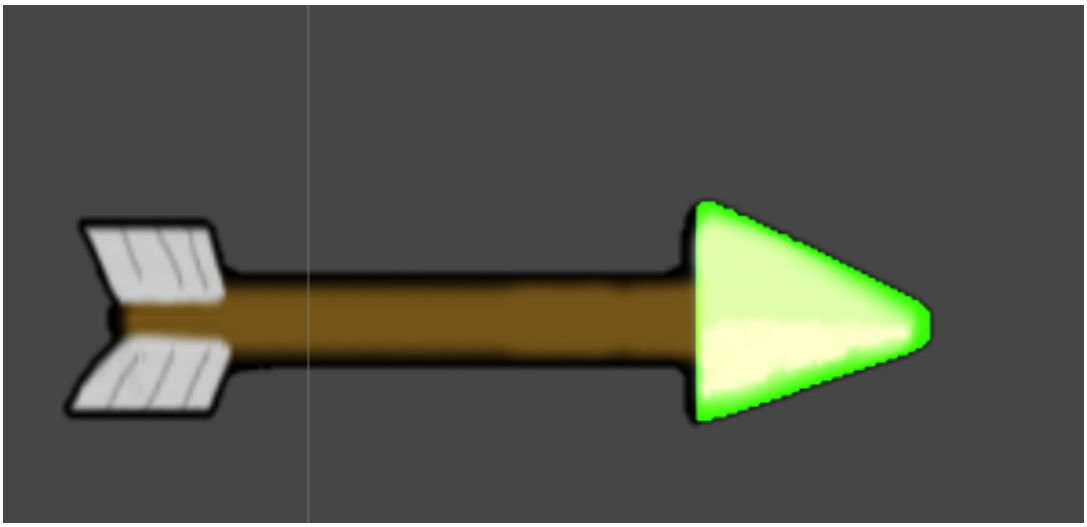


Рис. 4.18 Спрайт ArrowProjectile

Рис. 4.19 Приклад роботи скрипту Tower

19. Допрацьована система будівництва, створе клас BuildingConstruction, тепер будівлі не генеруються моментально, а консрукція будівель займає певний

чам.

20. Доданий елемент інтерфейсу - мінікарта, котра відображає розташуваня родовищ ресурсів та побудованих будівель(Рис. 4.20).

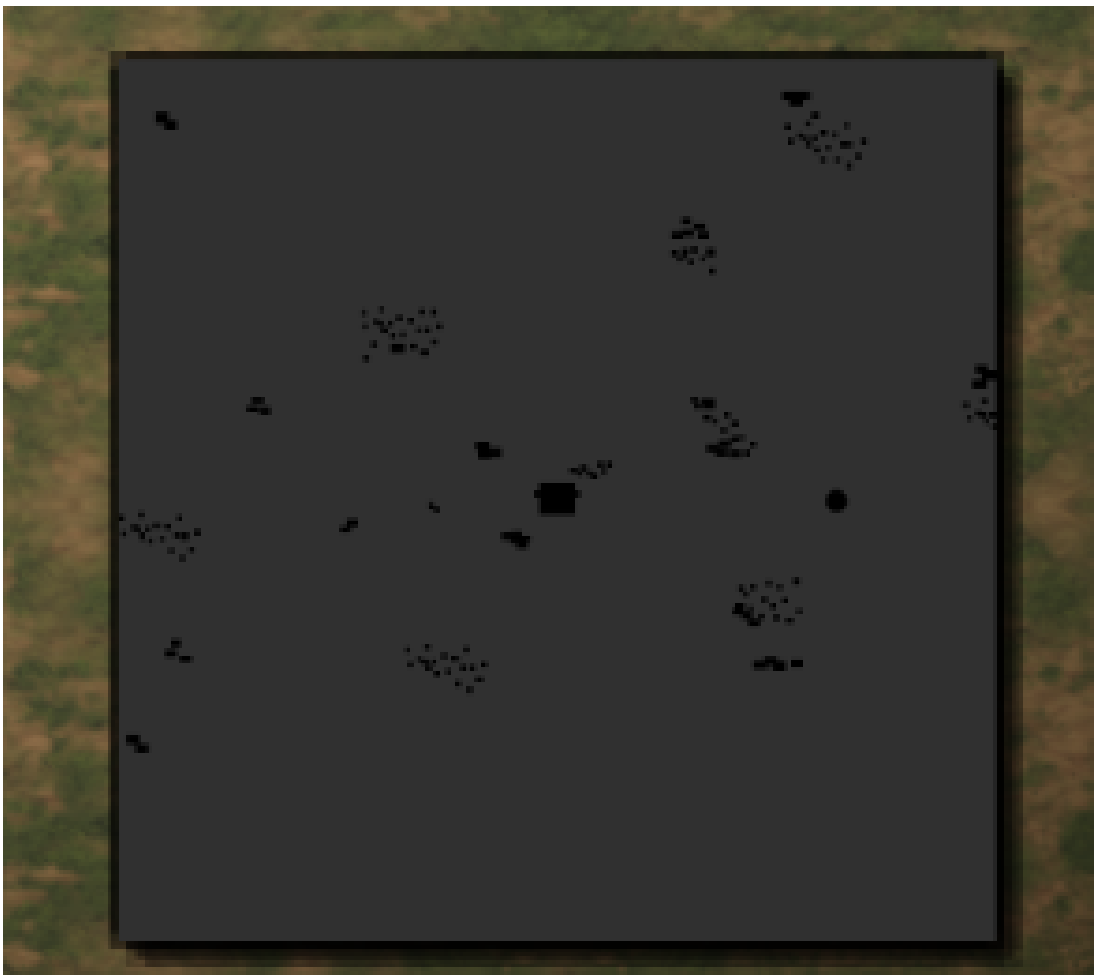


Рис. 4.20 Мінікарта

21. Додано елемент інтерфейсу - головне меню(Рис. 4.21), головне меню є окремою сценою проєкту Unity, головне меню містить кнопку "Play", що переносить гравця на сцену з самою грою, кноку "Quit", що закриває гру та . Створено клас MainMenuUI.

22. Додано звукові ефекти та фонова музика.Додано меню паузи та налаштувань(рис. 4.22), меню містить кнопки налаштування музики, заукових ефектів та Edge Scroling`гу.

Рис. 4.21 Головне меню

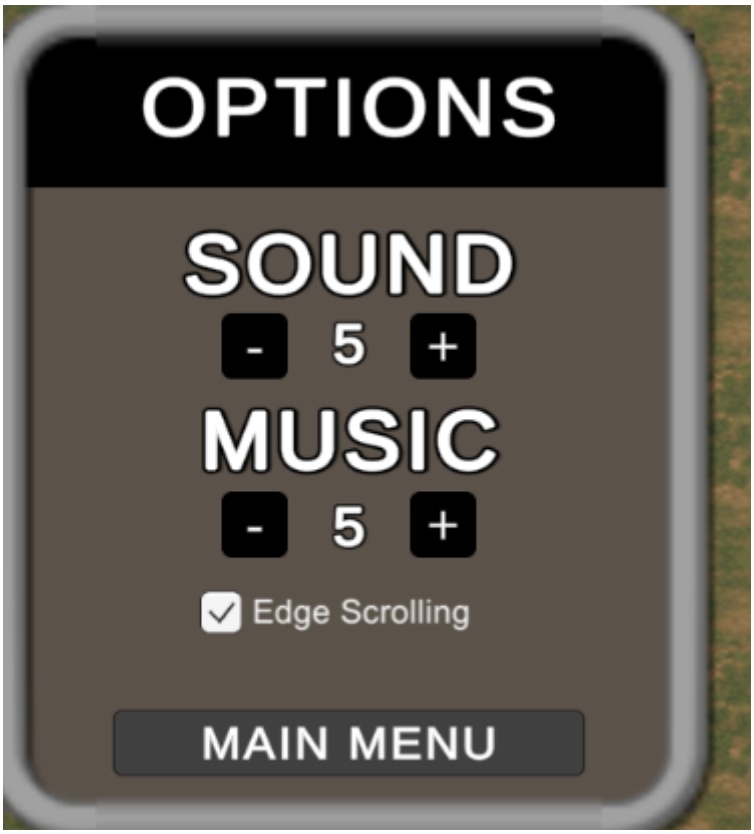


Рис. 4.22 Меню паузи та налаштувань

23. Допрацювання гри:

- Додані анімації готових будівель
- Додані анімації побудови споруд
- Додані анімації переміщення ворогів

- Додані particle ефекти при знищєні ворогів
- Додані декоративні елементи гри - кущі
- Додан цикл дня/ночі, котрий змінюнює градієнт освітлення кожний фіксований проміжок часу.
- Додана кнопка демонтажу, котра зносить готову споруду
- Додана кнопка ремонту, котра ремонтує існуючі споруди.

