Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Ігровий програмний застосунок у жанрі RPG з елементами економічної стратегії та roguelike. Економічні механіки, механіки на рівні історії
(тема)

Виконав:
студент 4 курсу, групи ПЗПІ-20-4

Прудіус В.Ю.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Програмна інженерія
(повна назва освітньої програми)

Керівник ст.викл. кафедри ПІ Новіков Ю.С.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ З.В.Дудар
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми Освітньо-професійна
Освітня програма Програмна Інженерія
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Прудіусу Владиславу Юрійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Ігровий програмний застосунок у жанрі RPG з елементами економічної стратегії та roguelike. Економічні механіки, механіки на рівні історії
Затверджена наказом по університету від 20.05. 2024р. № 471 Ст
2. Термін подання студентом роботи до екзаменаційної комісії 06.06.2024
3. Вихідні дані до роботи Розробити механіки економічних стратегій та розвитку персонажа для ігрового програмного застосунку в жанрі RPG з елементами економічної стратегії та roguelike. Для розробки використовувати можливості ігрового рушія Unity та середовище розробки Visual Studio.
4. Перелік питань, що потрібно опрацювати в роботі
Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проектування програмного забезпечення, опис прийнятих програмних рішень, тестування розробленого програмного забезпечення, висновки, додатки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	10.09.2023	*виконано*
2	Створення специфікації ПЗ	15.09.2023	*виконано*
3	Проектування ПЗ	15.10.2023	*виконано*
4	Розробка ПЗ	30.12.2023	*виконано*
5	Тестування ПЗ	30.01.2024	*виконано*
6	Оформлення пояснювальної записки	18.05.2024	*виконано*
7	Підготовка презентації та доповіді	25.05.2024	*виконано*
8	Нормоконтроль, рецензування	01.06.2024	*виконано*
9	Здача роботи у електронний архів	01.06.2024	*виконано*
10	Попередній захист	03.06.2024	*виконано*
11	Допуск до захисту у зав. кафедри	03.06.2024	*виконано*
12	Захист роботи	06.06.2024	*виконано*

Дата видачі завдання 8 квітня 2024р.

Студент (ка)_____ Прудіус В. Ю.
(підпис)

Керівник роботи _____ ст.викл. кафедри ПІ Новіков Ю.С.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Кваліфікаційна робота бакалавра, 115 стор., 65 рис., 1 табл., 14 джерел.

ГРА, ІГРОВИЙ РУШІЙ, МЕНЕДЖМЕНТ, ПЕРСОНАЖ, ПОСЕЛЕННЯ, РЕСУРСИ, РОЗВИТОК, СТРАТЕГІЯ, C#, UNITY

Об’єктом розробки є частина механік гри у жанрі RPG з елементами економічної стратегії та roguelike, що пов’язана з розвитком персонажа та менеджментом ресурсів.

Метою цієї розробки є створення цікавого ігрового середовища, що містить набір механік які допоможуть гравцеві прогресувати в грі та урізноманітнюватимуть ігровий процес не лише в рамках окремих ігрових сесій а й досвіду що виникає в наслідок багаторазового проходження гри.

Для виконання завдання пов’язаного з цією розробкою використовуються середовища розробки Visual Studio 2022 і Unity Editor, ігровий рушій Unity 2023 та мова програмування C#.

Результатом виконання роботи є повноцінно функціонуюча частина гри яка містить в собі механіки створення та розвитку персонажа, управління ресурсами та інвентарем та деякі основні системи гри, такі як система збережень, характеристик, модифікаторів тощо.

GAME, GAME ENGINE, MANAGEMENT, CHARACTER, SETTLEMENT, RESOURCES, DEVELOPMENT, STRATEGY, C#, UNITY

The object of development is a part of the game mechanics in the RPG genre with elements of economic strategy and roguelike, related to character development and resource management.

The purpose of this development is to create an interesting game environment that contains a set of mechanics that will help the player progress in the game and diversify

the gameplay not only within individual game sessions but also the experience that arises as a result of repeated playthroughs.

To accomplish the task related to this development, the Visual Studio 2022 and Unity Editor development environments, the Unity 2023 game engine, and the C# programming language are used.

The result of the work is a fully functioning part of the game that contains the mechanics of character creation and development, resource and inventory management, and some basic game systems, such as the system of saves, characteristics, modifiers, etc.

Я, Прудіус Владислав Юрійович, студент гр. ПЗПІ-20-4, здобувач вищої освіти на першому (бакалаврському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Ігровий програмний застосунок у жанрі RPG з елементами економічної стратегії та roguelike. Економічні механіки, механіки на рівні історії», що буде представлена до екзаменаційної комісії для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Усі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови до допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ВСТУП

В сучасному світі ігрова індустрія займає визначальну позицію серед розважальних сегментів, пропонуючи користувачам не лише можливість розважитися, але й пізнавати нові інтерактивні світи, взаємодіяти з ними та розвивати креативні навички. За даними багатьох аналітичних та фінансових платформ загальний дохід та об’єми ринку відеоігор значно переважають комбіновану вартість музичної та кінематографічної галузей, станом на 2022 рік [1]. Враховуючи вищесказане, можна зробити два основні висновки щодо актуальності теми розробки, а саме розробка ігрових застосунків в даних умовах є актуальною та поширеною, а також через великий об’єм ринку та його особливості ігровий продукт має пропонувати гравцям новий досвід.

Одним із найцікавіших і найбільш популярних жанрів в ігровій індустрії є RPG. Рольові ігри здебільшого відображають розвиток персонажа у віртуальному світі, де рішення гравців мають безпосередній вплив не лише на ігровий процес, а, часто, і на історію, що подається в грі. Однак, сучасний ринок пропонує багато класичних рішень в цьому жанрі [2], багато з яких, за роки еволюції, напрацювали формули «ідеальних RPG», тож для зацікавлення гравців в є тенденція до змішування різних жанрів. Тому поєднання елементів економічної стратегії і roguelike в рамках жанру RPG може бути надзвичайно цікавим та актуальним.

Метою даної дипломної роботи є дослідження можливостей створення ігрового програмного застосунку, що об'єднує в собі жанри RPG, елементи економічної стратегії та roguelike. Найбільше уваги тут буде приділено механікам розвитку персонажа та управління економічними процесами, а також методи та рішення для поєднання таких підходів в одній грі.

Результатами розробки мають бути повнофункціональні компоненти гри, які за доопрацювання та доповнення іншими необхідними механіками що не розглядаються в даній роботі можуть становити повноцінний ігровий продукт, готовий до розповсюдження.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

Предметною галуззю розробки є ринок ігрових застосунків пов’язаний з комп’ютерними рольовими іграми, тобто включає в себе велику кількість різноманітних рішень що повністю або частково включають в себе основні рольові механіки та підходи. Цей сегмент ринку є ледь не найрозвиненішим серед усього різноманіття ігрових застосунків та пропонує безліч ігор різного об’єму та з різними особливостями. Тож проаналізуємо предметну галузь, для визначення того на що слід звернути увагу при розробці даного проекту, від чого слід відмовитись і на які тенденції орієнтуватись.

Вхідною точкою нашого аналізу буде відповідь на питання: «Чого люди грають в RPG?». Простої відповіді на це питання не буде, адже цей жанр ігор є дуже комплексним, мінливим та різноманітним. Тож слід сконцентруватись на моментах, що є спільними серед більшості RPG, а саме віртуальний світ, який залежить від дій персонажа який залежить від гравця, і для великої кількості людей, саме це є визначальним для формування інтересу. Адже завдяки цьому кожен може реалізувати свої бажання, спробувати щось нове, цікаве, чи просто уявити себе кимось іншим [3] див. рис 1.1.

Рисунок 1.1 – Action RPG Witcher 3 (за даними [3])

За цієї точки зору, важливою є наявність в грі механік що дозволяють гравцеві змінювати навколишній світ. До цього є декілька підходів. Перший це вплив на історію, коли дії гравця впливають на сюжет гри. Це, мабуть, найбільш очевидний але найбільш складний в реалізації метод, адже приходиться або прописувати складний розгалужений сюжет, або вигадувати комплексні схеми та підходи для його імітації, мінімізуючи ресурси на розробку. Другий підхід це вплив напряму на світ, коли основний сюжет є більш менш лінійним але окремі дії гравця змінюють окремі елементи світу навколо, що також може бути складним в реалізації. І третім підходом є вплив на самого персонажа, коли дії гравця визначають ким він є в цьому світі, як він взаємодіє зі світом та як йому на це відповідає світ. В більшості ігор даного жанру хоча б один з описаних вище підходів реалізований в тій чи іншій мірі, часто зустрічаються і такі де реалізовані всі 3.

Приймаючи до уваги усе написане вище, можемо дати лаконічну відповідь на питання чому у ці ігри грають – тому що це цікаво. Цікаво дивитись як ті дії які вчиняє гравець змінюють гру. І чим більшими є зміни тим цікавіше грати, чим більше відрізняється кожне проходження від кожного наступного тим більший інтерес і цікавість це викликає у гравців.

Що ж стосується жанрів які планується поєднати з RPG розробка якої є предметом роботи, то досить легко можна визначити їх особливості та причина зацікавленості ними аудиторії, адже стратегії та roguelike є не менш популярними жанрами.

Щодо стратегій, то тут основним спільним концептом жанру є управління, стосовно ж економічних стратегій це може бути управління ресурсами чи якимись структурними одиницями (поселення, виробничі потужності, компоненти підприємств тощо). Відносно ж сучасних roguelike це безумовно дослідження випадково генерованих рівнів після чого присутня можливість зберегти прогрес та використовуючи отримані в подорожах ресурси покращити свого персонажа. Здавалося б, настільки різні жанри просто неможливо поєднати, але деякі точки дотику в них є, і їх можна використати для створення цікавої гри.

1.2 Виявлення та вирішення проблем

В сучасних рольових іграх є три основні проблеми. Вони здебільшого або складні в розробці і впровадженні контенту, або мають одноманітний ігровий процес що мало змінюється з часом і може набридати, або після першого проходження залишається настільки мало невикористаного контенту, що в цю гру не має сенсу грати ще раз. Ці проблеми часто вирішуються підмішуванням інших ігрових жанрів в гру, що і пропонується реалізувати впродовж виконання роботи.

Для більш детального дослідження подібних підходів сконцентруємо увагу саме на механіках, що пов’язані зі стратегічними іграми та можливостях їх інтеграції в рольову гру з елементами roguelike. Для цього оберемо 5 ігор, що мають схожі або споріднені концепції та механіки, та проаналізуємо їх проблеми.

Першою обраною грою є Cult of the lamb [4] див рис 1.2. Це один з найбільш схожих на розроблюваний проект аналогів, адже в ній присутні основні механіки рольових ігор, управління ресурсами та поселенням зі стратегій та дослідження випадкових локацій з roguelike. Має високі оцінки та якість. Але найбільше всього нас цікавить саме стратегічні механіки, управління ресурсами, розвиток власного поселення та персонажа.

Рисунок 1.2 – Гра Cult of the lamb (за даними [4])

Другою грою є відома покрокова стратегія XCOM 2 [5] див. рис. 1.3. Проект зовсім не схожий на попередній, але має низку цікавих механік які є ідеальним відображенням того що необхідно розроблюваному проекту з точки зору комбінації рольових та стратегічних механік. А саме механіки прогресу по історії, розвитку персонажів, генерації та управління ресурсами

Рисунок 1.3 – Гра XCOM 2 (за даними [5])

Третя гра – класика стратегій в реальному часі Age of empires 2 [6] див. рис. 1.4. Мстить низку механік без яких просто неможливо уявити сучасні RTS, деякі з них, зокрема містобудівництво, виробництво та управління ресурсами, дерева технологій що відрізняються у різних народів.

Рисунок 1.4 – Гра Age of Empires 2 (за даними [6])

Четверта гра це Frostpunk [7] див. рис. 1.5. Ще один продукт зовсім не схожий на попередні але з неймовірно гарними оцінками та концентрацією класичних механік економічних містобудівних стратегій на які безумовно слід звернути увагу, як і на механіки розвитку.

Рисунок 1.5 – Гра Frostpunk (за даними [7])

І остання гра це Fallout shelter [8] див. рис 1.6. Ще одна містобудівна стратегія яка виросла з маленької мобільної гри до масштабної крос-платформеної гри, при цьому отримавши купу нових цікавинок, пов’язаних з інтеграцією механік рольових та roguelike ігор.

Рисунок 1.6 – Гра Fallout shelter (за даними [8])

Далі для полегшення аналізу обраних ігор сформуємо порівняльну таблицю в якій визначимо показники по критеріям що нас цікавлять для кожної гри див. табл. 1.1. Дані з таблиці що позначають критерії починаючи з «Сюжет» і нижче, мають ранжування від 0 до 5, де 0 позначає відсутність предмету оцінювання, а 5 найвищий показник у порівнянні з конкурентами.

Таблиця 1.1 – Порівняльна таблиця ігор зі схожими механіками (таблиця виконана самостійно)

Критерій	Cult of the Lamb	X.C.O.M 2	Age of empires 2	Frostpunk	Fallout Shelter
Оцінка в Steam	94	84	94	92	88
Підтримка мов	10	11	17	12	6
Кількість досягнень	42	88	275	115	35
Ціна	~~499₴~~	~~679₴~~	~~505₴~~	~~600₴~~	0
Час гри	24	79	-	53	90
Сюжет	3	5	2	3	1
Візуальний стиль	5	4	3	4	5
Складність	4	4	5	5	2
Розбудова поселення	4	3	5	5	3
Управління ресурсами	5	2	5	5	2
Прокачка персонажа	2	5	0	0	3

Кінець таблиці 1.1

Прокачка в цілому	4	3	5	5	2
Кастомізація	2	5	0	0	4

Приймаючи до уваги аналіз обраних ігор, можна визначити низку моментів які можуть слугувати вирішенням для проблем на які було вказано вище у розділі:

– перенесення частини функцій пов’язаних з розвитком персонажа, на функції розвитку поселення (ігровий процес залежить від того в якому стані знаходиться його поселення);
– впровадження системи взаємодії з народами, як іншу частину функцій пов’язаних з розвитком персонажа (ігровий процес залежить від того на якому рівні відносин знаходиться гравець з кожним з 5 народів гри);
– ресурси мають відігравати ключову роль у розвитку поселення, що зобов’язує використовувати різні методи видобутку та переробки ресурсів;
– впровадження різного роду обмежень, що спонукатимуть гравця йти на компроміси між різними стилями гри, більш детально вивчати ігровий процес та, заохочуватиме до перепроходжень гри;
– впровадження модифікованих механік з настільних рольових ігор для зацікавлення гравців та впровадження ще більшого ступеня урізноманітнення ігрового процесу за рахунок випадковостей та реакцій на дії та прогрес гравця.

Реалізація описаних вище моментів в грі яка є об’єктом роботи, має бути основною задачею для створення конкурентоспроможного та цікавого для користувачів додатку. Тому ці моменти слід формалізувати та деталізувати для формування повноцінної задачі вирішенням якої будуть вихідні матеріали даної роботи.

1.3 Постановка задачі

Для вирішення проблем описаний у попередніх розділах пропонується розробити ігровий програмний застосунок, що міститиме наступний функціонал:

а) створення нового персонажа з вказанням такої інформації:
   1) ім’я;
   2) стать;
   3) аватар, іконка персонажа у грі;
   4) розподіл характеристик персонажа за 5 основними групами
б) спрощений процес створення персонажа, що включає в себе вибір одного з запропонованих досьє, що були згенеровані штучним інтелектом з попередньо розподіленими значеннями;
в) управління ресурсами, що включає в себе:
   1) перегляд наявних ресурсів;
   2) перегляд статистики по використанню та отриманню ресурсів;
   3) ринок ресурсів;
   4) вільне використання ресурсів на різні потреби (купівля спорядженння, будівництво, тощо);
г) управління поселенням, що включає в себе:
   1) перегляд поселення в режимі вільної камери;
   2) перегляд можливостей для будівництва;
   3) розміщення нових будівель;
   4) знесення старих будівель;
   5) взаємодія з розміщеними будівлями;
д) управління інвентарем, що включає:
   1) перегляд наявного екіпірування, та його деталей;
   2) управління екіпіруванням що використовується ігровим персонажем;

3) ринок екіпірування, що включає розбір, продаж та купівлю екіпірування;

е) відстеження розвитку персонажа, що включає:

1) перегляд прогресу, та визначення впливу основних характеристик прогресу: рівнем відомості, рівнем лояльності долі та взаємовідносин з народами;
2) перегляд основних характеристик персонажа;
3) перегляд та розподіл очок по деревам розвитку кожного з народів;

ж) впровадження системи завдань, що включає в себе:

1) перегляд наявних завдань, відслідковування обраних та відмова від їх виконання;
2) відслідковування прогресу проходження квестів;
3) видача нових квестів та нарахування винагороди при виконанні завдань;

и) збереження та завантаження ігрового прогресу;

к) впровадження гнучких і масштабованих рішень для основних систем проекту, що мають підтримувати швидке і зручне додавання або вилучення контенту за допомогою інструментів рушія, без потреби модифікації вихідного коду.

Реалізувавши описаний вище функціонал, отримаємо повноцінну базу для створення цікавого рішення в сфері рольових ігрових [9] програмних застосунків що зможе зацікавити користувачів відповідно до сучасних умов ринку.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

2.1 Описання гри

Механіка – RPG з елементами Roguelike та економічної стратегії.

Технологія – ПК.

Історія – головний герой – простолюдин, обраний.

Естетика – темне фентезі, гротескність, lowpoly.

Дана гра має на меті комбінування механік з популярних ігрових жанрів та максимально спрямована на надання різноманітного ігрового досвіду. Тому основною концепцією гри є неповторність ігрового досвіду для кожної історії окремо та можливість використання різних геймплейних стилів в залежності від вподобань користувача.