4.2 Тестування гри

Тестування готової гри - це дуже відповідальний момент розробки гри, оскільки саме воно дає оцінити готовність та функціональність продукту.

4.2.1 Unit тестування

Unit тестування - це тестування окремих класів чи методів в "вакуумі". Такий вид тестування використовується впродовж майже всієї розробки гри, коли додається новий клас або метод.

Прикладами Unit тестування при розробці Defensor TD є:

- Тестування методів додавання ресурсів.
- Тестування взаємодії гравця з інтерфейсом вибори будівлі
- Тестування вспливаюих підказок - текст та умови генерації
- Тестування фунціональності полоси здоров'я тощо.

4.2.2 Системне тестування

Фінальним шагом розробки гри жанру Tower Defence з елементами Real Time Strategy (Таб. 4.1). стало ситстемне тестування гри. Системне тестування - вид тестування програмного забезпечення, котрий виконує перевірку всієї системи системи.

Таблиця 4.1

Системне тестування гри

Крок	Очікуваний результат	Фактичний результат	Статус
У головному меню натиснути кнопку "Play."	Натискання має перенести граця на сцену гри	При натисканні, починається гра	Працює
Початок гри	Гравець має почати гру з Головною спорудою та 30 одиницями дерева	Гравець починає гру з Головною спорудою та 30 одиницями дерева.	Працює
Наведення курсора на одну з кнопкок вибору споруди у лівому нижньому куті	Має відобразтись вартість побудоби відповідної споруди.	Список підказок, що відобразилися для кожної споруди: "Wood Harvester W30", "Stone Harvester W80", "Gold Harvester W200 S200", "Tower 60W 20S 30G"	Працює
Натискання на кнопку вибору споруди	Кнопка обраної споруди має бути оточена білою рамою,	Кнопка обраної споруди оточується білою рамою,	Працює
Маючи обрану споруду, поводити курсором по мапі	Має відоражатись проєкція споруди на потенціальному місці будівництва	Проєкція споруди відоражається на потенціальному місці будівництва	Працює

Продовження таблиці 4.1

Системне тестування гри

Спробувати побудувати споруду поверх іншого ігрового об'єкту	Має відобразитись повідомлення "The aria is not clear"	Відображається повідомлення "The aria is not clear"	Працює
Спробувати побувати споруду, не маючи ресусів	Має відобразитись повідомлення "Cannot afford [building price]"	При спробі побудувати Gold Harvester виходить повідомлення " Cannot afford 200W 200S"	Працює
Спробувати побувати споруду далеко	Має відобразитись повідомлення "Too far from any other building!"	Відображається повідомлення "Too far from any other building!"	Працює
Спобувати побудувати добуваючу споруду поруч ідентичною спорудою	Має відобразитись повідомлення "Too close from a building of the same type!"	Відображається повідомлення "Too close from a building of the same type!"	Працює
Спобувати побудувати добуваючу споруду далеко від родовищ відповідного ресурсу	Має відобразитись повідомлення "No resource nodes nearby!"	Відображається "No resource nodes nearby!"	Працює

Продовження таблиці 4.1

Системне тестування гри

Спобувати побудувати споруду, виконавши всі умови	Мають списатися ресурси, рівні вартості обраної будівлі та початися процес побудови будівлі	При побудові будівлі Tower були списані 60 одинциць дерева, 20 каміння та 30 золота. Почалася анімація будівництва	Працює
Спорбувати відремонтувати пошкоджену споруду не маючи ресурсів	Має відобразитись повідомлення "Cannot afford [repair price]"	При спробі відремонтувати пошкожену вежу відобразитилося повідомлення "Cannot afford 33G"	Працює
Спорбувати відремонтувати пошкоджену споруду маючі ресурси	Мають списатися ресурси, рівні вартості ремонту, кількість одиниць здоров'я має бути рівною максимальному здоров'ю будівлі	При спробі відремонтувати пошкожену вежу було списано 34 одиниці золота, здоров'я будівлі було півністю відновлене	Працює
Спробувувати демонтувати будівлю	Будівля має бути знищена, 50% вартості будівлі має бути додано до ресурсів.	При демонтажі Wood Harvester було додвно 15 одиниць дерева, будівля була знищена	Працює

Продовження таблиці 4.1

Системне тестування гри

Натиснути кнопку паузи у верхньому правому куті	Гра має бути призупинена, має відобразитися меню паузи та налаштувань	Гра призупиняється, відображається меню паузи та налаштувань	Працює
Спочатку збльшити, потім зменшитисягучність музики	Музика має стати спочатку гучнішою потім тихішою	Музика стала спочатку гучнішою потім тихішою	Працює
Спочатку збільшити, потім зменшити гучність звукових ефектів	Звукові ефекти мають стати спочатку гучнішими потім тихішими	Звукові ефекти стали спочатку гучнішими потім тихішими	Працює
Вимкнути опцію Edge Scrolling	Камера має перестати рухатись, коли курсор біля краю єкрана	Камераперестала рухатись, коли курсор біля краю єкрана	Працює
Натиснути кнопку "MAIN MENU"	Гравця має перенести в головне меню	Гра переходить в головне меню	Працює

ВИСНОВКИ

Результатами виконаної роботи стало розширення геймплею жанру Tower Defence завдяки імплементації комбінації нововведень, що поглибшують геймплей, зберігаючи сильні сторони жанру Tower Defence.

Було виконано задачі дипломної роботи:

- Аналіз предметної галузі: було проаналізовано відмінні риси жанру відеоігор Tower Defence, зроблено огляд існуючих аналогів, визначені їхні сильні та слабкі риси; проведена характеристика цільової аудиторії. На основі аналізу предметної галузі створено концепт гри та сформовані вимоги до гри: можливість будівлі економічних та захисних споруд, можливість добування ресурсів за допомогою економічних споруд, можливість ремонтувати існуючі споруд, можливість демонтувати існуючі споруди, можливість знищення ігрових ворогів за допомогою захисних споруд.
- Обрано засоби реалізи реалізації: мову програмування C#, середовище програмування Visual Studio, ігровий рушій Unity, систему контролю версій GitHub.
- Розроблено архітектуру гри за допомогою діаграм прецедентів та класів.
- Реалізовано гру, використвоючі обрані засоби. Розроблена логіка та сформовано інтерфейс користувача на рушії Unity. Гра була фунціонально протестована та завантажена на GitHub.

Результати апробації роботи:

1. Черкасов К.А., Треньов М.Г. Сьогодення та майбутнє ігрового рушія Unity. Матеріали Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. 24 квітня 2025 року, ДУІКТ, м.Київ - С. 238-239
2. Черкасов К.А., Треньов М.Г. Розробка гри жанру tower defence з елементами real time strategy мовою C# Матеріали Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. 24 квітня 2025 року, ДУІКТ, м.Київ - С. 241-242

ПЕРЕЛІК ПОСИЛАНЬ

1. Gill S. How Many Gamers Are There in 2025? Latest Stats. *Priori Data*. 26.01.2025 https://prioridata.com/number-of-gamers/ (date of access: 25.05.2025).
2. Knezovic A. Mobile Strategy Games 2020 Report – Including Ads, Business & Monetization Tactics. *Udonis*. 23.11.2023 URL: https://www.blog.udonis.co/mobile-marketing/mobile-games/strategy-games-report (date of access: 25.05.2025).
3. Dunaway R. The Book of Visual Studio .NET. No Starch Press, 2002. 456 p.
4. Wagner B. Overview - A tour of C#. Microsoft Learn: Build skills that open doors in your career. 21.03. 2025 URL: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/overview (date of access: 25.05.2025).
5. Рейтинг мов програмування 2023. *DOU*. 20.03.2023 URL: https://dou.ua/lenta/articles/language-rating-2023/ (дата звернення: 25.05.2025).
6. What's new in C# 11. *Microsoft Learn: Build skills that open doors in your career*. 03.15.2024 URL: https://learn.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-11#extended-nameof-scope (date of access: 25.05.2025).
7. What is Unity?. *PubNub*. 02.10.2023 URL: https://www.pubnub.com/guides/unity/ (date of access: 25.05.2025).
8. Chacon S., Straub B. Pro Git. Berkeley, CA : Apress, 2014. 441 p.
9. Hall G. M. Adaptive Code: Agile coding with design patterns and SOLID principles. Microsoft Press, 2017. 448 p.
10. Smith R. B. The KISS principle: Approaches to building reliable systems. New York : PBI, 1983. 207 p.
11. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Design Patterns. Elements of Reusable Object-oriented Software. Addsion Wesley Longman, printed in India by Eastern Press, 1999. 417 p.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ(Презентація)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка гри жанру Tower Defence з елементами Real Time Strategy мовою C#

Виконав студент 4 курсу
групи ПД-43
Черкасов Карєн Андрійович
Керівник роботи
асистент кафедри Треньов Микита Георгійович

Київ – 2025

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи - розширення геймплею жанру Tower Defence з елементами Real Time Strategy завдяки імплементації комбінації нововведень, що поглибшують геймплей, зберігаючи сильні сторони жанру Tower Defence.