Щодо ігрового процесу, то його можна умовно поділити на 2 складові частини, розвиток власного поселення і персонажа та дослідження підземель з виконанням квестів. В даній роботі основний акцент зміщено в бік першої частини ігрового процесу.

Перед початком нової історії гравцеві пропонується створити нового персонажа та кастомізувати його характеристики, базуючись на власному розподілі очок або скориставшись більш просунутою системою формування персонажа.

Ця система пропонуватиме набір історій, пов’язаних з минулим персонажа, на основі яких базуватиметься розподіл очок характеристик. Обравши одну з запропонованих історій, гравець отримуватиме сформований розподіл характеристик, що напряму впливатиме на ігровий процес.

Ці характеристики містять 5 основних розділів, Сила, Вдача, Розсудок, Харизма та Спритність, що як напряму так і опосередковано впливатимуть на ігровий процес, розвиток та історію, кожна у свій спосіб, надаючи можливості урізноманітнення кожного проходження гри.

За історією головний герой стає на чолі невеликого поселення, що росте навколо напівзабутої таверни де буде відбуватись основний розвиток персонажу. В

самому поселенні планується низка активностей, спрямованих на економічні процеси (ринки, переробка ресурсів), розвиток персонажа (інвентар, уміння) та поселення (побудова, знесення взаємодія з будівлями).

Першою з таких активностей є налагодження взаємозв’язків з різними народами, що дозволяє поглибитись в їх культуру та дізнатись більше про їх технології. Відповідно покращуючи відносини гравець відкриває все більше і більше доступних таємниць народів, таких як спорядження, нові будівлі, активні уміння, секрети та потужні особливі технології. Налагоджувати ці відносини гравець може виконуючи у підземеллях завдання, отримані від представників кожного з народів;

Особливе місце займає також розбудова власного поселення, задля видобутку ресурсів, отримання доступу до тих чи інших елементів спорядження та для отримання постійних бонусів. Кожна будівля є унікальним об’єктом зі своїми особливостями та належать до одного з 5 народів, а для того щоб отримати можливості до їх розміщення необхідно розвивати відносини з одним з обраних народів. Ще однією цікавою особливістю поселення є обмежена площа під забудову, що має вигляд визначеної кількості ділянок на яких можна розмістити будівлі. Кількість цих ділянок є значно меншою за загальну кількість будівель, що заставляє гравця постійно оновлювати поселення встановлюючи нові, більш ефективні чи корисні будівлі, при цьому відмовляючись від застарілих;

Поряд з забудовою поселення, у гравця знаходитиметься ще один обов’язок – управління ресурсами, які добуваються при проходженні підземель. Для цього є багато різних способів, ринки ресурсів, їх переробка, вкладання у власне поселення або спорядження тощо;

Просуваючись сюжетом гравець отримуватиме вплив та впізнаваність у світі, що відбиватиметься не лише на характеристиках гравця, а й на ігровому процесі. Також поряд з впізнаваністю гравець може зіткнутися з потужними прокляттями, якими Доля “нагороджує” гравця за негативний вплив на світ.

2.2 Цільова аудиторія

Основна цільова аудиторія проекту - це люди обох полів з переважанням чоловічої частини, від 18 до 35 років. Різні підходи для проходження гри та пов’язана з цим гнучкість та можливі повторні перепроходження гри, наряду з цікавою історією, моральними виборами прокачкою персонажа та міста, менеджментом ресурсів та можливістю відіграшу конкретної ролі на острові неодмінно приверне увагу користувачів.

2.3 Час ігрової сесії

Так як гра проектується для персональних комп’ютерів, то час ігрової сесії є досить великим, мінімальний поріг приблизно 60 хвилин. За цей час гравець має змогу пройти одне “підземелля” і виконати декілька квестів для отримання ресурсів та підтримки взаємовідносин з фракціями.

2.4 Технічні характеристики

Випуск гри планується на платформі персональних комп’ютерів, з наступними характеристиками (за основу взято характеристики аналогів):

Мінімальні:

- потребує 64-бітний процесор та операційну систему;
- операційна система - Windows 7 або новіша;
- процесор - Intel Core i3-3240 (2 * 3400); AMD FX-4300 (4 * 3800);
- оперативна пам’ять - 4 GB ОП

– відеокарта - GeForce GTX 560 Ti (1024 VRAM); Radeon HD 7750 (1024 VRAM);

– місце на диску - 4 GB доступного місця

Рекомендовані:

– потребує 64-бітний процесор та операційну систему;

– операційна система - Windows 10/11;

– процесор - Intel Core i5-3470;

– оперативна пам’ять - 8 GB ОП

– відеокарта - GeForce GTX 1050 (2048 VRAM); Radeon R9 380 (2048 VRAM);

– місце на диску - 4 GB доступного місця

Два попередні списки формують вимоги до апаратного та програмного забезпечення користувачів. Вони сформовані на основі технічних характеристик ігор-конкурентів що мають співставні показники продуктивності та комплексності як і проект що є об’єктом розробки. Мінімальні вимоги визначають технічні характеристики персонального комп’ютера гравця необхідні для стабільної роботи застосунку в режимі з пониженими характеристиками. Рекомендовані вимоги – це набір обладнання необхідний для стабільної роботи додатку використовуючи будь яку комбінацію графічних налаштувань.

3 АРХІТЕКТУРА ТА ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Загальні положення

В цьому розділі визначимо низку умовностей які характеризують проект відносно середовищ розробки та інструментів які в них використовуються. Перш за все, слід зазначити, розробка буде проводитись за допомогою мови програмування C# в середовищі рушія Unity, що накладає обов’язки слідувати об’єктно орієнтованому підходу та компонентній архітектурі.

Систематизація вмісту в рушії проводиться наступним чином, на вищому рівні знаходиться концепт сцени, який за своєю суттю є контейнером який містить в собі ігрові об’єкти які завантажуються та відмальовуються при завантаженні сцени. Кожна дійова особа в середині рушія Unity є екземпляром GameObject, який інкапсулює в собі функціонал щодо основних функцій рушія щодо розміщення на сцені, відображення, життєвого циклу об’єкту тощо. Також кожен з GameObject може містити в собі низку компонентів MonoBehaviour які є C# класами, що містять поведінку об’єктів. Ще однією не менш важливою частиною інструментів рушія є ScriptableObject, що також є звичайними класами мови C#, але в середині рушія використовуються як самостійні контейнери даних які не потребують розміщення на сцені, та можуть містити незначний функціонал.

Тримаючи всі описані вище особливості використовуваних інструментів у голові перейдемо до проектування архітектури проекту.

3.2 UML проектування

Розпочнемо з однієї з ключових систем майбутнього проекту – системи характеристик. Для початку визначимо що це, і як це має працювати. Під

характеристиками слід розуміти набір якихось параметрів які властиві якомусь об’єкту. Це наприклад 5 основних характеристик персонажа (Сила, Спритність, Розсудок, Вдача, Харизма), які можуть бути використані будь якими компонентами системи для власних розрахунків. При чому в деяких випадках до цих показників можуть вноситись зміни, при цьому базові показники не мають змінюватись. Отже окрім характеристик слід ввести також концепцію модифікаторів, які мають впливати на показники характеристик не змінюючи при цьому їх базові значення.

Далі для зручної взаємодії з характеристиками, слід сформувати контейнер характеристик, який має містити в собі набір характеристик для об’єкта який він описує, а також надавати інтерфейси для взаємодії з ними.

Також було б зручно мати спільний інструмент для відслідковування та модифікації показників для усіх контейнерів характеристик, що допоможе накладати зміни на характеристики окремих об’єктів чи їх груп без потреби мати прямі посилання на їх контейнери.

За основу було взято прототип подібної системи, запропонованої одним з користувачів на офіційному форумі рушія [10]. За цим прикладом у нас формуються сутності Stat та Modifier, де перша містить в собі базове значення, список модифікаторів та інкапсулює логіку пов’язану з розрахунками показників характеристики відповідно до модифікаторів. В той же час модифікатори містять значення зміни характеристи, тип, що показує яким чином буде змінюватись базове значення (просто додаванням фіксованого числа, множенням на коефіцієнт, або множенням на коефіцієнт що додається з іншими коефіцієнтами), та ваговий коефіцієнт, що допомагає формувати комплексні формули для розрахунків.

Також для відповідності вимогам до системи описаним вище, вводимо концепції імені характеристики (StatName) що дозволить ідентифікувати їх в різних сутностях, та вводимо додаткові концепції, контейнеру (StatsContainer), що містить в собі список характеристик разом з функціоналом для їх опрацювання, а також глобальний трекер модифікаторів (GlobalModifiers), де у структурованому вигляді зберігаються усі накладені модифікатори, та наявний функціонал по їх накладанню та відкликанню на окремі контейнери та характеристики за визначеними умовами.

Таким чином отримуємо гнучку систему (див. рис. 3.1), де показники різних характеристик можуть змінюватись на льоту, при цьому з високим рівнем надійності та гнучкості в плані формування складних формул для розрахунків, а також масштабованістю, адже щоб додати нову характеристику достатньо додати новий елемент StatName, та оновити дані у відповідних контейнерах.

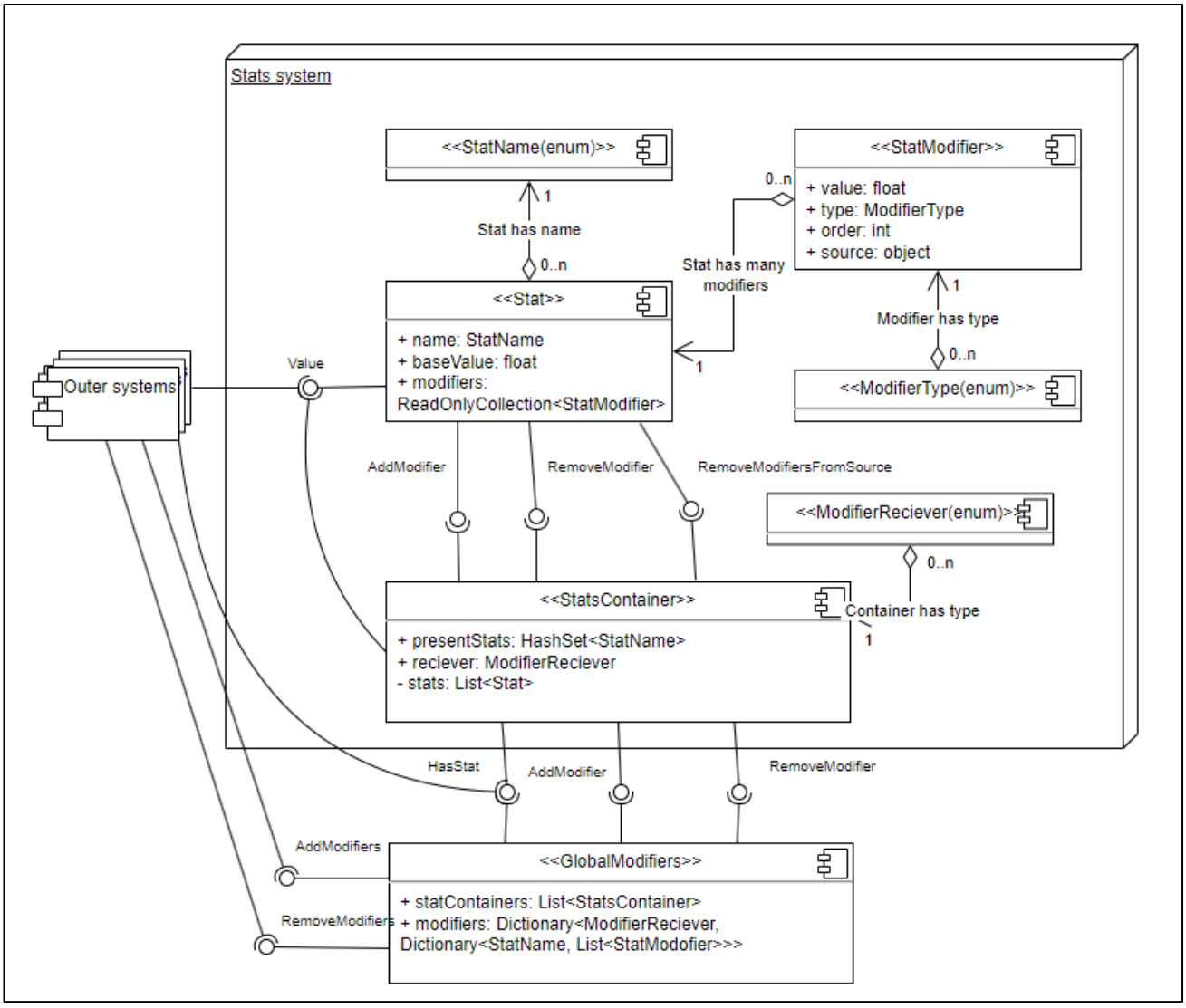


Рисунок 3.1 – Діаграма компонентів системи характеристик (власний рисунок)

На базі концепцій з отриманої системи характеристик, побудуємо не менш важливу систему ресурсів (див. рис. 3.2). З попередньої системи візьмемо концепцію характеристик і замінимо її на ресурс, базові модифікатори розширимо за допомогою ідентифікатора типу транзакцій, що дозволить нам накладати їх

окремо на різного типу дії пов’язані з ресурсами, а у контейнері ресурсів додамо функціонал щодо нарахування та списання ресурсів.

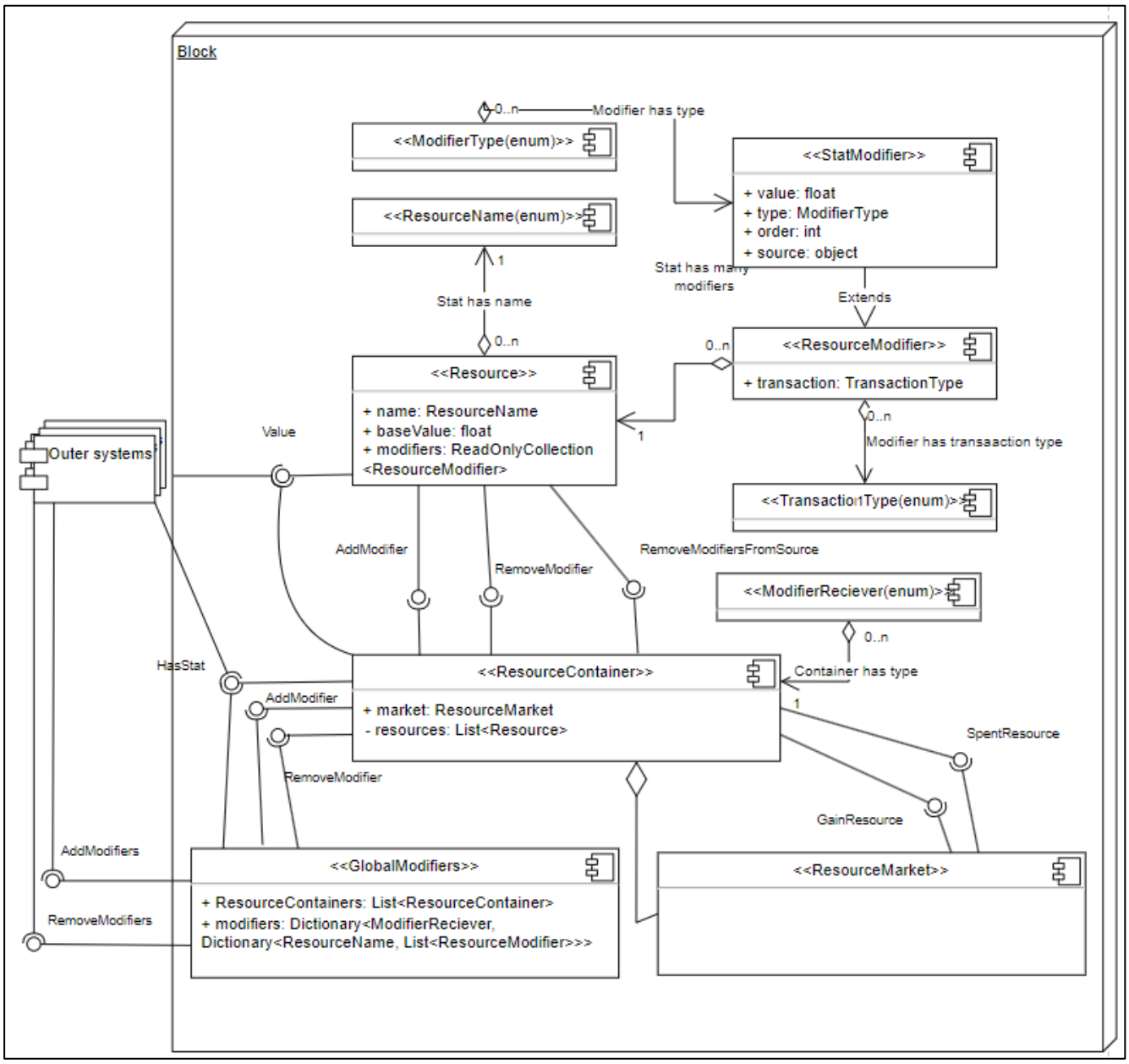


Рисунок 3.2 – Діаграма компонентів системи ресурсів (власний рисунок)

Далі від фундаменту основних систем проекту перейдемо до більш прикладних моментів на яких детально зможемо розглянути структуру сцен, їх об’єктів та компонентів. А саме до системи менеджменту будівель.

Для кращого розуміння цілей, та принципів функціонування цієї системи, розробимо Use-case діаграму (див. рис 3.3) за допомогою якої буде легше

проводити проектування архітектури та подальшого проектування користувацького інтерфейсу для цієї системи.

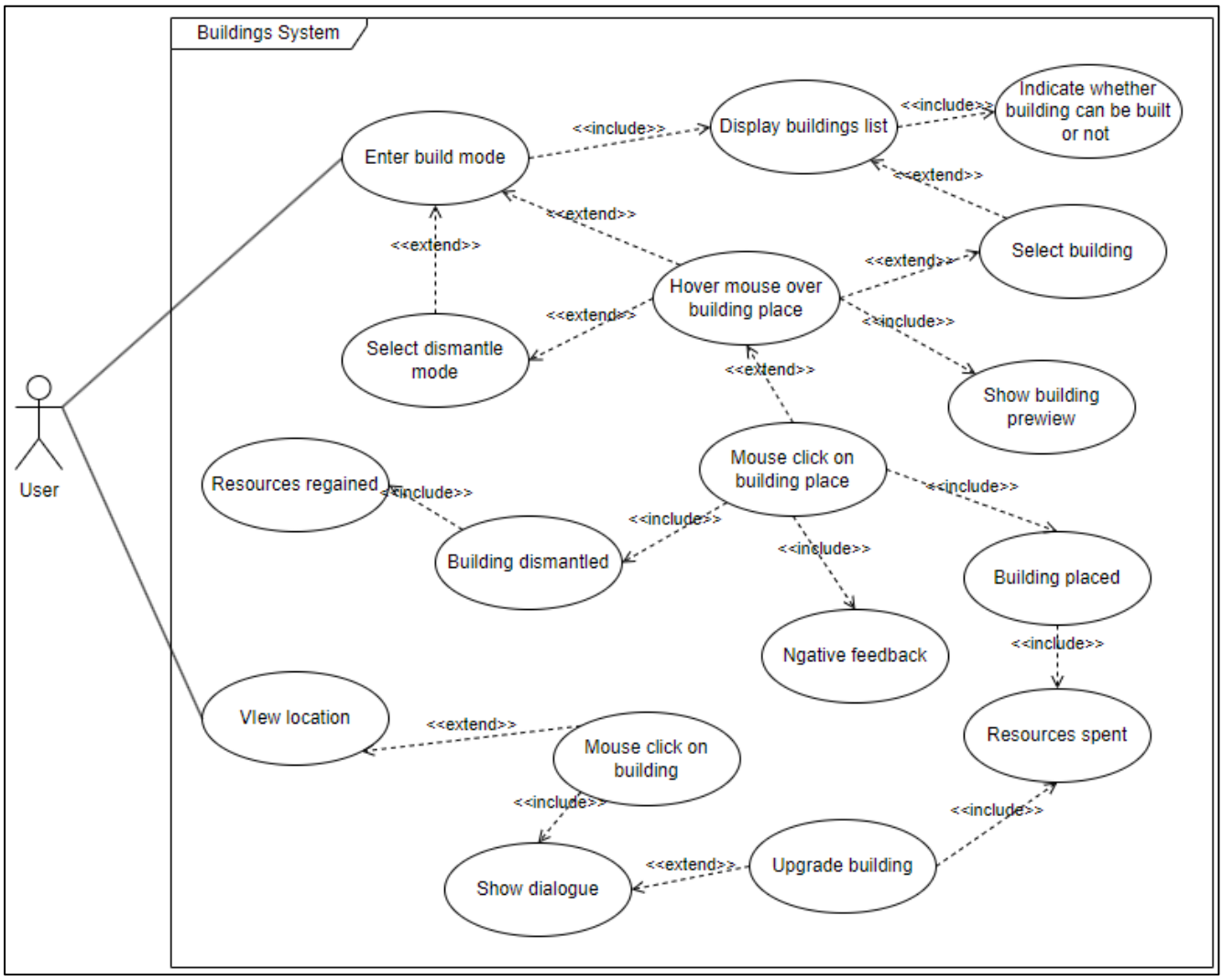


Рисунок 3.3 – Діаграма Use-case системи будівель (власний рисунок)

Продовжимо архітектурне проектування цієї системи за допомогою діаграми класів, а для цього слід визначити наші дійові особи. Для початку це звісно будівля, яку слід розділити на 2 логічних компоненти, перший міститиме дані щодо будівлі, а другий відповідатиме за поведінку будівлі у ігровому світі. Для їх реалізації використаємо два різних типи дійових осіб Unity ScriptableObject та MonoBehaviour відповідно.

Далі нам необхідний контролер будівництва на який ми покладемо обов’язки з обробки дій користувача в режимі будівництва. Але сам по собі, такий компонент

буде завеликим, та складним в управлінні та підтримці, тож декомпозуємо його декілька складових частин, зокрема на систему користувацького вводу, контролер розміщень та систему фідбеку, а також реалізуємо паттерн “Стратегія” для гнучкої можливості переходів між будівництвом та зносом будівель. Загальну структуру майбутньої системи будівель можна переглянути на спрощеній діаграмі класів (див. рис. 3.4), більш детально ознайомитись з якою можна у додатку В.

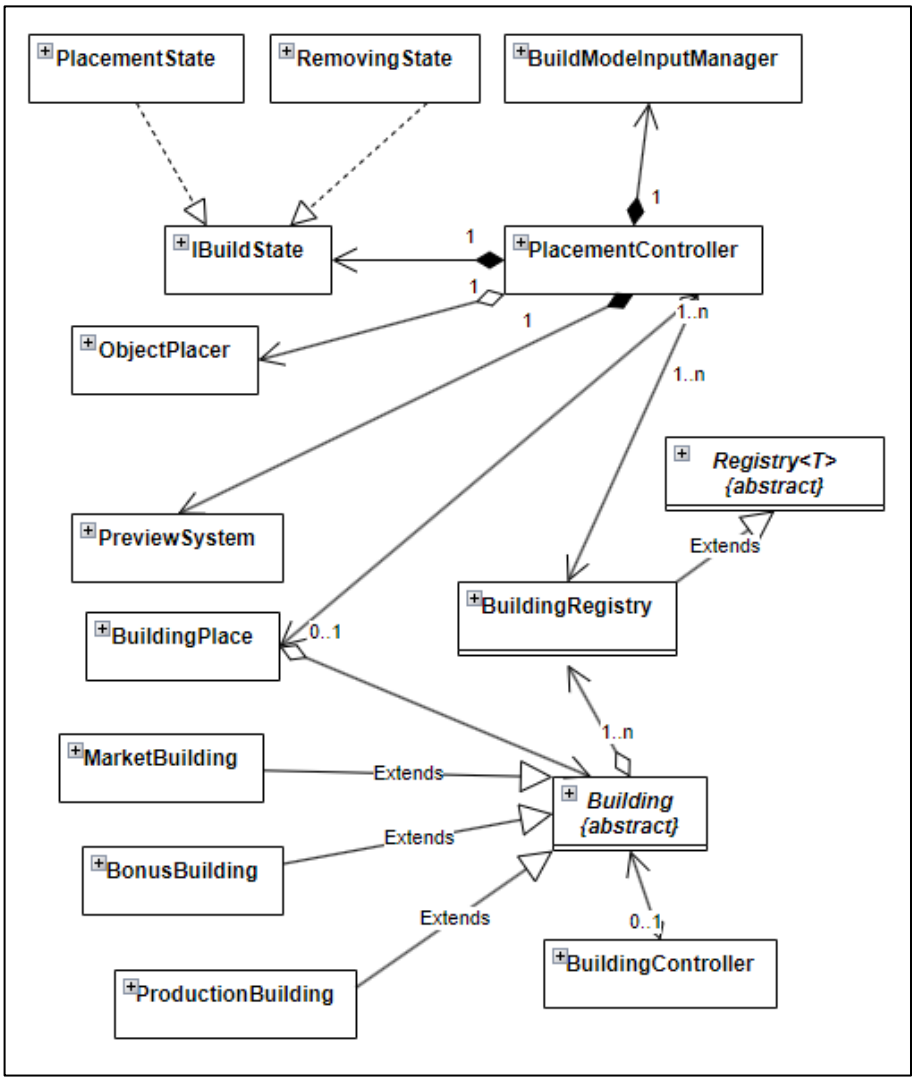


Рисунок 3.4 - Діаграма класів системи будівель у спрощеному вигляді (власний рисунок)

І закінчимо розділ UML проектування ще однією ключовою системою – системою інвентаря та екіпірування. Для цього побудуємо діаграму компонентів цієї системи (див. рис. 3.5) та більш детально розберемо її.

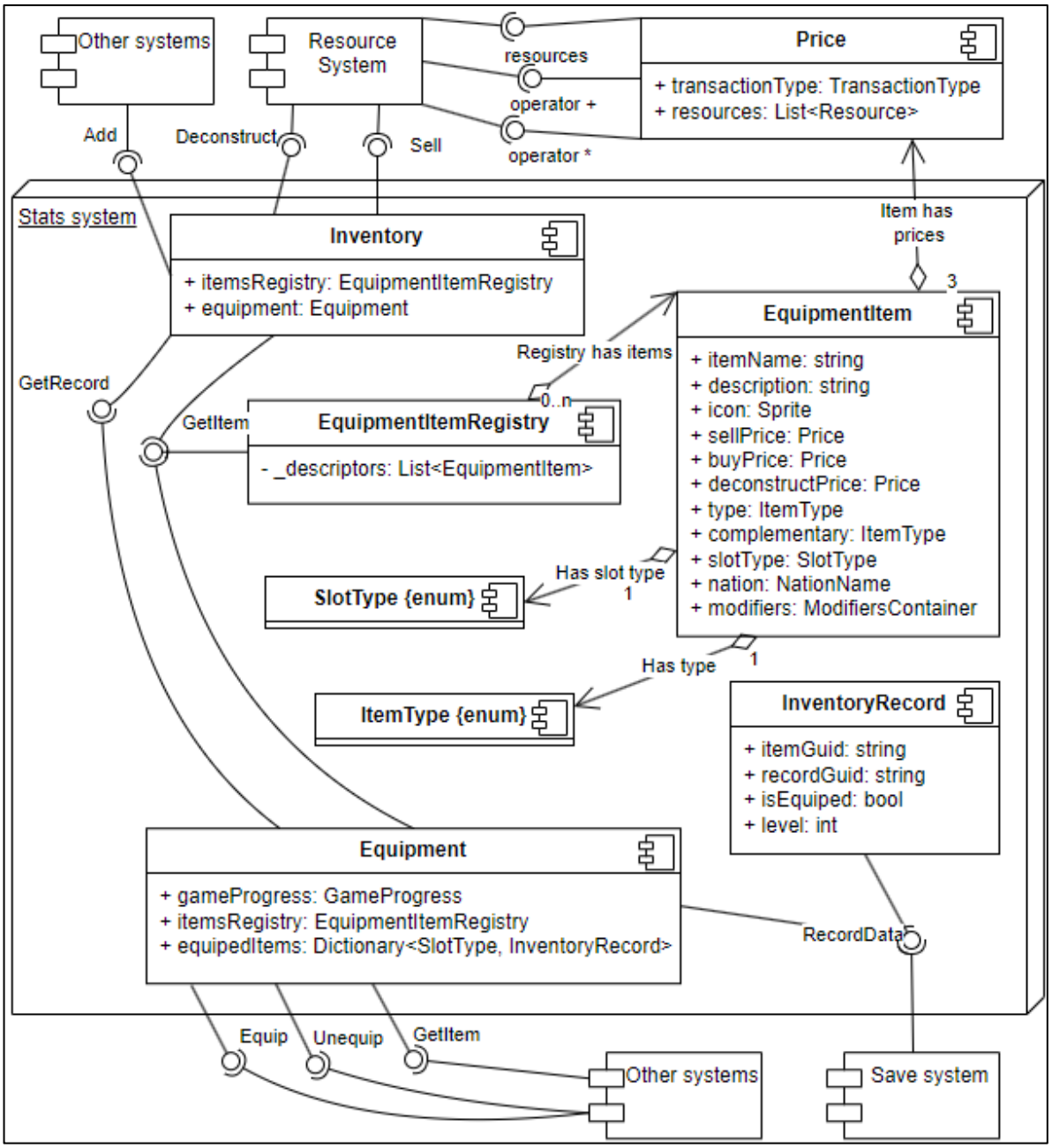


Рисунок 3.5 - Діаграма компонентів системи інвентаря (власний рисунок)

Тож як видно з діаграми вище, маємо 2 основні компоненти цієї системи, це інвентар, що представляє собою контейнер для усіх речей гравця, та екіпірування – контейнер для усіх речей гравця, що на даний момент використовуються. Також для визначення самого предмету який може використовувати гравець, використовуються два класи, один визначає загальні статичні дані для кожного предмету (EquipmentItem), та зберігається в регістрах, що дозволяє швидко

отримувати ці дані з будь якого місця де необхідно не маючи при цьому прямого посилання на предмет. А інший клас (InventoryRecord) є легковісною моделлю даних для передачі та зберігання інформації про наявність в інвентарі гравця тих чи інших предметів.

Також на цій діаграмі гарно видно взаємодію цієї системи з іншими системами гри, зокрема з системою збережень і ресурсів. За допомогою визначених інтерфейсів можемо гарантувати інтеграцію різних систем.

В цьому розділі було розглянуто основні дизайнерські рішення для проектування найбільш важливих систем майбутньої гри. Так як проект є досить великим, то його найбільш прості частини та ті компоненти які мають одноманітні чи типові для середовища розробки підходи були вилучені з розгляду для економії часу.

3.3 Проектування структури зберігання даних

Для початку обговоримо особливості зберігання даних в проекті. Першим моментом на який слід звернути увагу, є вбудовані в ігровий рушій інструменти для збереження даних, це AssetDatabase (база даних ресурсів гри), UserPrefs (вподобання гравця), та різноманітні внутрішні компоненти, такі як ScriptableObject та Prefab. Далі в розділі детальніше обговоримо кожен з цих компонентів, та перспективи їх використання в проекті, але також слід зазначити можливість впровадження самописних рішень для зберігання даних, що також буде розглянуто.

База даних ресурсів гри, це внутрішній механізм рушія Unity, для збереження та роботи з різними імпортованими або створеними за допомогою редактора Unity ресурсами. Це можуть бути будь які ресурси, починаючи від текстур і матеріалів та закінчуючи 3д моделями і файлами даних. Також рушій надає інструменти для взаємодії цією базою даних, використовуючи наприклад шлях до конкретних

елементів чи їх унікальні ідентифікатори. Це може бути нам дуже корисним особливо у поєднанні зі ScriptableObject.

ScriptableObject це одна з найбільш розповсюджених дійових осіб рушія Unity. Це зазвичай не великий файл, що є ресурсом гри і містить в собі якусь інформацію. Для використання цієї інформації створюється клас, що наслідує ScriptableObject (і таким чином отримує функціонал з читання / запису в цей файл), що також може містити невеликий функціонал, що зазвичай стосується обробки вмісту. Далі за посиланням на створений ресурс можна отримувати дані з цього файлу. При цьому слід мати на увазі, що всі зміни що вносяться до цього ресурсу, мають силу лише під час виконання програми, при повторному запуску, він буде містити останні дані, що були туди внесені зсередини редактора Unity. Тож найбільше такі структури підходять для зберігання статичних конфігураційних файлів, а для більш складних випадків потребують додаткових обгорток.

Також цей тип зручно використовувати коли присутній один екземпляр, і усюди де він використовується є одне і те ж посилання. Але як щодо ситуації коли є деяка кількість екземплярів, посилання на які можуть змінюватись системою під час виконання. В такому випадку зручно було б мати стандартизований інтерфейс доступу до колекцій таких елементів.

Для цього слід реалізувати концепцію регістрів. Це буквально ще один екземпляр ScriptableObject що матиме в собі список посилань на той тип ресурсів що нам необхідно, а також міститиме базовий функціонал з надання доступу до свого вмісту. Також було б зручно інтегрувати в такого роду систему елементи що використовуються в базі даних ресурсів, зокрема ідентифікатори. Це дозволить не пов’язаним системам зберігати можливість отримати необхідний елемент без необхідності зберігати на нього пряме посилання та зі збереженням цієї можливості при перезапуску програми.

Тож в результаті ми отримуємо таку структуру, яка вже неодноразово використовувалась в попередньому розділі (див. рис. 3.6).

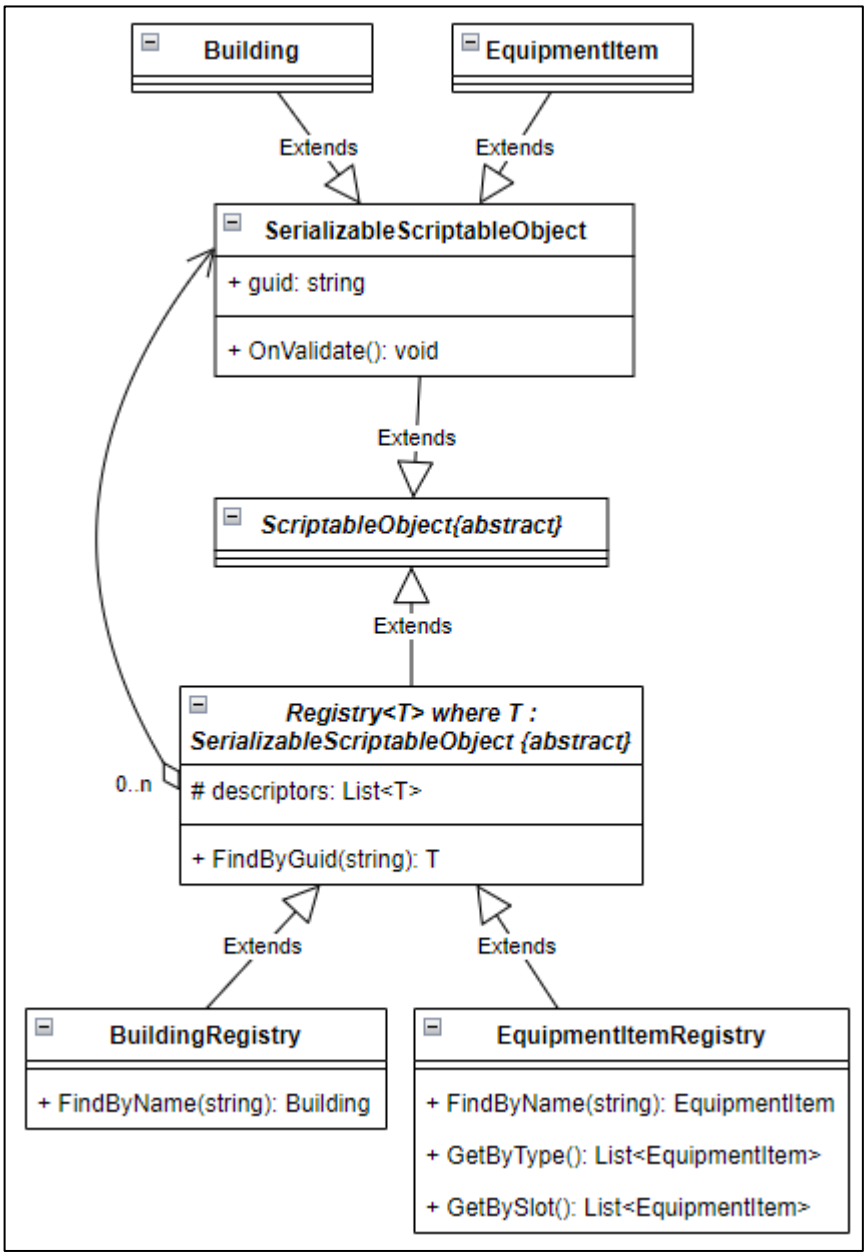


Рисунок 3.6 - Діаграма класів реалізації регістрів (власний рисунок)

На рисунку 3.6 представлено схему реалізації регістрів для файлів з інформацією про будинки та предмети інвентаря, але за подібною схемою можна побудувати регістри для будь яких структур даних та далі розширювати їх.