Об'єкт дослідження - геймплей гри жанру Tower Defence з елементами Real Time Strategy

Предмет дослідження - засоби реалізації гри у жанрі Tower Defence з елементами Real Time Strategy.

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Провести аналіз аналогічних ігор жанру Tower Defence.
2. Визначити та сформувати вимоги до гри жанру Tower Defence з елементами Real Time Strategy.
3. Спроєктувати архітектуру гри жанру Tower Defence з елементами Real Time Strategy.
4. Реалізувати гру жанру Tower Defence з елементами Real Time Strategy.
5. Провести тестування гри жанру Tower Defence з елементами Real Time Strategy.

3

АНАЛІЗ АНАЛОГІВ

Показник	**Plants vs. Zombies**	**Defense Grid: The Awakening**	**Kingdom Rush**	Defesor TD
Платформи	Android, Microsoft Windows	Microsoft Windows, Xbox3 60	Browser game, Android, Microsoft Windows	Microsoft Windows, Android
Наявність більш ніж одного будівельного ресурсу	Ні	Так	Ні	Так
Вільне розміщення споруд на карті	Ні	Ні	Ні	Так
Наявність безкінечного режиму гри	Так	Ні	Так	Так
Наявність ворогів, що атакують будь-які споруди гравця	Так	Ні	Ні	Так
Спосіб добування будівельних ресурсів окрім знищення ворогів	Так	Ні	Ні	Так

4

Концепт Гри

Головна задача гри - відбити максимальну кількість хвиль ігрових ворогів за допомогою споруд. Гравець може будувати споруди, використовуючі ресурси для будівництва: дерево, каміння та золото. Споруди використовуються для добування ресурсів та знищення ворогів.

Починає гру з Головною Спорудою(HQ) та невеликою кількістю ресурсів. Втрата HQ є умовою поразки гравця.

Вороги генеруються хвилями у заздалігдь визначених точках та рухаються по замовчуванню до HQ, атакуючі зустрічні споруди. Кожна наступна хвиля генерує більшу кількість ворогів.

5

ФУНКЦІОНАЛЬНІ ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1. Гравець має мати можливість будувати нові будівлі.
2. Гравець має мати можливість ремонтувати існуючі будівлі.
3. Гравець має мати можливість демонтувати існуючі будівлі.
4. Гравець має мати можливість налаштовувати гучність музики.
5. Гравець має мати можливість налаштовувати гучність ігрових ефектів.
6. Гравець має мати можливість вмикати або вимикати переміщення головної камери торканням курсором краю екрану.

6

НЕФУНКЦІОНАЛЬНІ ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

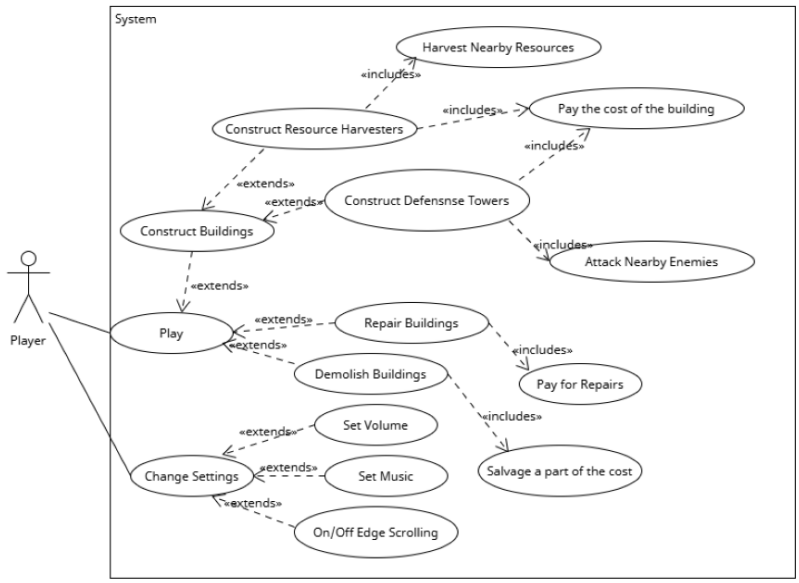
1. Гра має бути доступною на ПК та Android.
2. Гра має поєднувати жанр Tower Defence з елементами Real Time Strategy.
3. Гра має мати музику стилізований під фентезі саундтрек.
4. Гра має мати 2D графіку.
5. Вид на ігровий світ має бути "зверху".
6. Гра має бути помірно складною.

7

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

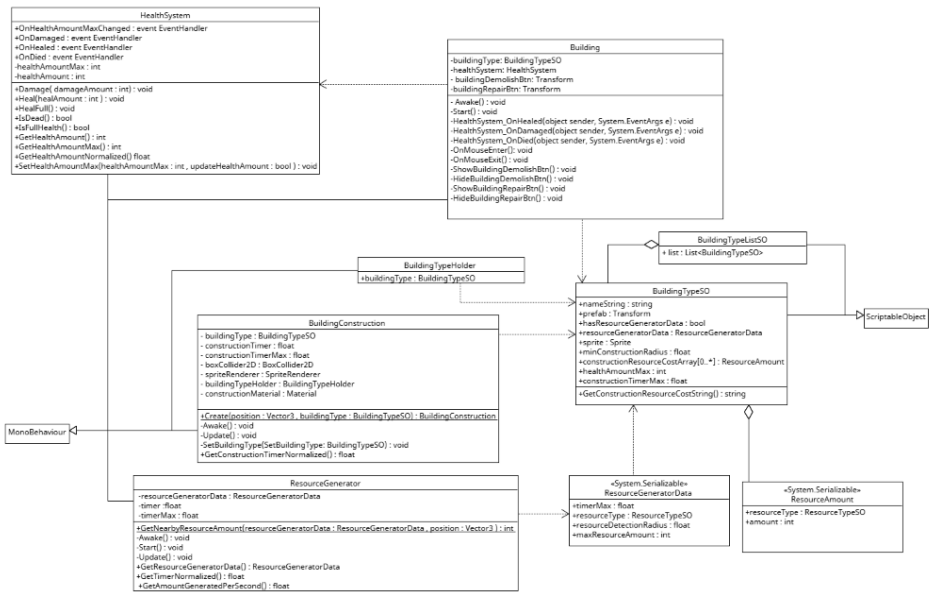
8

ДІАГРАМА ПРЕЦЕДЕНТІВ



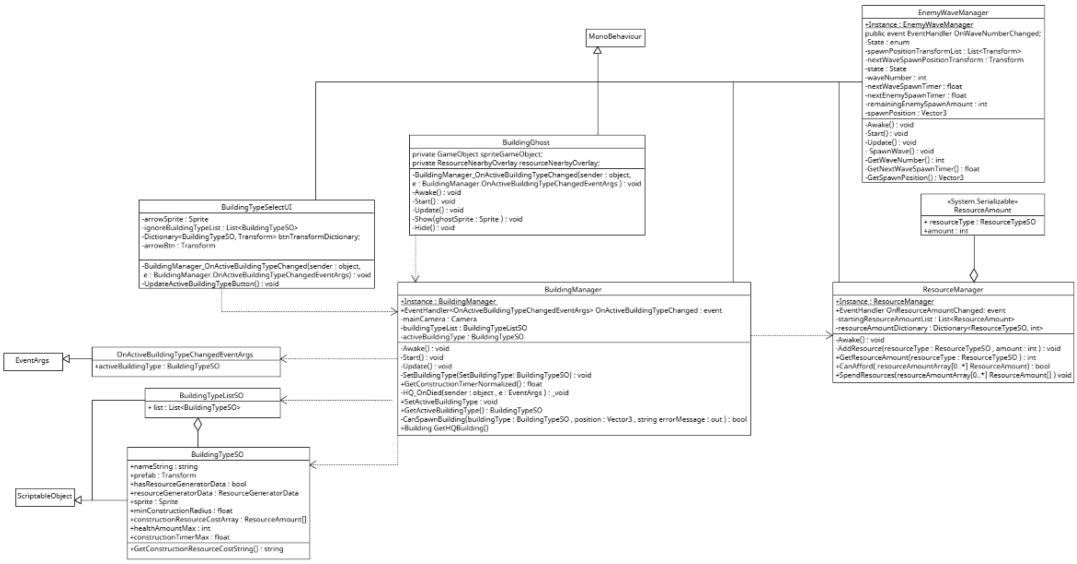
9

ДІАГРАМА КЛАСІВ



10

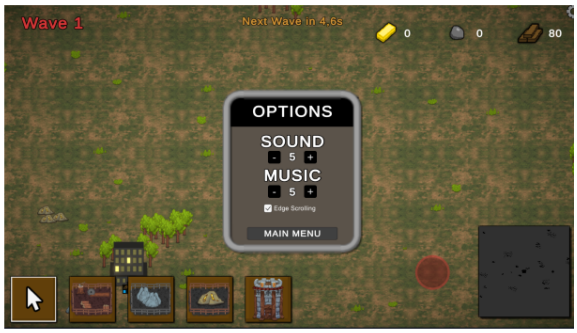
ДІАГРАМА КЛАСІВ - ПРОДОВЖЕННЯ



11

ЕКРАННІ ФОРМИ

Головне меню



Меню Опцій

12

ЕКРАННІ ФОРМИ - ПРОДОВЖЕННЯ

Геймплей

Меню Поразки

Відеодемонстрація

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Черкасов К.А., Треньов М.Г. Сьогодення та майбутнє ігрового рушія Unity. Матеріали Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. 24 квітня 2025 року, ДУІКТ, м.Київ - С. 238-239
2. Черкасов К.А., Треньов М.Г. Розробка гри жанру tower defence з елементами real time strategy мовою C# Матеріали Всеукраїнської науково -технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. 24 квітня 2025 року, ДУІКТ, м.Київ - С. 241-242

ВИСНОВКИ

1. Проведено аналіз аналогічних ігор жанру Tower Defence, в ході якого були виявлені їхні головні переваги, як суміщення простоти з тактичним різноманіттям, та їх недоліки: обмежена взаємодія між ігровими ворогами та вежами та обмеження розміщеня споруд на мапі.
2. Визначено та сформовано функціональні та нефункціональні вимоги до гри: гра має доповнювати сильні сторони Tower Defence глибиною Real Time Strategy, гравець має мати можливість будувати, ремонтувати та демонтувати ігрові споруди.

ВИСНОВКИ - ПРОДОВЖЕННЯ

3. Спроєктовано архітектуру гри жанру Tower Defence з елементами Real Time Strategy. Основою архітектури є принципи SOLID та KISS. Для створення класів гри були використані шаблони проєктування Singleton, Observer, Factory.
4. Гра реалізована за допомогою засобів ігровий рушій Unity, середовище програмування Visual Studio, мова програмування C#.
5. Проведено функціональне тестування та гри, перевірені на справність: можливості гравця - побудова, ремонт, демонтаж споруд, зміна опцій; механіки гри - добування ресурсів, генерація та поведінка ворогів, списання ресурсів при побудові споруди, взаємодія між спорудами та ворогами; Інтерфейс користувача - відображення кількості ресурсів, підказки, індикатори та таймери хвиль.