Наступним пунктом для організації зберігання даних є PlayerPref – платформозалежний регістр даних, що дозволяє зберігати у форматі словника з рядковим ключем числа з плаваючою точкою, цілі числа та рядкові значення. Такий тип зберігання часто використовується для збереження між сесіями невеликої кількості інформації наприклад стосовно налаштувань гри. Цей механізм є

повноцінно реалізованим, тож детально розглядати його не має сенсу, враховуючи також що цікавих випадків його використання в проекті не передбачається.

І нарешті переходимо до останнього пункту щодо збереження даних, це самописні рішення. Для полегшення сприйняття, почнемо їх розгляд одразу з прикладу компонентів майбутньої системи (див. рис. 3.7).

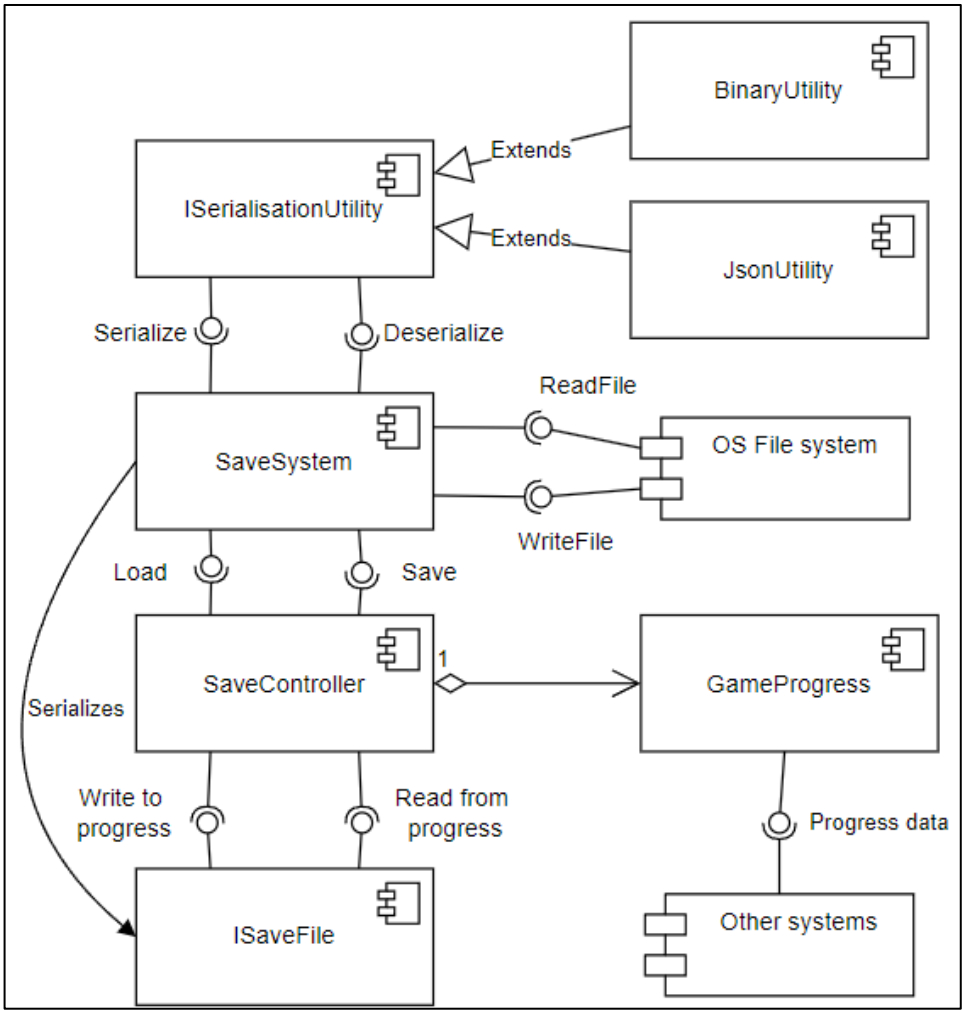


Рисунок 3.7 - Діаграма компонентів системи збереження даних (власний рисунок)

Почнемо розбирати цей прототип системи збереження даних з визначення того, що ми від неї потребуємо. Так як гра планується для персональних комп’ютерів та не є мережевою, то дані користувачів можна зберігати на їх пристроях користуючись файловою системою. Формат файлів має бути захищеним від прямих втручань користувача, тож це має бути бінарний файл, але в той же час було б зручно мати змогу під час розробки втручатись у вміст файлів збереження

та проводити додаткові перевірки за потребою. Для цього в процесі серіалізації слід абстрагуватись від конкретної реалізації та використовувати інтерфейс, для можливості зміни типу вмісту файлів.

Також основний контроль за загальним процесом збереження та завантаження даних слід винести в окремий контролер, як і сам процес обробки таких запитів та роботи з файловою системою. Для цього створено SaveController та SaveSystem компоненти відповідно.

Для використання даних отриманих з файлу використовуватимемо ScriptableObject під назвою GameProgress, в якому записуватимемо під час роботи програми дані зчитані з файлів збереження.

Далі нам необхідний об’єкт передачі даних, який міг би містити дані в форматі пригодному для серіалізації та запису у GameProgress. Для збільшення гнучкості створимо інтерфейс, що забезпечуватиме необхідні функції для взаємодії з класом прогресу.

На цьому усі необхідні проекту ситуації пов’язані зі збереженням даних покриті описаними в цьому розділі інструментами.

3.4 Приклади найцікавіших алгоритмів

3.4.1 Використання event-driven підходу

Часто в процесі роботи системи виникають ситуації, стан одного її компонента, має викликати зміни або якісь дії інших компонентів. Найпростішим виходом є проектування системи з використанням прямих посилань на ті чи інші компоненти, що мають реагувати. Але в такому випадку ми стикаємося з проблемою не гнучкого коду, бо коли нам необхідно додати або прибрати якісь з реактивних компонентів, нам прийдеться модифікувати існуючий код, що забирає час та несе ризики пов’язані з сумісністю.

Тому в таких випадках доречно використовувати підхід, що ґрунтується на використанні подій. В цьому випадку один з акторів запускає подію в залежності від інкапсульованих в ньому умов, а інші реактивні актори підписуються на ті чи інші події, і оброблюють їх у свій спосіб. Таким чином ми позбуваємось описаної вище проблеми, адже для додавання чи вилучення слухачів, слід лише підписатись або відписатись відповідно від необхідних подій.

Для реалізації подібних підходів можна використати різні рішення, першим найпростішим є використання Unity Event System, що несе в собі інструменти у вигляді подій, на які можна підписати ті чи інші функції які відповідають сигнатурі і вони будуть викликатись при активації події. Плюсом цього підходу є простота, адже нічого не потрібно реалізовувати, а лише додати відповідне поле до необхідного компоненту, а також зручність використання в середині редактора рушія, адже для цього вже реалізований інтерфейс. Але мінусом є те що в такому випадку ми не позбавляємось прямих посилань на компоненти-слухачі, адже функції слід підписувати напряму і в одному місці, що також не завжди може бути реалізовано. Отже цей підхід може використовуватись в простих казуальних проектах, де немає багатьох дійових осіб та можливі прямі посилання, що не є актуальним для гри що розглядається.

Ще одним рішенням є написання своїх рішень, без використання інструментів рушія. Наприклад це може бути статичний клас, в якому інкапсульовано як систему підписок так і активації подій, через який актори зможуть спілкуватись за допомогою подій. Такий підхід вирішує проблеми попереднього прикладу, зокрема потребу в прямих посиланнях, адже в цьому випадку існує проксі, через який виконуються дії, і який зберігає усі посилання, також в цьому випадку можливо реалізувати систему з динамічною передачею параметрів, що неможливо в попередньому варіанті. Але присутні і мінуси, для зручного використання такого підходу, слід знову ж таки використовувати Unity Event System, адже підписуватись на події можна лише зі скриптів, а це в свою чергу забирає у нас можливість формування динамічних параметрів, або ж розроблювати складні самописні рішення, що включатимуть розробку інтерфейсів

для редактора рушія, що потребує необхідної кваліфікації. Також існує ще одна проблема – формат зберігання посилань та подій, адже тут кожне рішення балансує між зручністю та простотою використання, і ідеальних рішень немає.

Але є ще один підхід, який і планується реалізувати в грі. Це використання системи івентів на базі ScriptableObject, що також частково включає в себе використання попередніх підходів. Основною концепцією є припущення про те що подія, є скриптовим об’єктом, тобто будь хто, будь те може отримати на нього посилання, а також існує сутність слухача подій як компонента що містить в собі подію Unity (див. рис. 3.8).

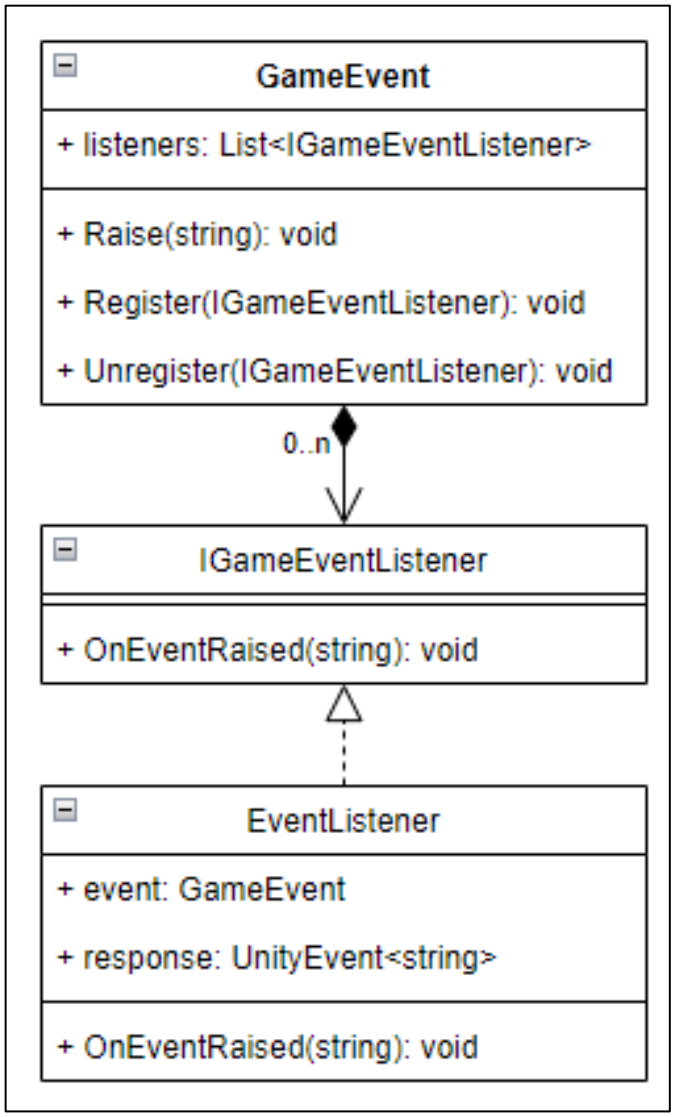


Рисунок 3.8 - Діаграма класів системи подій (власний рисунок)

Таким чином, ми суміщаємо попередні приклади, використовуємо вбудовані події рушія, за їх зручність у використанні в редакторі, використовуємо проксі у вигляді скриптового об’єкта для запобігання використання прямих посилань, а також за допомогою компонентів-слухачів можемо зручно підписуватись та відписуватись на івенти за допомогою середовища розробки. Але не без проблем і в цьому випадку, ми все ще обмежені у передачі параметрів, адже передаємо лише рядок, що можна вирішити за допомогою використання або скриптових об’єктів як контейнерів параметрів, або передавати параметри у серіалізованому вигляді, наприклад у форматі JSON.

Розглянемо алгоритм роботи компонентів системи з використанням описаного вище підходу (див. рис. 3.9).

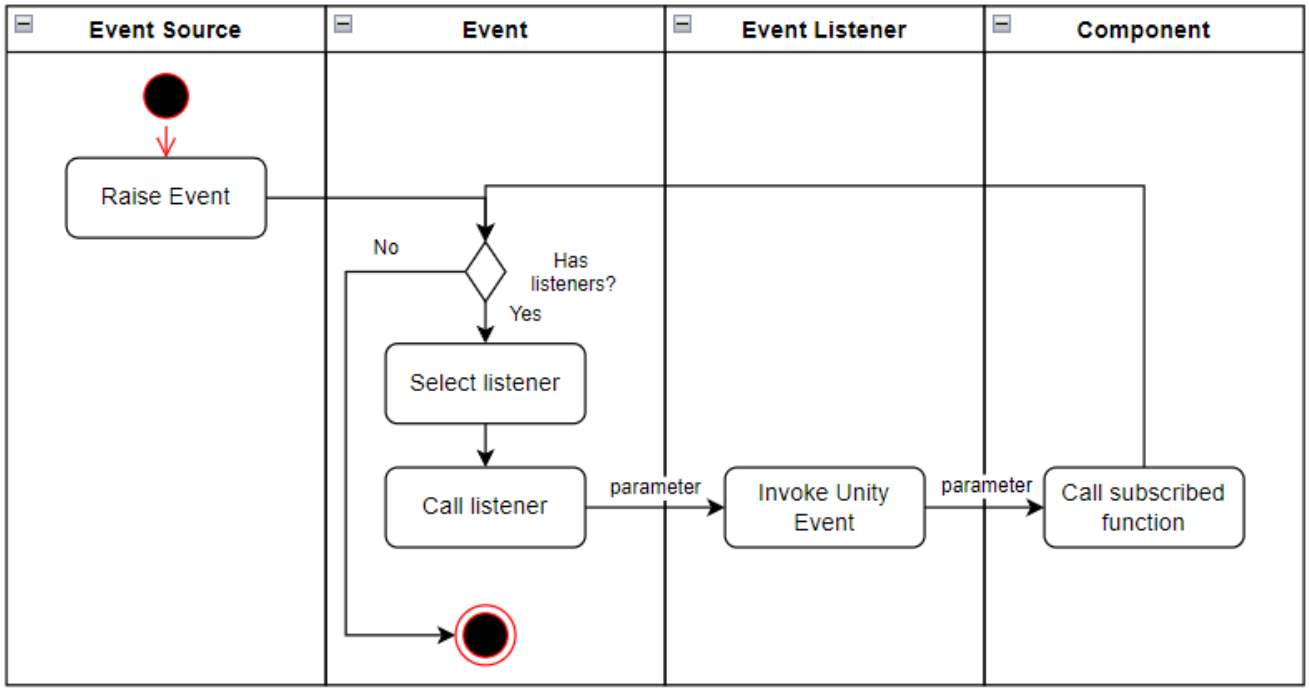


Рисунок 3.9 - Діаграма активності використання системи подій (власний рисунок)

Таким чином ми робимо розроблену архітектуру більш масштабованою та зручною в підтримці та розробці, особливо враховуючи необхідність частого розширення та перманентного наповнення контентом в процесі розробки.

3.4.2 Модифікатори

Концепція модифікаторів є невід’ємною частиною системи характеристик та ресурсів, і містить в собі цікавий підхід для розрахунків фінальних показників або нарахувань.

Як вже зазначалось раніше, модифікатори це інструменти впливу на показники характеристик та ресурсів ззовні, для перших це їх зміна під час виконання під впливом ігроладних процесів (використання бонусів, спорядження тощо), а для ресурсів це модифікатори їх отримання та використання. Хоча цілі модифікаторів відрізняються, але алгоритм та принципи їх впливу залишаються не змінними. Саме ці принципи ми і розглянемо більш детально.

Але перед тим як перейдемо до розгляду, слід звернути увагу на дві основні концепції модифікаторів, порядок та тип накладання. Перша визначає вагу, в залежності від якої визначається порядок накладання модифікаторів, що дозволяє керувати кінцевою формулою. Це є ключовим у поєднанні з типом накладання, яких існує 3:

– просте додавання (додавання визначеного показника до початкового);
– простий коефіцієнт (додавання до початкового значення результату множення початкового показника на коефіцієнт);
– коефіцієнт з додаванням (додавання до початкового значення результату множення початкового показника на кінцевий показник, значення якого формується з суми коефіцієнтів що йдуть по порядку).