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

u

```
sing UnityEngine;

public class ArrowProjectile : MonoBehaviour {

   public static ArrowProjectile Create(Vector3 position,
Enemy enemy) {
      Transform arrowTransform =
Instantiate(GameAssets.Instance.pfArrowProjectile,
position, Quaternion.identity);

      ArrowProjectile arrowProjectile =
arrowTransform.GetComponent<ArrowProjectile>();
      arrowProjectile.SetTarget(enemy);

      return arrowProjectile;
   }

   private Enemy targetEnemy;
   private Vector3 lastMoveDir;
   private float timeToDie = 2f;

   private void Update() {
      Vector3 moveDir;

      if (targetEnemy != null) {
         moveDir = (targetEnemy.transform.position -
transform.position).normalized;
         lastMoveDir = moveDir;
      } else {
         moveDir = lastMoveDir;
      }

      float moveSpeed = 20f;
      transform.position += moveDir * moveSpeed *
Time.deltaTime;

      transform.eulerAngles = new Vector3(0, 0,
UtilsClass.GetAngleFromVector(moveDir));

      timeToDie -= Time.deltaTime;
      if (timeToDie < 0f) {
         Destroy(gameObject);
      }
   }

   private void SetTarget(Enemy targetEnemy) {
      this.targetEnemy = targetEnemy;
   }

   private void OnTriggerEnter2D(Collider2D collision)
{
      Enemy enemy =
collision.GetComponent<Enemy>();
      if (enemy != null) {
         // Hit an enemy!
         int damageAmount = 10;
enemy.GetComponent<HealthSystem>().Damage(dama
geAmount);

         Destroy(gameObject);
      }
   }

}

using UnityEngine;

public class Building : MonoBehaviour {

   private BuildingTypeSO buildingType;
   private HealthSystem healthSystem;
   private Transform buildingDemolishBtn;
   private Transform buildingRepairBtn;

   private void Awake() {
      buildingDemolishBtn =
transform.Find("pfBuildingDemolishBtn");
      buildingRepairBtn =
transform.Find("pfBuildingRepairBtn");

      HideBuildingDemolishBtn();
      HideBuildingRepairBtn();
   }

   private void Start() {
      buildingType =
GetComponent<BuildingTypeHolder>().buildingType;

      healthSystem = GetComponent<HealthSystem>();

healthSystem.SetHealthAmountMax(buildingType.heal
thAmountMax, true);
      healthSystem.OnDamaged +=
HealthSystem_OnDamaged;
      healthSystem.OnHealed +=
HealthSystem_OnHealed;

      healthSystem.OnDied += HealthSystem_OnDied;
   }

   private void HealthSystem_OnHealed(object sender,
System.EventArgs e) {
      if (healthSystem.IsFullHealth()) {
         HideBuildingRepairBtn();
      }
   }

   private void HealthSystem_OnDamaged(object
sender, System.EventArgs e) {
      ShowBuildingRepairBtn();
SoundManager.Instance.PlaySound(SoundManager.Sou
nd.BuildingDamaged);
CinemachineShake.Instance.ShakeCamera(7f, .15f);
ChromaticAberrationEffect.Instance.SetWeight(1f);
   }

   private void HealthSystem_OnDied(object sender,
System.EventArgs e) {
Instantiate(GameAssets.Instance.pfBuildingDestroyedP
articles, transform.position, Quaternion.identity);
```

```
        Destroy(gameObject);
SoundManager.Instance.PlaySound(SoundManager.Sou
nd.BuildingDestroyed);
CinemachineShake.Instance.ShakeCamera(10f, .2f);
ChromaticAberrationEffect.Instance.SetWeight(1f);
    }

    private void OnMouseEnter() {
        ShowBuildingDemolishBtn();
    }

    private void OnMouseExit() {
        HideBuildingDemolishBtn();
    }

    private void ShowBuildingDemolishBtn() {
        if (buildingDemolishBtn != null) {
buildingDemolishBtn.gameObject.SetActive(true);
        }
    }

    private void HideBuildingDemolishBtn() {
        if (buildingDemolishBtn != null) {
buildingDemolishBtn.gameObject.SetActive(false);
        }
    }

    private void ShowBuildingRepairBtn() {
        if (buildingRepairBtn != null) {
            buildingRepairBtn.gameObject.SetActive(true);
        }
    }

    private void HideBuildingRepairBtn() {
        if (buildingRepairBtn != null) {
            buildingRepairBtn.gameObject.SetActive(false);
        }
    }

}

using UnityEngine;

public class BuildingConstruction : MonoBehaviour {

    public static BuildingConstruction Create(Vector3
position, BuildingTypeSO buildingType) {
        Transform buildingConstructionTransform =
Instantiate(GameAssets.Instance.pfBuildingConstructio
n, position, Quaternion.identity);

        BuildingConstruction buildingConstruction =
buildingConstructionTransform.GetComponent<Buildi
ngConstruction>();
buildingConstruction.SetBuildingType(buildingType);

        return buildingConstruction;
    }



    private BuildingTypeSO buildingType;
    private float constructionTimer;
    private float constructionTimerMax;
    private BoxCollider2D boxCollider2D;
    private SpriteRenderer spriteRenderer;
    private BuildingTypeHolder buildingTypeHolder;
    private Material constructionMaterial;

    private void Awake() {
        boxCollider2D =
GetComponent<BoxCollider2D>();
        spriteRenderer =
transform.Find("sprite").GetComponent<SpriteRendere
r>();
        buildingTypeHolder =
GetComponent<BuildingTypeHolder>();
        constructionMaterial = spriteRenderer.material;

Instantiate(GameAssets.Instance.pfBuildingPlacedParti
cles, transform.position, Quaternion.identity);
    }

    private void Update() {
        constructionTimer -= Time.deltaTime;

        constructionMaterial.SetFloat("_Progress",
GetConstructionTimerNormalized());

        if (constructionTimer <= 0f) {
            Instantiate(buildingType.prefab,
transform.position, Quaternion.identity);
Instantiate(GameAssets.Instance.pfBuildingPlacedParti
cles, transform.position, Quaternion.identity);
SoundManager.Instance.PlaySound(SoundManager.Sou
nd.BuildingPlaced);
            Destroy(gameObject);
        }
    }

    private void SetBuildingType(BuildingTypeSO
buildingType) {
        this.buildingType = buildingType;

        constructionTimerMax =
buildingType.constructionTimerMax;
        constructionTimer = constructionTimerMax;

        spriteRenderer.sprite = buildingType.sprite;

        boxCollider2D.offset =
buildingType.prefab.GetComponent<BoxCollider2D>
().offset;
        boxCollider2D.size =
buildingType.prefab.GetComponent<BoxCollider2D>
().size;

        buildingTypeHolder.buildingType = buildingType;
    }

    public float GetConstructionTimerNormalized() {
        return 1 - constructionTimer /
constructionTimerMax;
    }

}

using UnityEngine;
using UnityEngine.UI;

public class BuildingDemolishBtn : MonoBehaviour {

    [SerializeField] private Building building;

    private void Awake() {
transform.Find("button").GetComponent<Button>().on
```

```
Click.AddListener(() => {
            BuildingTypeSO buildingType =
building.GetComponent<BuildingTypeHolder>().buildi
ngType;
            foreach (ResourceAmount resourceAmount in
buildingType.constructionResourceCostArray) {
ResourceManager.Instance.AddResource(resourceAmo
unt.resourceType,
Mathf.FloorToInt(resourceAmount.amount * .6f));
            }
            Destroy(building.gameObject);
        });
    }
}


using UnityEngine;

public class BuildingGhost : MonoBehaviour {

    private GameObject spriteGameObject;
    private ResourceNearbyOverlay
resourceNearbyOverlay;

    private void Awake() {
        spriteGameObject =
transform.Find("sprite").gameObject;
        resourceNearbyOverlay =
transform.Find("pfResourceNearbyOverlay").GetComp
onent<ResourceNearbyOverlay>();

        Hide();
    }

    private void Start() {
BuildingManager.Instance.OnActiveBuildingTypeChan
ged +=
BuildingManager_OnActiveBuildingTypeChanged;
    }

    private void
BuildingManager_OnActiveBuildingTypeChanged(obj
ect sender,
BuildingManager.OnActiveBuildingTypeChangedEven
tArgs e) {
        if (e.activeBuildingType == null) {
            Hide();
            resourceNearbyOverlay.Hide();
        } else {
            Show(e.activeBuildingType.sprite);
            if
(e.activeBuildingType.hasResourceGeneratorData) {
resourceNearbyOverlay.Show(e.activeBuildingType.res
ourceGeneratorData);
            } else {
                resourceNearbyOverlay.Hide();
            }
        }
    }

    private void Update() {
        transform.position =
UtilsClass.GetMouseWorldPosition();
    }

    private void Show(Sprite ghostSprite) {
        spriteGameObject.SetActive(true);
spriteGameObject.GetComponent<SpriteRenderer>().s
prite = ghostSprite;
    }

    private void Hide() {
        spriteGameObject.SetActive(false);
    }


}

using System;
using UnityEngine;
using UnityEngine.EventSystems;

public class BuildingManager : MonoBehaviour {

    public static BuildingManager Instance { get; private
set; }


    public event
EventHandler<OnActiveBuildingTypeChangedEventAr
gs> OnActiveBuildingTypeChanged;

    public class
OnActiveBuildingTypeChangedEventArgs : EventArgs
{
        public BuildingTypeSO activeBuildingType;
    }


    [SerializeField] private Building hqBuilding;

    private Camera mainCamera;
    private BuildingTypeListSO buildingTypeList;
    private BuildingTypeSO activeBuildingType;

    private void Awake() {
        Instance = this;

        buildingTypeList =
Resources.Load<BuildingTypeListSO>(typeof(Buildin
gTypeListSO).Name);
    }

    private void Start() {
        mainCamera = Camera.main;

hqBuilding.GetComponent<HealthSystem>().OnDied
+= HQ_OnDied;
    }

    private void HQ_OnDied(object sender, EventArgs e)
{
SoundManager.Instance.PlaySound(SoundManager.Sou
nd.GameOver);
        GameOverUI.Instance.Show();
    }

    private void Update() {
        if (Input.GetMouseButtonDown(0)
&& !EventSystem.current.IsPointerOverGameObject())
{
            if (activeBuildingType != null) {
                if (CanSpawnBuilding(activeBuildingType,
UtilsClass.GetMouseWorldPosition(), out string
errorMessage)) {
                    if
(ResourceManager.Instance.CanAfford(activeBuilding
Type.constructionResourceCostArray)) {
```

```
ResourceManager.Instance.SpendResources(activeBuildingType.constructionResourceCostArray);
                //Instantiate(activeBuildingType.prefab, UtilsClass.GetMouseWorldPosition(), Quaternion.identity);
                BuildingConstruction.Create(UtilsClass.GetMouseWorldPosition(), activeBuildingType);
                SoundManager.Instance.PlaySound(SoundManager.Sound.BuildingPlaced);
            } else {
                TooltipUI.Instance.Show("Cannot afford " + activeBuildingType.GetConstructionResourceCostString(),
                    new TooltipUI.TooltipTimer { timer = 2f });
            }
        } else {
            TooltipUI.Instance.Show(errorMessage, new TooltipUI.TooltipTimer { timer = 2f });
        }
    }
  }

  public void SetActiveBuildingType(BuildingTypeSO buildingType) {
    activeBuildingType = buildingType;

    OnActiveBuildingTypeChanged?.Invoke(this,
      new OnActiveBuildingTypeChangedEventArgs { activeBuildingType = activeBuildingType }
    );
  }

  public BuildingTypeSO GetActiveBuildingType() {
    return activeBuildingType;
  }

  private bool CanSpawnBuilding(BuildingTypeSO buildingType, Vector3 position, out string errorMessage) {
    BoxCollider2D boxCollider2D = buildingType.prefab.GetComponent<BoxCollider2D>();

    Collider2D[] collider2DArray = Physics2D.OverlapBoxAll(position + (Vector3)boxCollider2D.offset, boxCollider2D.size, 0);

    bool isAreaClear = collider2DArray.Length == 0;
    if (!isAreaClear) {
      errorMessage = "Area is not clear!";
      return false;
    }


    collider2DArray = Physics2D.OverlapCircleAll(position, buildingType.minConstructionRadius);

    foreach (Collider2D collider2D in collider2DArray) {
      BuildingTypeHolder buildingTypeHolder = collider2D.GetComponent<BuildingTypeHolder>();
      if (buildingTypeHolder != null) {
        if (buildingTypeHolder.buildingType == buildingType) {
          errorMessage = "Too close to another building of the same type!";
          return false;
        }
      }
    }

    if (buildingType.hasResourceGeneratorData) {
      ResourceGeneratorData resourceGeneratorData = buildingType.resourceGeneratorData;
      int nearbyResourceAmount = ResourceGenerator.GetNearbyResourceAmount(resourceGeneratorData, position);

      if (nearbyResourceAmount == 0) {
        errorMessage = "There are no nearby Resource Nodes!";
        return false;
      }
    }


    float maxConstructionRadius = 25;
    collider2DArray = Physics2D.OverlapCircleAll(position, maxConstructionRadius);

    foreach (Collider2D collider2D in collider2DArray) {
      BuildingTypeHolder buildingTypeHolder = collider2D.GetComponent<BuildingTypeHolder>();
      if (buildingTypeHolder != null) {
        errorMessage = "";
        return true;
      }
    }

    errorMessage = "Too far from any other building!";
    return false;
  }

  public Building GetHQBuilding() {
    return hqBuilding;
  }

}


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

public class BuildingTypeSelectUI : MonoBehaviour {

  [SerializeField] private Sprite arrowSprite;
  [SerializeField] private List<BuildingTypeSO> ignoreBuildingTypeList;

  private Dictionary<BuildingTypeSO, Transform> btnTransformDictionary;
  private Transform arrowBtn;

  private void Awake() {
    Transform btnTemplate = transform.Find("btnTemplate");
    btnTemplate.gameObject.SetActive(false);

    BuildingTypeListSO buildingTypeList = Resources.Load<BuildingTypeListSO>(typeof(Buildin
```

```
gTypeListSO).Name);

        btnTransformDictionary = new
Dictionary<BuildingTypeSO, Transform>();

        int index = 0;

        arrowBtn = Instantiate(btnTemplate, transform);
        arrowBtn.gameObject.SetActive(true);

        float offsetAmount = +130f;
arrowBtn.GetComponent<RectTransform>().anchored
Position = new Vector2(offsetAmount * index, 0);

arrowBtn.Find("image").GetComponent<Image>().spri
te = arrowSprite;
arrowBtn.Find("image").GetComponent<RectTransfor
m>().sizeDelta = new Vector2(0, -40);

arrowBtn.GetComponent<Button>().onClick.AddListe
ner(() => {
BuildingManager.Instance.SetActiveBuildingType(nul
l);
        });

        MouseEnterExitEvents mouseEnterExitEvents =
arrowBtn.GetComponent<MouseEnterExitEvents>();
        mouseEnterExitEvents.OnMouseEnter += (object
sender, EventArgs e) => {
            TooltipUI.Instance.Show("Arrow");
        };
        mouseEnterExitEvents.OnMouseExit += (object
sender, EventArgs e) => {
            TooltipUI.Instance.Hide();
        };

        index++;


        foreach (BuildingTypeSO buildingType in
buildingTypeList.list) {
            if
(ignoreBuildingTypeList.Contains(buildingType))
continue;
            Transform btnTransform =
Instantiate(btnTemplate, transform);
            btnTransform.gameObject.SetActive(true);

            offsetAmount = +130f;
btnTransform.GetComponent<RectTransform>().ancho
redPosition = new Vector2(offsetAmount * index, 0);

btnTransform.Find("image").GetComponent<Image>().
sprite = buildingType.sprite;

btnTransform.GetComponent<Button>().onClick.AddL
istener(() => {
BuildingManager.Instance.SetActiveBuildingType(buil
dingType);
            });

            mouseEnterExitEvents =
btnTransform.GetComponent<MouseEnterExitEvents>
();
            mouseEnterExitEvents.OnMouseEnter +=
(object sender, EventArgs e) => {
TooltipUI.Instance.Show(buildingType.nameString +
"\n" +
buildingType.GetConstructionResourceCostString());
            };
            mouseEnterExitEvents.OnMouseExit +=
(object sender, EventArgs e) => {
                TooltipUI.Instance.Hide();
            };

            btnTransformDictionary[buildingType] =
btnTransform;

            index++;
        }
    }

    private void Start() {
BuildingManager.Instance.OnActiveBuildingTypeChan
ged +=
BuildingManager_OnActiveBuildingTypeChanged;
        UpdateActiveBuildingTypeButton();
    }

    private void
BuildingManager_OnActiveBuildingTypeChanged(obj
ect sender,
BuildingManager.OnActiveBuildingTypeChangedEven
tArgs e) {
        UpdateActiveBuildingTypeButton();
    }

    private void UpdateActiveBuildingTypeButton() {
arrowBtn.Find("selected").gameObject.SetActive(false);
        foreach (BuildingTypeSO buildingType in
btnTransformDictionary.Keys) {
            Transform btnTransform =
btnTransformDictionary[buildingType];
btnTransform.Find("selected").gameObject.SetActive(f
alse);
        }

        BuildingTypeSO activeBuildingType =
BuildingManager.Instance.GetActiveBuildingType();
        if (activeBuildingType == null) {
arrowBtn.Find("selected").gameObject.SetActive(true);
        } else {
btnTransformDictionary[activeBuildingType].Find("sel
ected").gameObject.SetActive(true);
        }
    }

}

using UnityEngine;

[CreateAssetMenu(menuName =
"ScriptableObjects/BuildingType")]
public class BuildingTypeSO : ScriptableObject {

    public string nameString;
    public Transform prefab;
    public bool hasResourceGeneratorData;
    public ResourceGeneratorData
resourceGeneratorData;
    public Sprite sprite;
    public float minConstructionRadius;
    public ResourceAmount[]
constructionResourceCostArray;
    public int healthAmountMax;
    public float constructionTimerMax;
```

```
    public string GetConstructionResourceCostString() {
        string str = "";
        foreach (ResourceAmount resourceAmount in
constructionResourceCostArray) {
            str += "<color=#" +
resourceAmount.resourceType.colorHex + ">" +
                resourceAmount.resourceType.nameShort +
resourceAmount.amount +
                "</color> ";
        }
        return str;
    }

}


using UnityEngine;

public class Enemy : MonoBehaviour {

    public static Enemy Create(Vector3 position) {
        Transform enemyTransform =
Instantiate(GameAssets.Instance.pfEnemy, position,
Quaternion.identity);

        Enemy enemy =
enemyTransform.GetComponent<Enemy>();
        return enemy;
    }



    private Rigidbody2D rigidbody2d;
    private Transform targetTransform;
    private HealthSystem healthSystem;
    private float lookForTargetTimer;
    private float lookForTargetTimerMax = .2f;

    private void Start() {
        rigidbody2d = GetComponent<Rigidbody2D>();
        if (BuildingManager.Instance.GetHQBuilding() !=
null) {
            targetTransform =
BuildingManager.Instance.GetHQBuilding().transform;
        }
        healthSystem = GetComponent<HealthSystem>();
        healthSystem.OnDamaged +=
HealthSystem_OnDamaged;
        healthSystem.OnDied += HealthSystem_OnDied;

        lookForTargetTimer = Random.Range(0f,
lookForTargetTimerMax);
    }

    private void HealthSystem_OnDamaged(object
sender, System.EventArgs e) {
SoundManager.Instance.PlaySound(SoundManager.Sou
nd.EnemyHit);
        CinemachineShake.Instance.ShakeCamera(2f, .1f);
ChromaticAberrationEffect.Instance.SetWeight(.5f);