Для кращого розуміння звернемо увагу на приклад що розміщений на рисунку 3.10. Тут ми маємо три різні приклади на яких можна побачити як вага визначає порядок накладання модифікаторів та який вплив кожен з їх типів має на кінцевий результат. Слід також зазначити що в кожному з прикладів використовуються модифікатори з абсолютно однаковими значеннями, і накладаються вони на однакове початкове значення, але визначений порядок та тип

має неабиякий вплив на кінцевий результат розрахунків, що чітко видно при порівнянні значень.

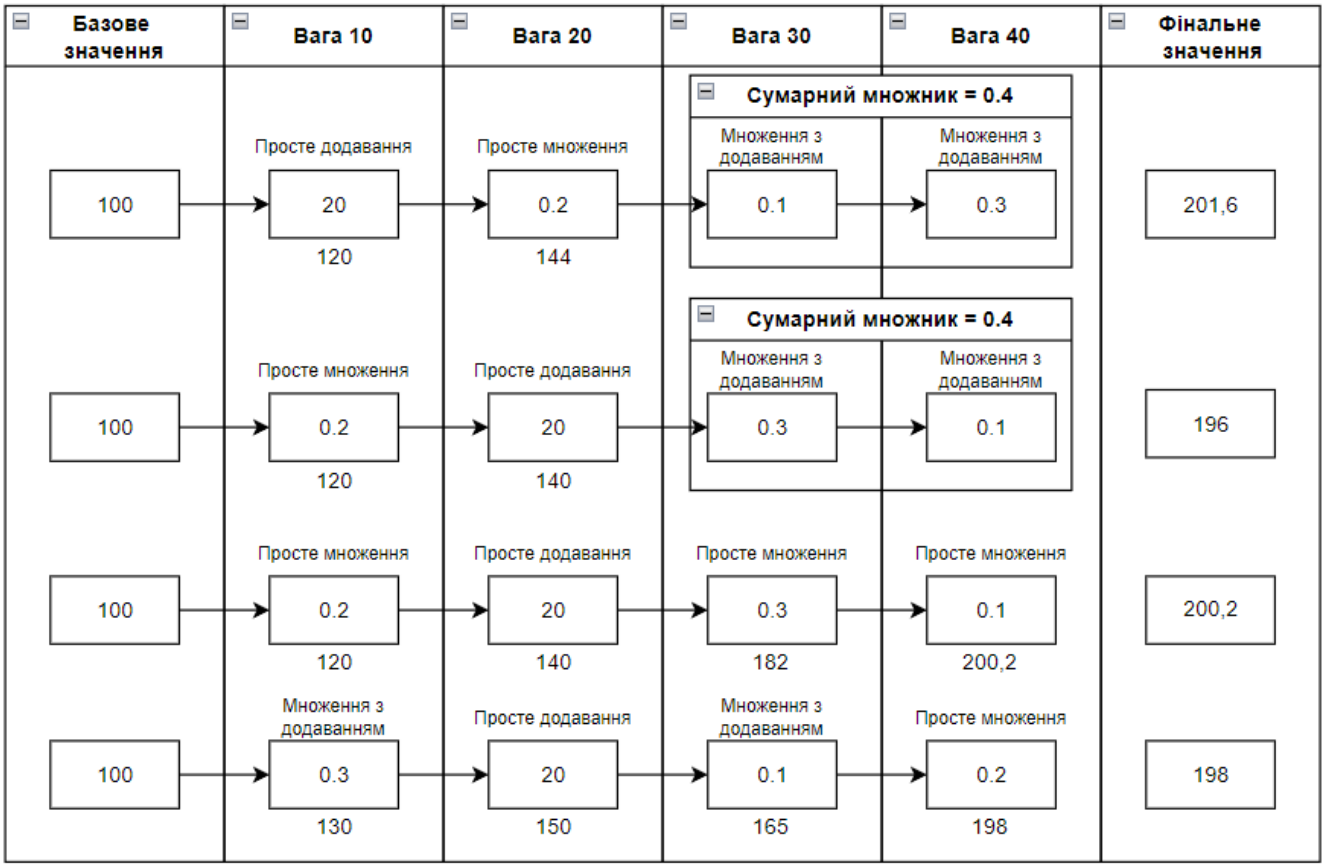


Рисунок 3.10 – Приклади накладання модифікаторів (власний рисунок)

В перший і другий приклади на рисунку вище відрізняються лише тим що попарно змінені місцями модифікатори 1, 2 та 3, 4, причому результат накладання другої пари не змінюється через особливості їх типу, але результати є різними.

При порівнянні другого та третього прикладів, бачимо що показники модифікаторів множення не змінились, але обидва що мали тип множення з додаванням змінили його на просте множення, і результуюче значення знову було змінене.

І останній приклад призваний показати те, що при розриві множників з додаванням вони фактично поводяться як звичайні множники, і ще раз підкреслити вплив порядку на розрахунок.

Таким чином корегуючи ці два показники, ми можемо формувати складні формули в залежності від ситуації. Тож перейдемо до проектування алгоритму, який би забезпечив нам можливість використання такого підходу (див. рис. 3.11).

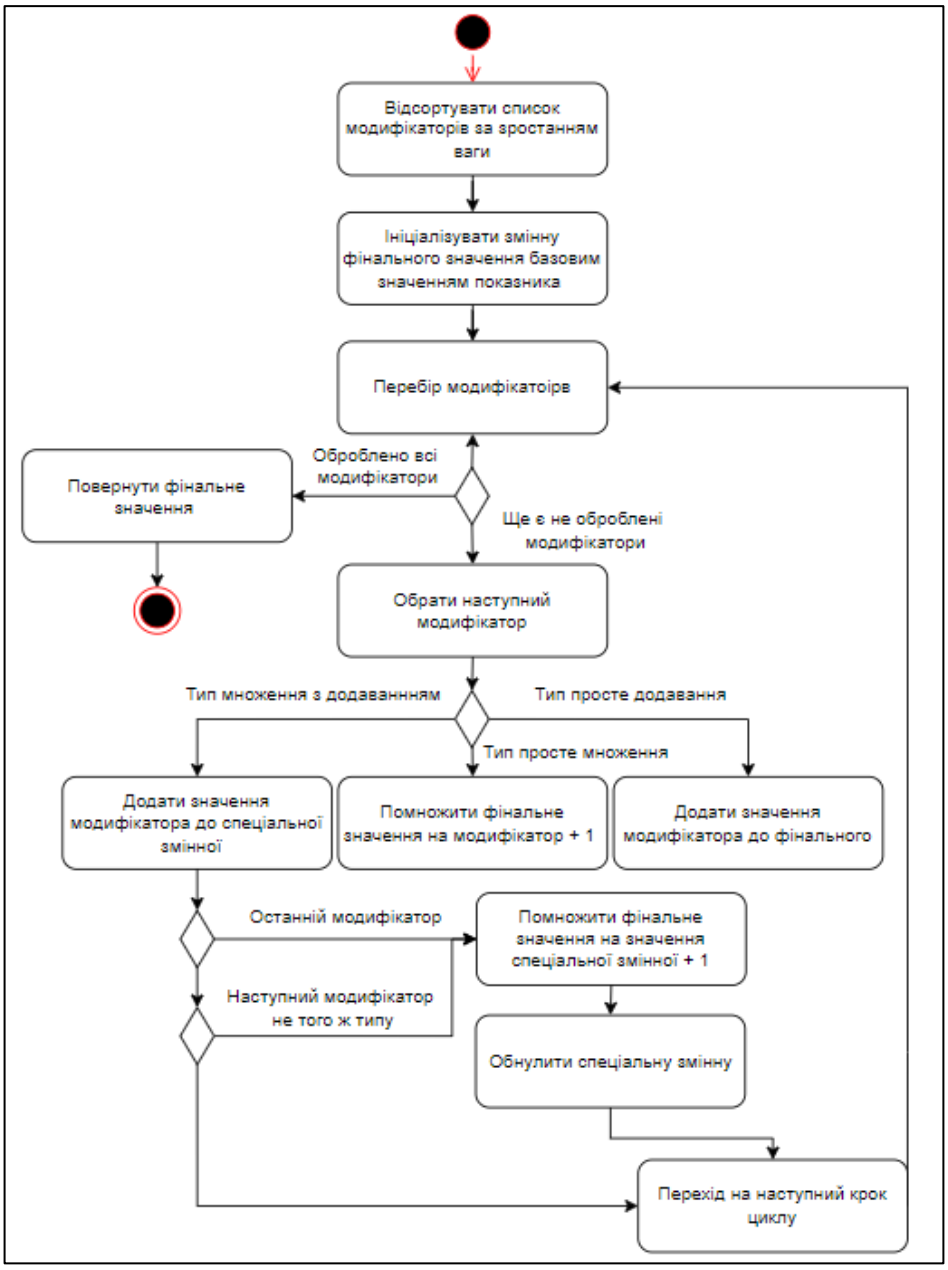


Рисунок 3.11 – Діаграма активності алгоритму накладання модифікаторів (власний рисунок)

За допомогою описаного вище алгоритму, ми зможемо виконати усі поставлені перед системою задачі, що включають накладання модифікаторів, зокрема в системах характеристик та ресурсів.

3.5 Проектування UI / UX

Частина функціоналу що розглядається в цій роботі надзвичайно сильно залежить від користувацького інтерфейсу, адже більшість механік не використовують тривимірні моделі, а цілком спитаються на стандартні інтерфейси. Тому на моменті проектування слід розробити макети користувацьких інтерфейсів та екранних форм, які будуть впроваджені в гру пізніше. Для цього скористаємось спеціалізованим програмним забезпеченням Figma та приблизно спроектуємо майбутній інтерфейс гри.

Далі розглянемо та проаналізуємо найбільш цікаві екранні форми що були спроектовані. Для початку розглянемо інтерфейс головного екрану поселення (див. рис. 3.12).

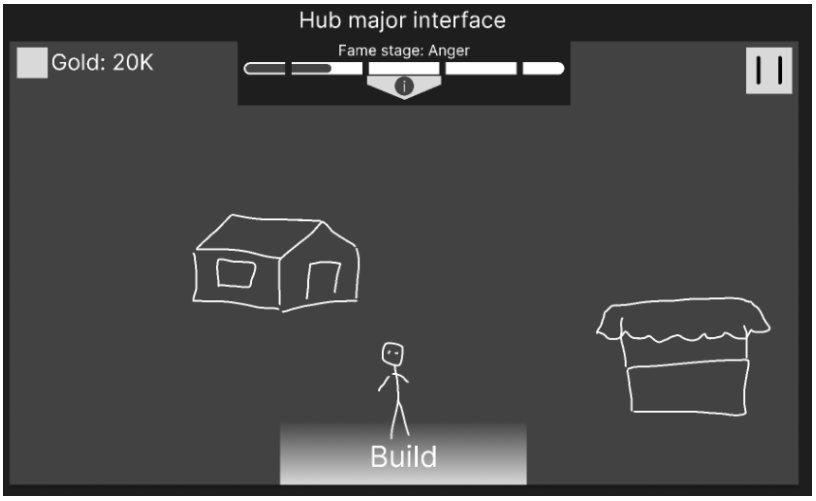


Рисунок 3.12 – Прототип головного інтерфейсу поселення (власний рисунок)

На цьому рисунку присутні основні керуючі та інформаційні елементи, цієї екранної форми, це показники прогресу гри, та кнопки переходу в меню налаштувань та режим будівництва. Також більшість будівель, що розташовуватимуться на карті і доступні для перегляду гравцем мають бути інтерактивними та відповідати за переходи до інших вікон.

Далі розглянемо вигляд цього ж вікна при натисканні відповідної кнопки та переходу в режим будівництва (див. рис. 3.13).

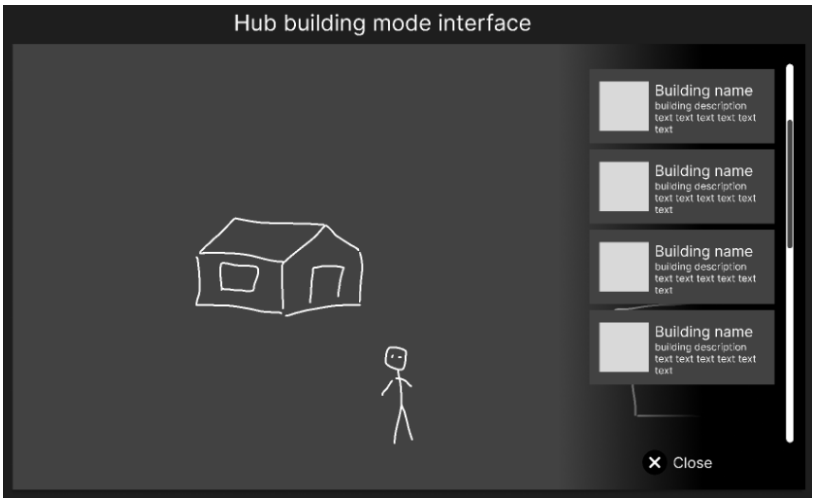


Рисунок 3.13 – Прототип інтерфейсу в режимі будівництва (власний рисунок)

В цьому стані з правого боку екрану з’являється панель що відповідає за відображення списку будівель, а увесь інтерфейс попереднього вікна зникає. Кожен елемент у списку будівель має відображати єдину будівлю з її описанням, назвою та списком необхідних ресурсів. Також кожен елемент списку має відображати стан будівлі (побудовано, можна розмістити, не доступно або не вистачає ресурсів). Після натискання на елемент на місцях для розміщення будівель на які наводиться курсор, має показуватись модель будівлі.

Далі перейдемо до головного інтерфейсу таверни (див. рис. 3.14). Перейти до нього можна буде після взаємодії з 3д моделлю таверни що розташована на карті. В цьому вікні міститимуться дві кнопки для переходу на сторінки з інвентарем та управлінням ресурсами, а також індикатори відносин з кожним з 5 народів світу. Ці індикатори мають містити кожен свою шкалу прогресу, рівень якої відповідатиме за рівень відносин з народом, де максимальна заповненість ідентифікуватиме максимальну інтеграцію в життя народу. Також ці індикатори відіграватимуть роль кнопок, які вестимуть на відповідні сторінки з деревами вмінь народів.

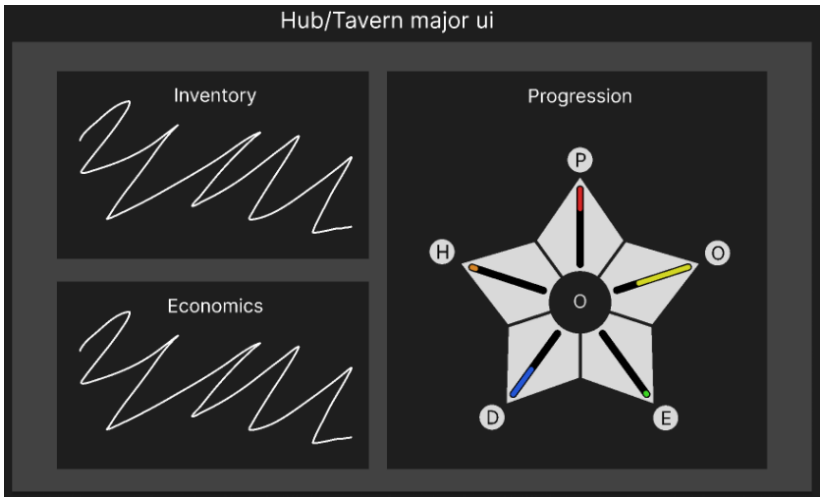


Рисунок 3.14 – Прототип головного інтерфейсу таверни (власний рисунок)

Далі розглянемо приблизний інтерфейс вікна дерева вмінь (див. рис. 3.15)

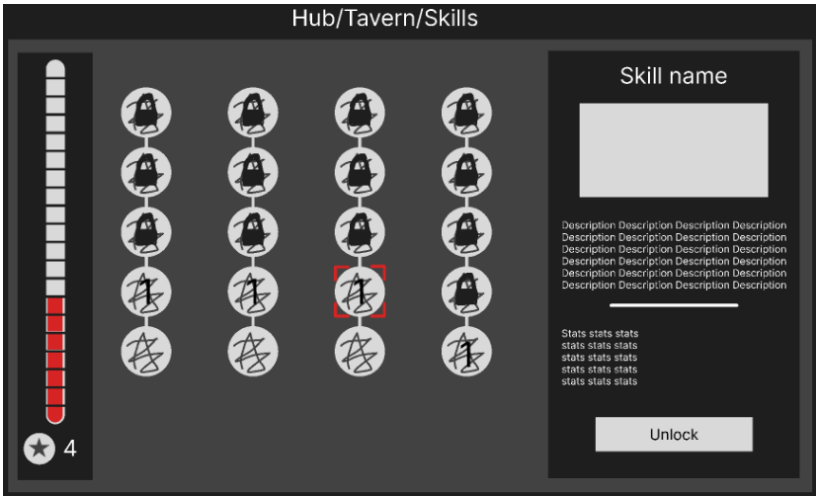


Рисунок 3.15 – Приблизний прототип інтерфейсу дерева вмінь (власний рисунок)

На прототипі цього інтерфейсу видно, що вікно умовно поділене на 3 частини. Ліворуч знаходиться шкала прогресу, яка розділена на 24 пункти, кожен з яких індикує про отримання гравцем одного очка навичок. Під шкалою знаходиться показник кількості вільних очок. Центральну частину екрану має займати так зване «дерево вмінь» з 24 вузлів, кожен з яких відображує одне з умінь, а також його стан (досліджено, доступно, недоступно, обрано). Ці 24 вузли мають бути розподілені за 3 рівнями розвитку (по 8 на кожному), і розблоковуватись по ходу витрати очок. При натисканні на зображення кожного з вмінь, в правій панелі

має відображатись описання цього вміння, його назва та список пов’язаних вмінь необхідних для розблокування поточного, та кнопка для розблокування.