    }

    private void HealthSystem_OnDied(object sender,
System.EventArgs e) {
SoundManager.Instance.PlaySound(SoundManager.Sou
nd.EnemyDie);
CinemachineShake.Instance.ShakeCamera(5f, .15f);
ChromaticAberrationEffect.Instance.SetWeight(.5f);
Instantiate(GameAssets.Instance.pfEnemyDieParticles,
transform.position, Quaternion.identity);
        Destroy(gameObject);
    }

    private void Update() {
        HandleMovement();
        HandleTargeting();
    }

    private void OnCollisionEnter2D(Collision2D
collision) {
        Building building =
collision.gameObject.GetComponent<Building>();
        if (building != null) {
            HealthSystem healthSystem =
building.GetComponent<HealthSystem>();
            healthSystem.Damage(10);
            this.healthSystem.Damage(999);
        }
    }

    private void HandleMovement() {
        if (targetTransform != null) {
            Vector3 moveDir = (targetTransform.position -
transform.position).normalized;

            float moveSpeed = 6f;
            rigidbody2d.velocity = moveDir * moveSpeed;
        } else {
            rigidbody2d.velocity = Vector2.zero;
        }
    }

    private void HandleTargeting() {
        lookForTargetTimer -= Time.deltaTime;
        if (lookForTargetTimer < 0f) {
            lookForTargetTimer +=
lookForTargetTimerMax;
            LookForTargets();
        }
    }



    private void LookForTargets() {
        float targetMaxRadius = 10f;
        Collider2D[] collider2DArray =
Physics2D.OverlapCircleAll(transform.position,
targetMaxRadius);

        foreach (Collider2D collider2D in
collider2DArray) {
            Building building =
collider2D.GetComponent<Building>();
            if (building != null) {
                if (targetTransform == null) {
                    targetTransform = building.transform;
                } else {
                    if (Vector3.Distance(transform.position,
building.transform.position) <
                        Vector3.Distance(transform.position,
targetTransform.position)) {
                        // Closer!
                        targetTransform = building.transform;
                    }
                }
            }
        }

        if (targetTransform == null) {
```

```
            if
(BuildingManager.Instance.GetHQBuilding() != null) {
                targetTransform =
BuildingManager.Instance.GetHQBuilding().transform;
            }
        }
    }


}

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

public class EnemyWaveManager : MonoBehaviour {


    public static EnemyWaveManager Instance { get;
private set; }



    public event EventHandler OnWaveNumberChanged;

    private enum State {
        WaitingToSpawnNextWave,
        SpawningWave,
    }

    [SerializeField] private List<Transform>
spawnPositionTransformList;
    [SerializeField] private Transform
nextWaveSpawnPositionTransform;

    private State state;
    private int waveNumber;
    private float nextWaveSpawnTimer;
    private float nextEnemySpawnTimer;
    private int remainingEnemySpawnAmount;
    private Vector3 spawnPosition;

    private void Awake() {
        Instance = this;
    }

    private void Start() {
        state = State.WaitingToSpawnNextWave;
        spawnPosition =
spawnPositionTransformList[UnityEngine.Random.Ra
nge(0, spawnPositionTransformList.Count)].position;
        nextWaveSpawnPositionTransform.position =
spawnPosition;
        nextWaveSpawnTimer = 10f;
    }

    private void Update() {
        switch (state) {
            case State.WaitingToSpawnNextWave:
                nextWaveSpawnTimer -= Time.deltaTime;
                if (nextWaveSpawnTimer < 0f) {
                    SpawnWave();
                }
                break;
            case State.SpawningWave:
                if (remainingEnemySpawnAmount > 0) {
                    nextEnemySpawnTimer -=
Time.deltaTime;
                    if (nextEnemySpawnTimer < 0f) {
                        nextEnemySpawnTimer =
UnityEngine.Random.Range(0f, .2f);
                        Enemy.Create(spawnPosition +
UtilsClass.GetRandomDir() *
UnityEngine.Random.Range(0f, 10f));
                        remainingEnemySpawnAmount--;

                        if (remainingEnemySpawnAmount <= 0)
{
                            state =
State.WaitingToSpawnNextWave;
                            spawnPosition =
spawnPositionTransformList[UnityEngine.Random.Ra
nge(0, spawnPositionTransformList.Count)].position;
nextWaveSpawnPositionTransform.position =
spawnPosition;
                            nextWaveSpawnTimer =
Mathf.Clamp(30f - 4f * waveNumber, 10f, 30f);
                        }
                    }
                }
                break;
        }
    }
    private void SpawnWave() {
        remainingEnemySpawnAmount = 4 + 3 *
waveNumber;
        state = State.SpawningWave;
        waveNumber++;
        OnWaveNumberChanged?.Invoke(this,
EventArgs.Empty);
    }

    public int GetWaveNumber() {
        return waveNumber;
    }

    public float GetNextWaveSpawnTimer() {
        return nextWaveSpawnTimer;
    }

    public Vector3 GetSpawnPosition() {
        return spawnPosition;
    }

}
using UnityEngine;
using TMPro;

public class EnemyWaveUI : MonoBehaviour {

    [SerializeField] private EnemyWaveManager
enemyWaveManager;

    private Camera mainCamera;
    private TextMeshProUGUI waveNumberText;
    private TextMeshProUGUI waveMessageText;
    private RectTransform
enemyWaveSpawnPositionIndicator;
    private RectTransform
enemyClosestPositionIndicator;

    private void Awake() {
        waveNumberText =
transform.Find("waveNumberText").GetComponent<T
extMeshProUGUI>();
        waveMessageText =
transform.Find("waveMessageText").GetComponent<T
extMeshProUGUI>();
```

```
        enemyWaveSpawnPositionIndicator =
transform.Find("enemyWaveSpawnPositionIndicator").
GetComponent<RectTransform>();
        enemyClosestPositionIndicator =
transform.Find("enemyClosestPositionIndicator").GetC
omponent<RectTransform>();
    }

    private void Start() {
        mainCamera = Camera.main;
        enemyWaveManager.OnWaveNumberChanged +=
EnemyWaveManager_OnWaveNumberChanged;
        SetWaveNumberText("Wave " +
enemyWaveManager.GetWaveNumber());
    }

    private void
EnemyWaveManager_OnWaveNumberChanged(object
sender, System.EventArgs e) {
        SetWaveNumberText("Wave " +
enemyWaveManager.GetWaveNumber());
    }

    private void Update() {
        HandleNextWaveMessage();
        HandleEnemyWaveSpawnPositionIndicator();
        HandleEnemyClosestPositionIndicator();
    }

    private void HandleNextWaveMessage() {
        float nextWaveSpawnTimer =
enemyWaveManager.GetNextWaveSpawnTimer();
        if (nextWaveSpawnTimer <= 0f) {
            SetMessageText("");
        } else {
            SetMessageText("Next Wave in " +
nextWaveSpawnTimer.ToString("F1") + "s");
        }
    }

    private void
HandleEnemyWaveSpawnPositionIndicator() {
        Vector3 dirToNextSpawnPosition =
(enemyWaveManager.GetSpawnPosition() -
mainCamera.transform.position).normalized;

enemyWaveSpawnPositionIndicator.anchoredPosition
= dirToNextSpawnPosition * 300f;
        enemyWaveSpawnPositionIndicator.eulerAngles =
new Vector3(0, 0,
UtilsClass.GetAngleFromVector(dirToNextSpawnPosit
ion));

        float distanceToNextSpawnPosition =
Vector3.Distance(enemyWaveManager.GetSpawnPositi
on(), mainCamera.transform.position);
enemyWaveSpawnPositionIndicator.gameObject.SetAc
tive(distanceToNextSpawnPosition >
mainCamera.orthographicSize * 1.5f);
    }

    private void HandleEnemyClosestPositionIndicator()
{
        float targetMaxRadius = 9999f;
        Collider2D[] collider2DArray =
Physics2D.OverlapCircleAll(mainCamera.transform.po
sition, targetMaxRadius);

        Enemy targetEnemy = null;

        foreach (Collider2D collider2D in
collider2DArray) {
            Enemy enemy =
collider2D.GetComponent<Enemy>();
            if (enemy != null) {
                if (targetEnemy == null) {
                    targetEnemy = enemy;
                } else {
                    if (Vector3.Distance(transform.position,
enemy.transform.position) <
                        Vector3.Distance(transform.position,
targetEnemy.transform.position)) {
                        // Closer!
                        targetEnemy = enemy;
                    }
                }
            }
        }

        if (targetEnemy != null) {
            Vector3 dirToClosestEnemy =
(targetEnemy.transform.position -
mainCamera.transform.position).normalized;

enemyClosestPositionIndicator.anchoredPosition =
dirToClosestEnemy * 250f;
            enemyClosestPositionIndicator.eulerAngles =
new Vector3(0, 0,
UtilsClass.GetAngleFromVector(dirToClosestEnemy));

            float distanceToClosestEnemy =
Vector3.Distance(targetEnemy.transform.position,
mainCamera.transform.position);
enemyClosestPositionIndicator.gameObject.SetActive
(distanceToClosestEnemy >
mainCamera.orthographicSize * 1.5f);
        } else {
enemyClosestPositionIndicator.gameObject.SetActive(f
alse);
        }
    }


    private void SetMessageText(string message) {
        waveMessageText.SetText(message);
    }

    private void SetWaveNumberText(string text) {
        waveNumberText.SetText(text);
    }

}


using System;
using UnityEngine;

public class HealthSystem : MonoBehaviour {

    public event EventHandler
OnHealthAmountMaxChanged;
    public event EventHandler OnDamaged;
    public event EventHandler OnHealed;
    public event EventHandler OnDied;

    [SerializeField] private int healthAmountMax;
    private int healthAmount;
```

```
    private void Awake() {
        healthAmount = healthAmountMax;
    }

    public void Damage(int damageAmount) {
        healthAmount -= damageAmount;
        healthAmount = Mathf.Clamp(healthAmount, 0,
healthAmountMax);

        OnDamaged?.Invoke(this, EventArgs.Empty);

        if (IsDead()) {
            OnDied?.Invoke(this, EventArgs.Empty);
        }
    }

    public void Heal(int healAmount) {
        healthAmount += healAmount;
        healthAmount = Mathf.Clamp(healthAmount, 0,
healthAmountMax);

        OnHealed?.Invoke(this, EventArgs.Empty);
    }

    public void HealFull() {
        healthAmount = healthAmountMax;

        OnHealed?.Invoke(this, EventArgs.Empty);
    }

    public bool IsDead() {
        return healthAmount == 0;
    }

    public bool IsFullHealth() {
        return healthAmount == healthAmountMax;
    }

    public int GetHealthAmount() {
        return healthAmount;
    }

    public int GetHealthAmountMax() {
        return healthAmountMax;
    }

    public float GetHealthAmountNormalized() {
        return (float)healthAmount / healthAmountMax;
    }

    public void SetHealthAmountMax(int
healthAmountMax, bool updateHealthAmount) {
        this.healthAmountMax = healthAmountMax;

        if (updateHealthAmount) {
            healthAmount = healthAmountMax;
        }

        OnHealthAmountMaxChanged?.Invoke(this,
EventArgs.Empty);
    }

}
```

```
using UnityEngine;

public class ResourceGenerator : MonoBehaviour {

    public static int
GetNearbyResourceAmount(ResourceGeneratorData
resourceGeneratorData, Vector3 position) {
        Collider2D[] collider2DArray =
Physics2D.OverlapCircleAll(position,
resourceGeneratorData.resourceDetectionRadius);

        int nearbyResourceAmount = 0;
        foreach (Collider2D collider2D in
collider2DArray) {
            ResourceNode resourceNode =
collider2D.GetComponent<ResourceNode>();
            if (resourceNode != null) {
                if (resourceNode.resourceType ==
resourceGeneratorData.resourceType) {
                    nearbyResourceAmount++;
                }
            }
        }

        nearbyResourceAmount =
Mathf.Clamp(nearbyResourceAmount, 0,
resourceGeneratorData.maxResourceAmount);

        return nearbyResourceAmount;
    }



    private ResourceGeneratorData
resourceGeneratorData;
    private float timer;
    private float timerMax;

    private void Awake() {
        resourceGeneratorData =
GetComponent<BuildingTypeHolder>().buildingType.r
esourceGeneratorData;
        timerMax = resourceGeneratorData.timerMax;
    }

    private void Start() {
        int nearbyResourceAmount =
GetNearbyResourceAmount(resourceGeneratorData,
transform.position);

        if (nearbyResourceAmount == 0) {

            enabled = false;
        } else {
            timerMax = (resourceGeneratorData.timerMax /
2f) +
                resourceGeneratorData.timerMax *
                (1 - (float)nearbyResourceAmount /
resourceGeneratorData.maxResourceAmount);
        }
    }

    private void Update() {
        timer -= Time.deltaTime;
        if (timer <= 0f) {
            timer += timerMax;
ResourceManager.Instance.AddResource(resourceGene
ratorData.resourceType, 1);
        }
    }

    public ResourceGeneratorData
```

```
GetResourceGeneratorData() {
        return resourceGeneratorData;
    }

    public float GetTimerNormalized() {
        return timer / timerMax;
    }

    public float GetAmountGeneratedPerSecond() {
        return 1 / timerMax;
    }

}

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

public class ResourceManager : MonoBehaviour {

    public static ResourceManager Instance { get;
private set; }


    public event EventHandler
OnResourceAmountChanged;

    [SerializeField] private List<ResourceAmount>
startingResourceAmountList;

    private Dictionary<ResourceTypeSO, int>
resourceAmountDictionary;

    private void Awake() {
        Instance = this;

        resourceAmountDictionary = new
Dictionary<ResourceTypeSO, int>();

        ResourceTypeListSO resourceTypeList =
Resources.Load<ResourceTypeListSO>(typeof(Resour
ceTypeListSO).Name);

        foreach (ResourceTypeSO resourceType in
resourceTypeList.list) {
            resourceAmountDictionary[resourceType] = 0;
        }

        foreach (ResourceAmount resourceAmount in
startingResourceAmountList) {
            AddResource(resourceAmount.resourceType,
resourceAmount.amount);
        }
    }
    public void AddResource(ResourceTypeSO
resourceType, int amount) {
        resourceAmountDictionary[resourceType] +=
amount;

        OnResourceAmountChanged?.Invoke(this,
EventArgs.Empty);
    }

    public int GetResourceAmount(ResourceTypeSO
resourceType) {
        return resourceAmountDictionary[resourceType];
    }

    public bool CanAfford(ResourceAmount[]
resourceAmountArray) {
        foreach (ResourceAmount resourceAmount in
resourceAmountArray) {
            if
(GetResourceAmount(resourceAmount.resourceType)
>= resourceAmount.amount) {

            } else {

                return false;
            }
        }


        return true;
    }

    public void SpendResources(ResourceAmount[]
resourceAmountArray) {
        foreach (ResourceAmount resourceAmount in
resourceAmountArray) {
resourceAmountDictionary[resourceAmount.resourceT
ype] -= resourceAmount.amount;
        }
    }

}

using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using TMPro;

public class ResourcesUI : MonoBehaviour {

    private ResourceTypeListSO resourceTypeList;
    private Dictionary<ResourceTypeSO, Transform>
resourceTypeTransformDictionary;

    private void Awake() {
        resourceTypeList =
Resources.Load<ResourceTypeListSO>(typeof(Resour
ceTypeListSO).Name);

        resourceTypeTransformDictionary = new
Dictionary<ResourceTypeSO, Transform>();

        Transform resourceTemplate =
transform.Find("resourceTemplate");
        resourceTemplate.gameObject.SetActive(false);

        int index = 0;
        foreach (ResourceTypeSO resourceType in
resourceTypeList.list) {
            Transform resourceTransform =
Instantiate(resourceTemplate, transform);
            resourceTransform.gameObject.SetActive(true);

            float offsetAmount = -160f;
resourceTransform.GetComponent<RectTransform>().
anchoredPosition = new Vector2(offsetAmount * index,
0);

resourceTransform.Find("image").GetComponent<Ima
ge>().sprite = resourceType.sprite;

resourceTypeTransformDictionary[resourceType] =
resourceTransform;
```

```
            index++;
        }
    }

    private void Start() {
ResourceManager.Instance.OnResourceAmountChange
d += ResourceManager_OnResourceAmountChanged;

        UpdateResourceAmount();
    }

    private void
ResourceManager_OnResourceAmountChanged(object
sender, System.EventArgs e) {
        UpdateResourceAmount();
    }

    private void UpdateResourceAmount() {
        foreach (ResourceTypeSO resourceType in
resourceTypeList.list) {
            Transform resourceTransform =
resourceTypeTransformDictionary[resourceType];

            int resourceAmount =
ResourceManager.Instance.GetResourceAmount(resour
ceType);
resourceTransform.Find("text").GetComponent<TextM
eshProUGUI>().SetText(resourceAmount.ToString());
        }
    }


}

using UnityEngine;

public class Tower : MonoBehaviour {

    [SerializeField] private float shootTimerMax;

    private float shootTimer;
    private Enemy targetEnemy;
    private float lookForTargetTimer;
    private float lookForTargetTimerMax = .2f;
    private Vector3 projectileSpawnPosition;

    private void Awake() {
        projectileSpawnPosition =
transform.Find("projectileSpawnPosition").position;
    }

    private void Update() {
        HandleTargeting();
        HandleShooting();
    }

    private void HandleTargeting() {
        lookForTargetTimer -= Time.deltaTime;
        if (lookForTargetTimer < 0f) {
            lookForTargetTimer +=
lookForTargetTimerMax;
            LookForTargets();
        }
    }

    private void HandleShooting() {
        shootTimer -= Time.deltaTime;
        if (shootTimer <= 0f) {
            shootTimer += shootTimerMax;
            if (targetEnemy != null) {
ArrowProjectile.Create(projectileSpawnPosition,
targetEnemy);
            }
        }
    }

    private void LookForTargets() {
        float targetMaxRadius = 30f;
        Collider2D[] collider2DArray =
Physics2D.OverlapCircleAll(transform.position,
targetMaxRadius);

        foreach (Collider2D collider2D in
collider2DArray) {
            Enemy enemy =
collider2D.GetComponent<Enemy>();
            if (enemy != null) {
                // Is a enemy!
                if (targetEnemy == null) {
                    targetEnemy = enemy;
                } else {
                    if (Vector3.Distance(transform.position,
enemy.transform.position) <
                        Vector3.Distance(transform.position,
targetEnemy.transform.position)) {
                        // Closer!
                        targetEnemy = enemy;
                    }
                }
            }
        }
    }

}
```