Ще одним важливим інтерфейсом є вікно інвентаря (див. рис. 3.16)

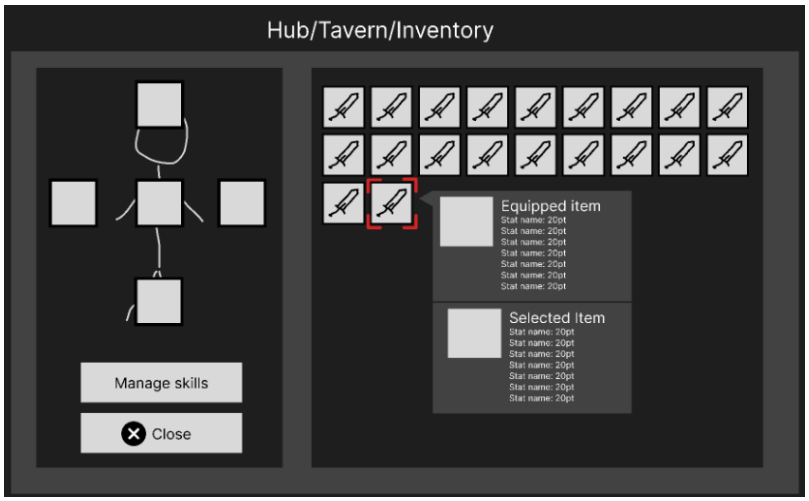


Рисунок 3.16 – Прототип головного інтерфейсу таверни (власний рисунок)

В цьому вікні розташовано усі керівні елементи для екіпірування та управління інвентарем. Зокрема ліворуч, розташовується панель в якій знаходяться комірки для використання різних предметів. Для того щоб вдягнути або замінити щось, необхідно перетягнути необхідний предмет в відповідну комірку. В панелі праворуч знаходяться елементи керування для відображення списку предметів що знаходяться у інвентарі користувача, при наведенні на які з’являється віконце з підказкою про його характеристики, а якщо в екіпіруванні присутній альтернативний предмет, то його характеристики показуються поруч. Також за допомогою перетягування предметів в зони що мають з’являтись в нижній частині списку предметів інвентаря їх має бути змога продати або розібрати.

І останнім інтерфейсом який слід розглянути є вікно управління ресурсами. Це один з найважливіших компонентів економічних механік гри, на який покладається велика кількість функціоналу, це зокрема перегляд наявних та ресурсів відстеження та аналітика процесів витрати та видобутку ресурсів, а також швидкий спосіб отримати доступ до функціоналу ринку ресурсів, щоб мати змогу налагоджувати щотижневі (в рамках ігрового світу) поставки з імпорту та експорту

ресурсів. Тож так як тут гравець може проводити велику кількість часу зосередимось на проектуванні цього вікна (див. рис. 3.17).

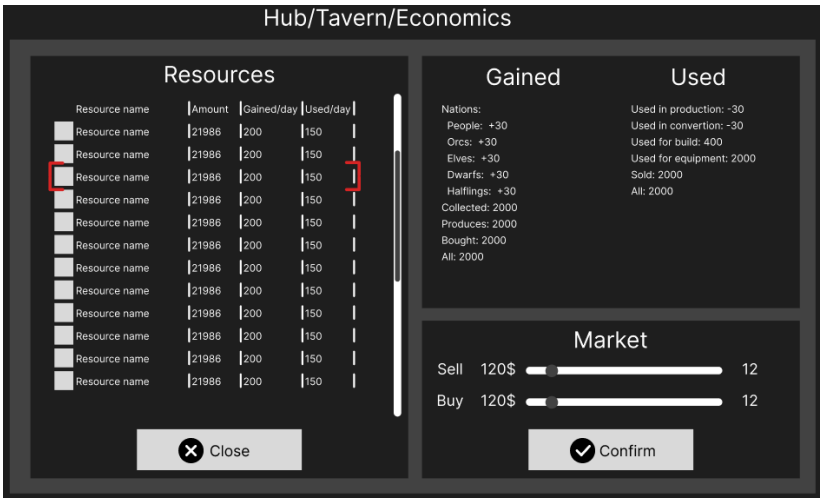


Рисунок 3.17 – Прототип інтерфейсу вікна економіки (власний рисунок)

Це вікно умовно можна розділити на 3 основні частини, відносно 3 основних задач. Всю ліву половину займатиме список ресурсів, де буде міститись їх зображення, назва, поточна кількість та динаміка використання та отримання в день. Верхню частину правої половини екрану має займати детальна статистика щодо отримання та використання ресурсів, з розбивкою по джерелам. Ці дані мають показуватись та змінюватись при виборі відповідного ресурсу з панелі ліворуч. І наприкінці, нижню частину правої половини екрану займатиме компактна панель ринку, де можна буде побачити яку кількість обраного ресурсу гравець на даний момент експортує, а яку імпортує, з можливістю відредагувати необхідні показники за допомогою повзунків.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Розробка головної сцени

Після детального етапу проектування можемо приступати до процесу розробки програмного продукту. Для початку звернемо свою увагу на головну сцену що містить у собі основний функціонал управління поселенням та розвитком персонажа.

Внутрішній устрій будь якої сцени в рушії Unity є формалізованою ієрархічною структурою де кожен об’єкт є екземпляром GameObject. Поведінка або стани кожного об’єкта визначаються компонентами (скриптами) які прикріплені до нього. Тож для формування необхідної нам сцени сформуємо попередню ієрархію, яку в подальшому наповнимо необхідним нам функціоналом (див. рис. 4.1).

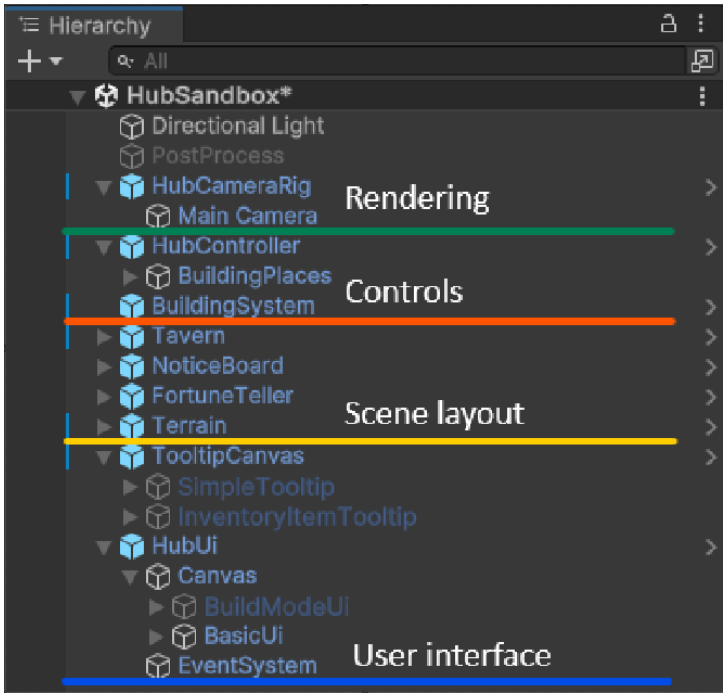


Рисунок 4.1 – Ієрархія сцени поселення в рушії Unity (власний рисунок)

На попередньому рисунку можна побачити об’єкти високого рівня в ієрархії сцени поселення розроблюваної гри. Також їх умовно розділено на 4 групи за їх призначенням, далі більш детально про кожну з них.

Група об’єктів рендерингу (на рисунку 4.1 підкреслено зеленим) це кілька важливих акторів, що відповідають за рендеринг сцени. Здебільшого це об’єкти з вбудованими компонентами Unity, такими як глобальне джерело світла, камера та область пост-обробки. Але також тут присутні і скрипти власної розробки, наприклад компонент керування камерою, що відповідно до команд користувача дозволяє переміщувати камеру сценою (див. рис. 4.2).

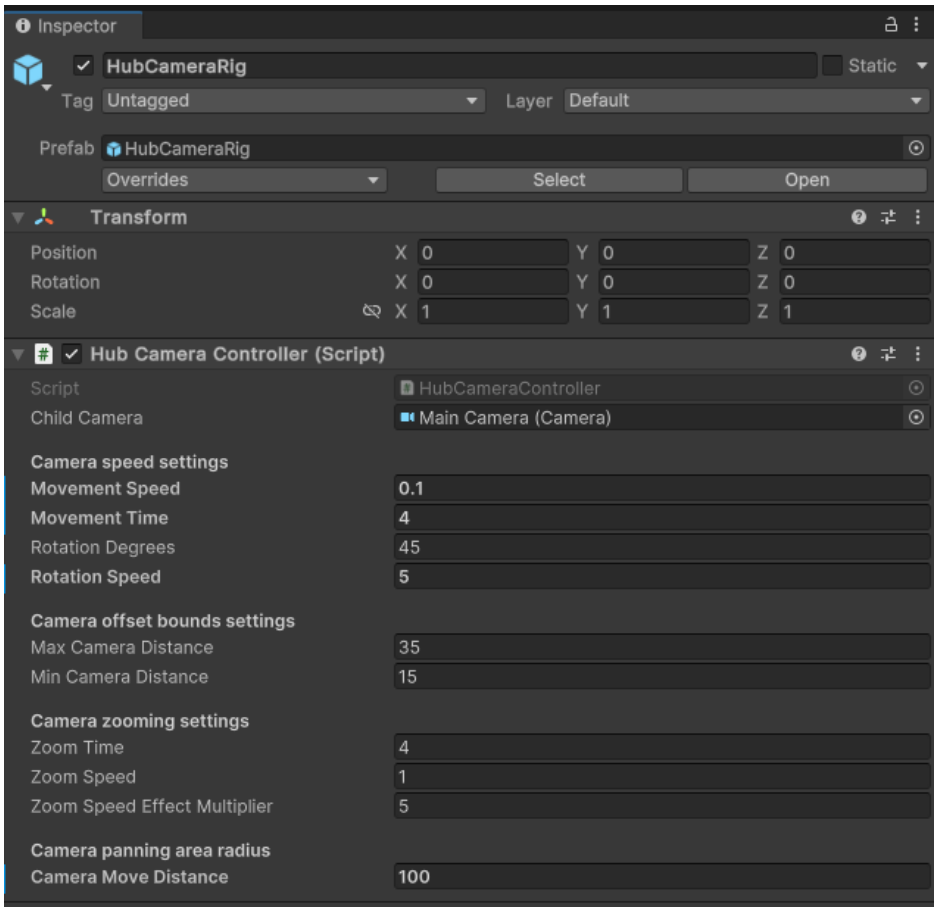


Рисунок 4.2 – Об’єкт HubCameraRig з компонентом управління камерою (власний рисунок)

Цей компонент дозволяє переміщувати камеру по локації, повертати та приближувати її, а також надає інтерфейс для корегування параметрів з середини редактора рушія. Таким чином ми можемо корегувати швидкість та інерцію переміщень та поворотів камери, рамки та параметри швидкості та сили її

приближення та віддалення, а також контролювати зону в якій камера може рухатись.

Наступною групою є контролюючі об’єкти (на рисунку 4.1 підкреслено червоним), це в основному актори з самописною поведінкою які відповідають за глобальні процеси на сцені, такі як команди щодо потоку виконання та процеси будівництва та управління поселенням. Найбільш цікавим є об’єкт системи будівництва, що містить в собі низку компонентів відповідно до описаної у розділі 3.2 архітектури (див. рис. 4.3).

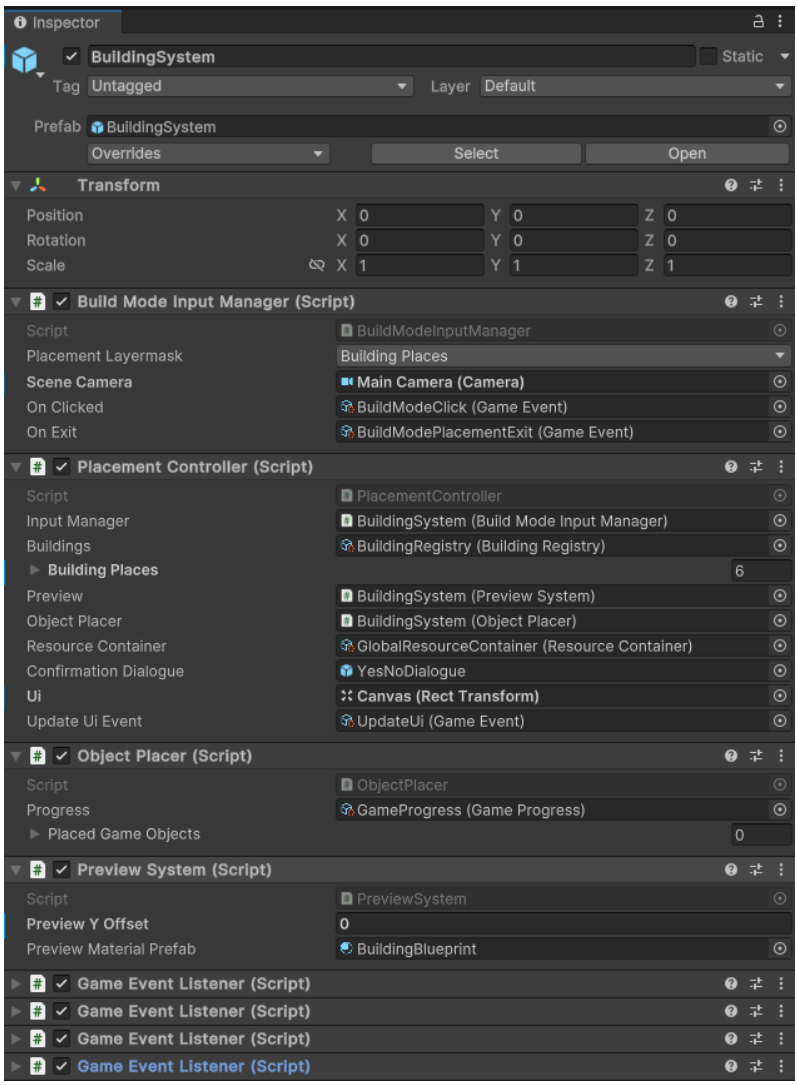


Рисунок 4.3 – Список компонентів об’єкта BuildSystem (власний рисунок)

Цей об’єкт містить низку компонентів що відповідають за опрацювання дій користувача, контроль розміщення об’єктів та загальних процесів управління

будівлями, безпосереднє розміщення будівель, систему передпоказу дій та відслідковування подій (відповідно до порядку розміщення компонентів на рисунку 4.3 починаючи з InputManager). Подивимось на вміст класу компоненту на прикладі розміщувача об’єктів.

```
public class ObjectPlacer : MonoBehaviour
{
    public GameProgress progress;
    public List<GameObject> placedGameObjects = new();

    public int PlaceObject(Building building, BuildingPlace place)
    {
        GameObject newObject = Instantiate(building.prefab);
        newObject.transform.position = place.transform.position;
        newObject.transform.rotation = place.transform.rotation;
        placedGameObjects.Add(newObject);

        place.SetBuilding(building, placedGameObjects.Count - 1);

        newObject.GetComponent<BuildingController>().isBuilt = true;

        progress.AddBuilding(building);

        return placedGameObjects.Count - 1;
    }

    public void RemoveObjectAt(BuildingPlace place)
    {
        int idx = place.BuildingIndexInPlacer;

        if (placedGameObjects.Count <= idx
            || idx < 0
            || placedGameObjects[idx] == null)
        {
            return;
        }

        progress.RemoveBuilding(place.Building);

        Destroy(placedGameObjects[idx]);
        placedGameObjects[idx] = null;

        place.RemoveBuilding();
    }
}
```

Цей невеличкий компонент є звичайним C# класом що розширює вбудований клас рушія MonoBehaviour для використання як компонента. Тут містяться два публічних поля через які ми ззовні можемо отримати інформацію щодо розміщених

об’єктів та оновлювати стани інших систем гри. Також тут присутні дві функції, що відповідають за розміщення та знесення будівель і використовуються як в середині системи будівництва так і зовнішніми системами, наприклад для завантаження прогресу.

Наступною групою об’єктів сцени є візуальні 3д об’єкти що формують її зовнішній вигляд (на рисунку 4.1 підкреслені жовтим). Тут зазвичай присутні 2 види акторів, звичайні 3д об’єкти такі як декорації, загальна карта, тощо, тобто такі що не мають поведінки, а також більш комплексні об’єкти головною ціллю яких є все ще декорування сцени але присутні додаткові обов’язки та функціонал. Такими наприклад можуть бути будівлі, які здебільшого просто розміщені на локації але і мають функціонал щодо взаємодії.

І останньою групою об’єктів на сцені є користувацький інтерфейс. Це складна ієрархічна структура елементів що відповідає за відображення різноманітних візуальних елементів від кнопок та тексту до вікон на екрані користувача. Більшість функціоналу щодо відображення взаємодії та низькорівневих процедур тут є вже реалізованою в середині рушія, але все ще майже більшість акторів цієї групи мають особливий функціонал описаний власними скриптами, що формують складні підсистеми. Більш детально про цю групу об’єктів поговоримо в наступних розділах окремо.

4.2 Особливості впровадження не інстанційованих систем

У попередньому розділі ми розглянули вміст нашої головної сцени, розібрали його за завданням, впровадженням та реалізацією. Але це не повний обсяг того, що використовується для її функціонування. Інтерфейси до більшості систем гри реалізовані за допомогою скриптових об’єктів і їх екземпляри створюються на льоту програмою тоді і у тому місці де в них є потреба, базуючись на стані цих систем, що зберігається у ресурсах гри.

Прикладом такого компоненту програми є система ресурсів, доступ до якої здійснюється через посилання на файл ресурсів (Asset) типу ResourceContainer, більш детально код якого можна подивитись в додатку Г. Цей клас представляє собою набір ресурсів в різних контекстах гри, тобто це може бути як список усіх ресурсів гравця, так і тимчасовий інвентар з ресурсами при дослідженні підземель або вміст джерела ресурсів. Але окрім ресурсів контейнер також містить низку функцій що здатні змінювати його стан, це маніпуляції над ресурсами, різноманітні перевірки або формування статистик. Тож як можна побачити з описання, цей контейнер – не простий об’єкт що живе в контексті однієї сцени, а складна концепція що може бути використана у будь якому куточку гри. Тож цей компонент системи ресурсів спроектовано як скриптовий об’єкт, що зберігає свій стан в процесі виконання і може бути переданий за прямим посиланням на файл відповідного ресурсу, що не лише зручно, а й безпечно в плані залежностей для передачі даних між сценами.

Наведемо приклад використання цього класу.

```
private void ActionBase(Vector3 position)
{
    var place = buildingPlaces.Find(p =>
        p.transform.position == position);
    int index = objectPlacer.PlaceObject(building, place);

    previewSystem.UpdatePosition(position, false);

    resourceContainer.Spend(building.price);
}
```

Наведена функція є частиною реалізації інтерфейсу стратегії режиму будівництва, а саме є кінцевою дією щодо розміщення будівлі. В цій функції ми знаходимо місце для розміщення будівлі на яке вказав користувач, розміщуємо на ньому будівлю, вимикаємо систему передпоказу та списуємо ресурси з загального контейнера ресурсів користувача. При цьому ми використовуємо функцію списання зі спрощеною сигнатурою, що приймає екземпляр класу Price що міститься в описанні кожної будівлі. Таким чином реалізована система списання

ресурсів при будівництві, при чому, завдяки описаній у попередньому розділі системі, на цю ціну також впливають накладені модифікатори. І для того щоб отримати посилання на екземпляр типу контейнера не потрібно проводити жодних пошуків, чи звертатись до залежностей, достатньо обрати необхідний ассет в інспекторі та надати його необхідному компоненту.

Зовсім інша ситуація це переробка ресурсів, яка все ще використовує контейнери. Далі наведено частину коду функції обробки процесу виробництва ресурсів.

```
if (resources.CanAfford(totalPrice))
{
    var record = new Dictionary<ResourceName, ResourceDynamic>();
    resources.Spend(totalPrice);

    foreach (var resource in totalPrice.resources)
        record[resource.name] = new ResourceDynamic
            { gained = 0, spent = resource.amount,
                resource = resource.name };

    foreach (var res in item.recipe.resources)
    {
        if (!record.ContainsKey(res.name))
            record[res.name] = new ResourceDynamic
                { gained = 0, spent = 0, resource = res.name };

        var totalAmount = item.recipe
            .TotalAmount(res.name, item.workers);
        resources.GainResource(res.name, totalAmount,
            TransactionType.Production);

        record[res.name].gained += totalAmount;
    }

    log.productionBuldingsLog.Add(new BuildingProductionLogItem
    {
        building = progress.placedBuildings
            .First(b => b.Guid == item.buildingGuid),
        resources = record.Values.ToList(),
    });
}
```

Таким чином контролер обробки подій в поселенні проводить транзакції що відповідають процесу переробки ресурсів що описані рецептами кожної з розміщених будівель-виробництв. В наданій вище функції використовується майже увесь функціонал контейнера, від трекінгу поточного стану до нарахування

та списання коштів. І знову, для використання цього функціоналу, достатньо надати відповідне посилання за допомогою інспектора Unity.

Схожим чином працюють багато систем в грі, зокрема також система характеристик, інвентаря, екіпірування, прогресу та квестова система. Але в той час як цей підхід ідеально підходить для гнучкої передачі даних між компонентами та сценами, він не дозволяє зберігати тяглість між ігровими сесіями. Тож в наступному підрозділі розглянемо реалізацію системи збереження, та те як вона допомагає нам нівелювати цю проблему.

4.3 Реалізація системи передачі даних між сесіями

Основні принципи системи збереження даних описані у попередньому розділі, але якщо просто то це система серіалізації у файл класу що виступає контейнером простих типів даних (логічних, числових, рядкових, їх масивів та структур даних що містять в собі попередньо описане).

Далі для розуміння принципів відновлення стану скриптових об’єктів за допомогою системи збереження слід заглибитись у принципи їх роботи. Скриптові об’єкти мають попередньо визначений у редакторі стан, що завантажується у пам’ять при першому посиланні на об’єкт. Далі під час роботи додатку ці дані можуть змінюватись і гарантовано бути консистентними при використанні в будь яких куточках програми. При завершенні роботи програми та вивантаженні даних з пам’яті усі оновленні що мали місце під час виконання втрачаються, і при наступному завантаженні ми отримуємо ті дані що були визначені в редакторі. Тобто скриптові об’єкти є змінюваними в контексті окремих сесій, але є незмінними в контексті багатьох сесій.

Тож для забезпечення консистентності даних між сесіями слід на початку відновлювати стан кожної системи відповідно до даних що збережені у окремому файлі зберження.

Для спрощення цього процесу, його було централізовано за допомогою ще одного скриптового об’єкта GameProgress, що містить в собі посилання на усі системи або дані, що потребують збереження (див. рис. 4.4).

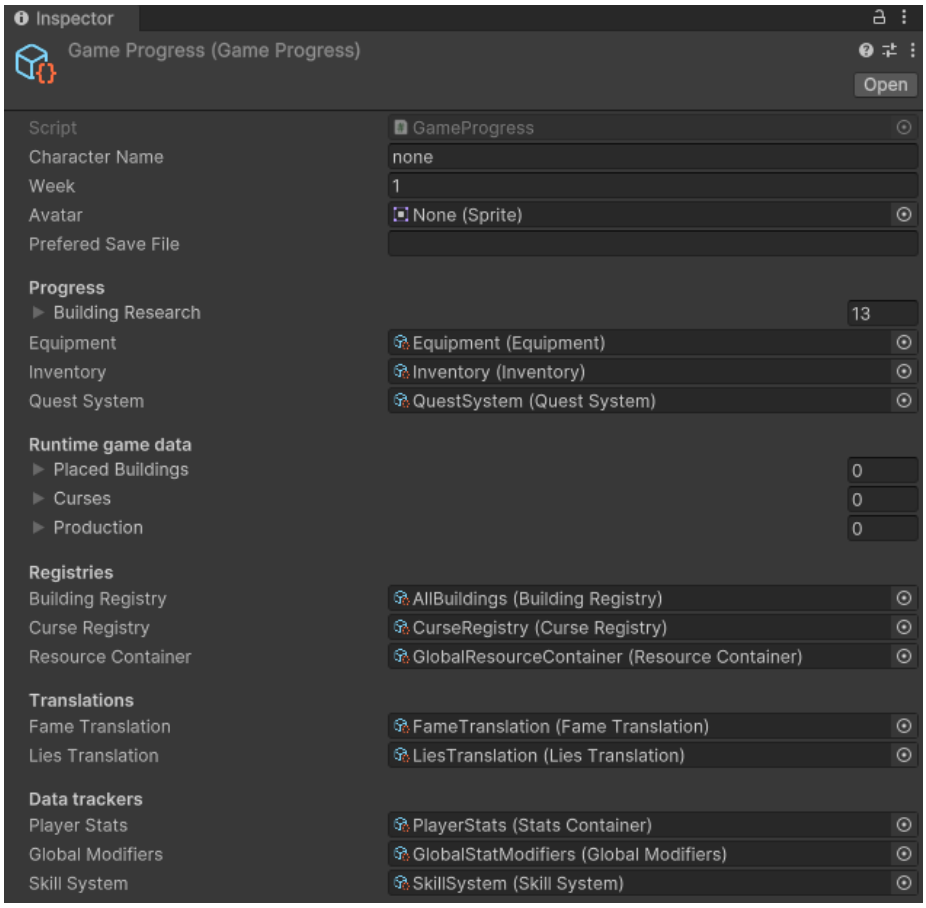


Рисунок 4.4 – Список полів об’єкта GameProgress (власний рисунок)

На попередньому рисунку можна побачити усі основні елементи даних що потребують збереження, але вони не можуть буди збережені за допомогою цього класу, адже він містить складні структури даних та класи що не можуть бути серіалізовані. Тож перед збереженням нам слід перевести їх у прості типи даних. У цьому нам допоможе система регістрів, що надає унікальні ідентифікатори для необхідних типів, а також окремий клас SaveFile який ми і будемо серіалізувати.

```
private void SaveResources(GameProgress progress)
{
    resources = progress.resourceContainer.Resources;
    resourceStatistics = new List<ResStatItem>();
```

```
        everydayTransactions = new();

        foreach (var res in progress.resourceContainer.Resources)
        {
            var transaction = progress.resourceContainer
                .resourceMarket.GetTransaction(res.name);
            transaction.resource = res.name;
            everydayTransactions.Add(transaction);

            foreach (var item in progress
                .resourceContainer.GetStatistics(res.name))
            {
                resourceStatistics.Add(new ResStatItem
                {
                    resource = res.name,
                    transactionType = item.Key,
                    gained = item.Value.gained,
                    spent = item.Value.spent,
                });
            }
        }
    }
```

У коді вище зображена функція що відповідає за переведення стану головного контейнера ресурсів у простий формат, що можна серіалізувати. Завантажити ж дані назад з десеріалізованого файлу в клас прогресу можна за допомогою наступної функції.

```
    private void LoadResources(GameProgress progress)
    {
        progress.resourceContainer.Resources.Clear();
        progress.resourceContainer.AddResources(resources);
        var resGroup = resourceStatistics.GroupBy(r => r.resource);

        foreach (var res in everydayTransactions)
        {
            progress.resourceContainer.resourceMarket
                .SetTransaction(res.resource, res);
        }

        foreach (var group in resGroup)
        {
            var record = new Dictionary<TransactionType,
    ResourceDynamic>();

            foreach (var item in group)
            {
                record.Add(item.transactionType,
                    new ResourceDynamic
                    {
                        gained = item.gained,
```

```
                    spent = item.spent
                };
            );
        }

        progress.resourceContainer.AddStatistics(group.Key, record);
    }
}
```

Таким чином і реалізується передача даних між сесіями. Також у цього підходу є ще один плюс, система спроектована таким чином повністю залежить від даних що дає нам змогу маніпулювати її станами ззовні не потребуючи маніпуляцій з кодом програми.

4.4 Створення користувацького інтерфейсу

Одним з найбільш важливих процесів для частини функціоналу гри, що розглядається в цій роботі є створення інтерфейсу, адже більшість механік безпосередньо залежать від нього. Тож для початку розглянемо основні положення розробки користувацького інтерфейсу в Unity.

Базовими елементами будь якого інтерфейсу в цьому рушії є Canvas, що відповідає за відображення та EventSystem, що відповідає за події, такі як натискання кнопок, наведення курсора, перетягування тощо. Для формування самого інтерфейсу використовуються в основному вбудовані елементи, такі як кнопки, повзунки тощо, зображення що використовують спрайти та компоненти що відповідають за розташування. Зазвичай інтерйфейси гри створюються за допомогою редактора рушія, але його гнучкість дозволяє створювати деякі елементи візуального супроводу на льоту, генеруючи їх за допомогою скриптів.

Комбінуючи всі вищезазначені компоненти можна формувати прекрасні комплексні інтерфейси для будь якої гри (див. рис. 4.5)

Рисунок 4.5 – Зовнішній вигляд вікна вмінь (власний рисунок)

На попередньому рисунку показано зовнішній вигляд інтерфейсу одного з найбільш цікавих вікон гри, адже воно є максимально динамічним та реактивним. Всього в грі присутні 5 фракцій, кожна з яких має своє унікальне дерево вмінь, при цьому одне розроблене вікно здатне відображати кожне з них, маючи відповідні налаштування.

4.4.1 Відображення динамічних списків

Відображення списків в інтерфейсі будь якої гри є однією з основних задач. Це може бути як простий список ресурсів чи характеристик так і комплексні візуальні структури. Якщо такі елементи є статичними, наприклад як список фракцій гри з відповідними індикаторами прогресу, то для їх реалізації достатньо вбудованих інструментів та редактора. Для формування списків в Unity використовуються компоненти сімейства LayoutGroup, це три види списків, горизонтальні, вертикальні та у вигляді решітки. За їх допомогою можна

розташовувати та управляти положенням колекцій об’єктів користувацього інтерфейсу, та формувати майже будь які макети інтерфейсів.

Але що ж коли такі списки мають бути динамічними? Наприклад інвентар користувача. Предмети звідти можуть зникати та з’являтись під час виконання програми, а відображення має відповідати цим змінам в реальному часі. Для цього рушій не надає ніяких спеціальних інструментів, тож слід використовувати самописні рішення.

Таким рішенням є компонент ListViewController, це інструмент для зовнішнього керування списками, що дозволяє відображати колекції візуальних елементів, оновлювати їх за запитом та надає функціонал для відслідковування взаємодії з елементами списку. За його допомогою можна досить просто «оживити» більшість списків і зробити їх не лише динамічними, а і інтерактивними (див. рис. 4.6). Детальний код цього компоненту наведено в додатку Г.

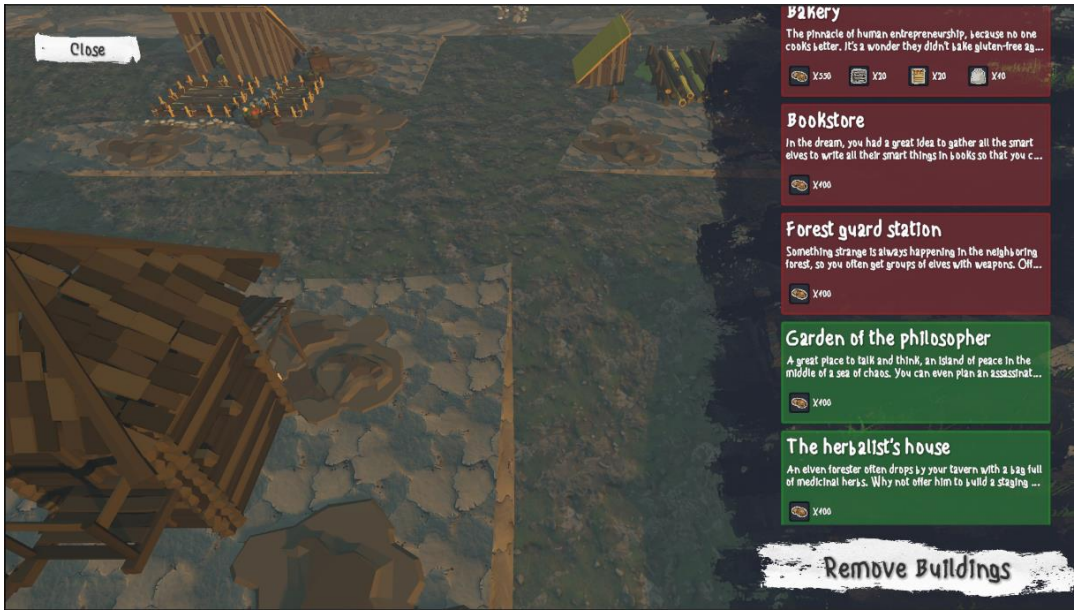


Рисунок 4.6 – Динамічний і інтерактивний список будівель в режимі будівництва (власний рисунок)

Прикладом використання цього інструменту є список будівель в режимі будівництва. Це складна композиція для представлення, адже тут фігурує не лише велика кількість елементів з різним наповненням, а ще й різні стани елементів на

які має реагувати список. Але за допомогою компоненту для відображення списків можна просто реалізувати таку структуру, головне сформувати правильний префаб (шаблон ігрового об’єкту) візуального елементу.

Для роботи цей компонент потребує 2 основні параметри. Перший це префаб візуального елементу, що має бути відображений та реалізує інтерфейс IListItem, що дає змогу ініціалізувати цей лемент даними, отримати порівняння, та реагувати на взаємодію з елементом. Цей параметр є статичним та задається у вікні редактора. На основі наданих префабів і буде формуватись список. Другий параметр задається під час виконання через функції Init чи Refresh, і є списком посилань на об’єкти що є носіями даних для елементів списку.

4.4.2 Система підказок

Враховуючи величезну кількість інтерфейсно орієнтованих механік та їх велику концентрацію на одній сцені, користувачеві може бути складно орієнтуватись та запам’ятовувати призначення кожної кнопки чи індикатора. Але на допомогу йому прийде система підказок що впроваджена в гру.

Це гнучка та проста система що дозволяє надавати додаткові роз’яснення щодо візуальних елементів інтерфейсу при довгому наведенні. При цьому простота реалізації вражає. Нам необхідний об’єкт Canvas в якому розмішуватиметься об’єкт підказки, що містить в собі заголовок та описання. А другим компонентом є тригер підказок, що відслідковує наведення курсора на візуальний елемент та повідомляє системі підказок які текстові дані слід відображати, в залежності від джерела цих даних чи введених статичних показників.

```
public class TooltipTrigger : MonoBehaviour, IPointerEnterHandler,
IPointerExitHandler
{
    public string header;
    [TextArea]
```

```
    public string content;
    public float delay = 0.5f;

    public TooltipDataProvider dataProvider;
    public string providerTag = null;

    public void Init(string content, string header = "")
    {
        this.content = content;
        this.header = header;
    }

    public void OnPointerEnter(PointerEventData eventData)
    {
        StartCoroutine(ShowRoutine());
    }

    public void OnPointerExit(PointerEventData eventData)
    {
        StopAllCoroutines();
        TooltipSystem.Hide();
    }

    private IEnumerator ShowRoutine()
    {
        yield return new WaitForSeconds(delay);

        if (dataProvider is not null)
        {
            header = dataProvider.GetHeader(providerTag);
            content = dataProvider.GetContent(providerTag);
        }

        TooltipSystem.Show(content, header);
    }
}
```

Маючи такий компонент тригера можемо навісити його на будь який елемент інтерфейсу і отримувати спливаючі підказки кожного разу при наведенні курсора на нього (див. рис. 4.7).

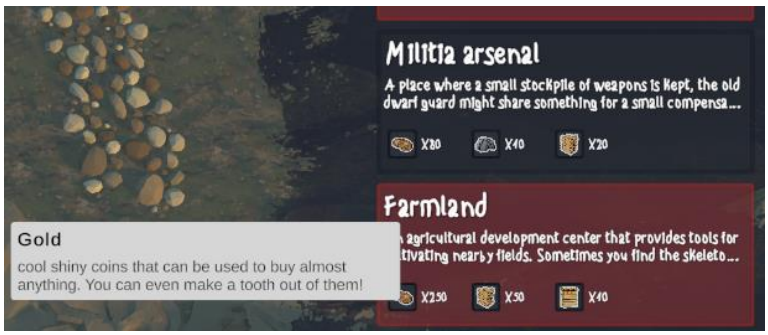


Рисунок 4.7 – Приклад простої текстової підказки (власний рисунок)

Але на основі цього підходу можна формувати і більш комплексні на інформативні підказки. Наприклад в системі інвентаря, досить складно розмістити статичні елементи що відзначають показники тих чи інших елементів екіпірування. Але за допомогою спливаючих підказок це можливо досить просто та елегантно вирішити, та ще й надавати порівняння з екіпірованими предметами.

Рисунок 4.7 – Приклад підказки характеристик предметів (власний рисунок)

Ці ускладнені підказки працюють за тим же принципом що і звичайні текстові, але як джерело даних приймають об’єкт предмету інвентаря. Також таким чином можна формувати будь які комплексні підказки, що дозволить не лише розгрузити статичні візуальні елементи, а й значно полегшити сприйняття інтерфейсу гравцем.

4.4.3 Управління вікнами

Однією з найбільш сильних сторін рушія Unity є простота та гнучкість, що дозволяє максимально модифікувати та надбудовуватись над тими інструментами

що він надає. Але через це в стандартній бібліотеці відсутні складні підсистеми що включають роботу з інтерфейсами в середині гри. Наприклад відсутнє не те що система управління, а самі поняття модальних, діалогових та звичайних вікон. А оскільки в розроблюваній грі ці візуальні компоненти відіграють ключову роль, особливо в контексті механік поселення, слід розробити подібну систему власноруч.

Основним завданням цієї системи є зручне створення та управління вікнами в середині гри, без прив’язки до конкретної імплементації цих вікон. Адже в контексті проекту присутні як невеликі суто інформаційні віконця, так і складні багатошарові модальні вікна, з купою функціоналу та можливістю відкривати інші.

Для цього знову скористаємось скриптовими об’єктами та сформуємо систему контролю модальних вікон, що міститиме в собі стан (кількість та якість відкритих вікон) та дві контролюючі функції закривання та відкривання вікон. Система управління базується на структурі даних стеку, де додавання нового вікна завжди буде відображувати його поверх попередніх, а видалення закриватиме останнє додане до стеку вікно.

Також для повноцінного функціонування цієї системи, нам необхідна абстракція самого модального вікна, код відповідного розробленого класу можна передивитись в додатку Г. Ця реалізація надає простий скелет алгоритму для виклику та видалення простого модального вікна що має заголовок, основний текст та декілька кнопок що повертають якийсь результат діалогу. В поєднанні з класом менеджера вікон, дещо нагадує шаблон проектування стратегії, адже модальне вікно підтримує розширення, тож для реалізації більш складних вікон, слід унаслідувати функціонал від базового класу, вписати свої параметри та перевизначити необхідні функції.

Основним плюсом такого підходу є те, що ми можемо контролювати відкриття вікон в залежності від їх стану та стану гри. Наприклад якщо є якісь вікна що мають блокувати відкриття інших, то реалізувати це можна обравши відповідний пункт в налаштуваннях відповідного вікна (див. рис. 4.8).

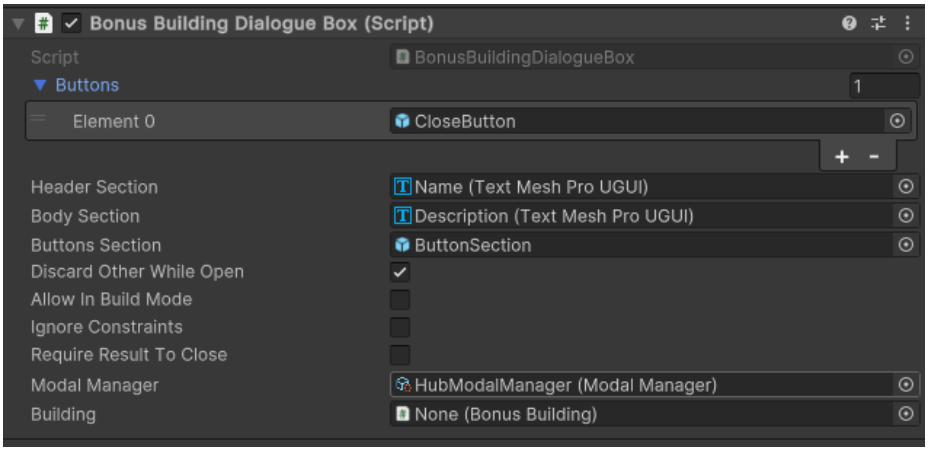


Рисунок 4.8 – Налаштування розширеного модального вікна (власний рисунок)

Таким чином можна формувати будь які діалогові вікна за допомогою префабів та зручно і ефективно контролювати їх стан у будь якому куточку програми.

4.4.4 Впровадження графічного матеріалу за допомогою штучного інтелекту

Головним трендом індустрії інформаційних технологій останні два роки є штучний інтелект. І хоча можливості штучного інтелекту використовуються в ігрових продуктах вже досить давно, сучасні розробки потужних генеративних мовних моделей відкривають перспективи використання сучасних інструментів на їх основі не тільки в ігровому процесі, а й для формування контенту.

Очевидним є використання сучасних мовних моделей (ChatGPT, Bard) для формування текстового вмісту гри, але більш цікавим є використання генеративних інструментів на їх основі для створення візуального вмісту. Прикладом такого інструменту є Neural.love [11], що дозволяє перетворювати прості текстові команди у цифрові зображення. Основною перевагою цього інструменту є те, що усі зображення що генеруються за допомогою безкоштовного режиму,

розповсюджуються за вільною ліцензією CC0, що дозволяє їх використовувати в будь якій діяльності без надавання посилань чи грошових відшкодувань.

Протестуємо можливості використання цього інструменту в грі на прикладі формування іконок для ресурсів. Всього в грі присутній 51 різний ресурс, від золотих монет і дерева, до алхімічних реагентів та хутряних виробів. Для невеликої команди розробки, що не включає в себе графічних дизайнерів формування графічного вмісту відповідної якості є надскладною задачею, адже пошук безкоштовних рішень підіймає проблему відмінності стилів та постійної відсутності необхідних деталей, купівля наборів зображень на маркетплейсах стикається з тими самими проблемами до яких ще додаються значні витрати коштів, а замовлення матеріалів у сторонніх художників може бути занадто коштовним. Тож більш-менш логічним виходом в таких умовах є використання генеративних рішень. Перевіримо чи доцільно застосовувати подібні інструменти на прикладі згаданого раніше NeuralLove.

Припустимо нам необхідно згенерувати іконку для ресурсу вугілля. Для цього використаємо наступні налаштування інструменту (див. рис. 4.9) та запустимо генерацію.

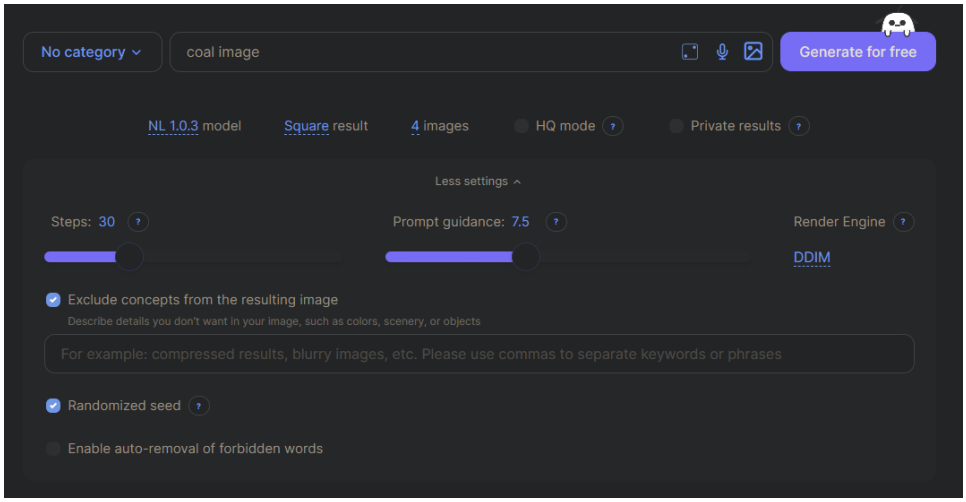


Рисунок 4.9 – Приклад налаштуваннь генератора зображень (власний рисунок)

З використанням налаштувань, які зображені на попередньому рисунку ми отримуємо наступні зображення (див. рис. 4.10)

Рисунок 4.10 – Приклад зображень, що були згенеровані за першочерговими налаштуваннями (власний рисунок)

На попередньому рисунку бачимо що цей штучний інтелект має тенденцію до генерації повноцінних сцен з реалістичними елементами. Це не зовсім підходить під естетику гри, а запропоновані інструментом стилі генерації нам не підходять, тож спробуємо вказати «мальований» стиль в якості запиту та конкретизуємо що нам необхідні іконки (див. рис. 4.11).

Рисунок 4.11 – Зображення другої спроби (власний рисунок)

В цьому випадку бачимо що зображення взагалі мало схожі на те що нам потрібно, тож слід експериментувати з налаштуваннями та командами. В результаті цих експериментів було визначено оптимальний формат команд для генерації. Це має бути запит на набір стікерів, у необхідному стилі з максимально спрощеними критеріями об’єкту генерації (тобто замість вугілля це має бути чорний камінь, замість пороху – чорний гранульований порошок тощо). Це дозволяє тримати приблизно однаковий стиль іконок, отримувати одразу велику кількість результатів для пришвидшення роботи та в деяких випадках дозволяє уникнути проблем з генерацією абстракції (див. рис. 4.12).

Рисунок 4.12 – Приклад зображень, що були згенеровані за оптимальними налаштуваннями (власний рисунок)

Ці зображення все ще не є ідеальними, але в процесі генерації та за допомогою додаткової обробки зображень що включає в себе обрізання, видалення фону, додавання спеціальних ефектів, штучне покращення якості зображень, можна досить швидко отримати велику кількість якісних зображень що підпадають під конкретні вимоги та відповідають визначеному стилю, при цьому не витрачаючи багато часу (див. рис. 4.13).

Рисунок 4.13 – Генеровані зображення що були впроваджені в гру (власний рисунок)

Але все ж такий підхід є не ідеальним, кожне генероване зображення залежить від набору даних на якому тренувалась модель. Тому часто виникають проблеми, коли наприклад необхідно згенерувати купу дерев'яних колод, для представлення ресурсу дерева, для цього використовуються всі описані вище застереження, а на виході ми отримуємо абстракцію, яка або не відповідає запиту або порушує візуальне сприйняття об'єкту, або взагалі переосмислює той чи інший, здавалося б простий об'єкт, у той спосіб який не є прийнятним (див. рис. 4.14).

Рисунок 4.14 – Проблемні генерації купи колод та простого меча (власний рисунок)

Причиною цього є недостатня «обізнаність» генераційної моделі деякими концепціями, що і призводить до дивних результатів. І єдиним вирішенням такої проблеми є лише зміна генеративної моделі, що не завжди є доцільним.

Тож як висновок можна винести тезу про можливість використання генеративних інструментів на основі штучного інтелекту для генерації візуального матеріалу для ігор. Але слід приймати до уваги зазначені вище проблеми, та наявність платних сервісів, які, можливо, надають більший спектр можливостей, якості генерації та налаштувань, при цьому не занадто обтяжуючи витратами команду розробників.

Також окрім сервісів генерації слід відмітити деякі інші інструменти які стали доступними завдяки прогресу в розвитку штучного інтелекту. Існує безліч різноманітних сервісів що інкорпорують в свою роботу можливості штучного інтелекту і можуть значно спростити роботу пов’язану з генерацією візуального контенту для різних ігор. Ми ж зосередимося на сервісах покращення якості та видалення фону.

Такі сервіси існували безумовно і раніше, і виконували те для чого були призначені при цьому базуючись більше на алгоритмічних підходах. Але у випадку видалення фонів, то завжди приходилось виділяти області що мають бути видалені, а результати покращення якості часто могли бути не такими як очікувалось. Впровадження ж можливостей штучного інтелекту значно прискорює ці процеси, звільняючи відповідальних за ці прості але кропіткі процеси людей від монотонної роботи, та надаючи змогу витрачати зекономлений час на розробку та впровадження продукту.

5 ТЕСТУВАННЯ

Тестування програмного продукту, особливо в ігровій сфері є одним з ключових етапів розробки. При цьому саме тестування проводиться увесь час, починаючи з ранніх стадій розробки і часто не закінчується і на етапі випуску і підтримки застосунку.

В нашому випадку додаток знаходиться на проміжній стадії розробки, коли присутнє основне ядро механік, що взаємодіють між собою та стартовий набір контенту що надає змогу перевірити їх працеспроможність. Тестування продукту на цьому етапі є визначальним для подальшої якості продукту, адже сформоване ядро надалі буде лише розвиватись та обростати новим контентом, але основний функціонал має залишатись надійним та відповідати вимогам. Таким чином слід звернути увагу також і на тестування продукту.

Для початку звернемося до сучасних практик з тестування програмних продуктів. Зокрема нас цікавить організація циклу тестування, адже це є неперервним процесом та проводитиметься протягом всього процесу розробки. В сучасній розробці часто застосовують стратегію схожу на стандартний цикл розробки програмного забезпечення – STLC [12], життєвий цикл тестування програмного забезпечення (див. рис. 5.1).

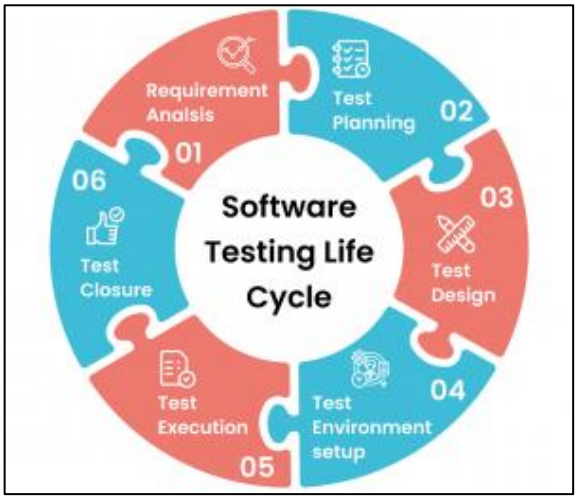


Рисунок 5.1 – Цикл STLC (з за даними [12])

Цей цикл включає в себе 6 основних етапів які слід враховувати при тестуванні будь якого програмного продукту. Тож надалі розберемо кожен з них, та застосуємо такий підхід на прикладі даного проекту.

Першим пунктом є визначення та аналіз вимог до системи та її компонентів. Тут формується основна загальна документація щодо етапу тестування, де вказуються цілі, вхідні та вихідні критерії та об’єкт тестування. Даний проект тестувався комплексно після розробки основного функціоналу, тож на першому етапі формувався майстер тест-план, який визначав описані вище показники для всього розробленого функціоналу. Примірник цього тест-плану можна передивитись у додатку Д. Усі подальші посилання на тест-план у цьому документі стосуються саме нього.

На наступному етапі – плануванні тестування, ми формуємо загальне уявлення про методи, підходи та об’єми тестування, а також про інструментар та персонал що буде залучений. Цей пункт також вимагає роботу з тест-планом, а саме доповнення його цією інформацією. Для даного проекту визначено 5 видів тестування:

- функціональне тестування – тестування окремих компонентів та функцій програми на відповідність специфікації;
- тестування взаємодії – тестування взаємодії різних компонентів та можливостей на перетині їх функціоналу;
- тестування продуктивності – тестування показників працеспроможності програми та виявлення проблем пов’язаних з продуктивністю;
- тестування зручності – перевірка можливості комфортного використання основних функцій тестованого компоненту програми;
- тестування користувацького інтерфейсу – перевірка відповідності користувацького інтерфейсу специфікаціям та виконання ним своїх функцій.

Наступний етап характеризується формуванням так званих тест-кейсів, невеликих документів що детально описують кожен тестовий випадок, тобто що та як тестується. Ці документи є невід’ємною частиною повноцінно

задокументованого процесу тестування і формуються на основі вимог та області тестування зазначених в тестовому плані. На основі цих тест-кейсів далі проводиться самі випробування та перевірки програмного забезпечення. Приклад заповненого тест-кейсу для перевірки функціонування інвентаря можна подивитись у додатку Е.

Наступний етап це умовно підготовка до тестування, тут у відповідності до попередньо розроблених документів тест плану та тест кейсів налаштовується середовище для тестування продукту. Сюди в контексті ігрової розробки може входити:

- конфігурація рушія;
- конфігурація віртуальних середовищ виконання;
- конфігурація налаштувань якості;
- визначення необхідних налаштувань гри;
- створення необхідних умов для тестування функціоналу (створення необхідного для тестування контенту).

Далі йде найбільш кропіткий етап – власне тестування. Воно також як і на попередньому етапі спирається н попередньо розроблену документацію, але також і генерує власну. Основним артефактом на цьому етапі є баг-репорт (приклад якого можна побачити у додатку Ж), що інформує команду про виявлення тієї чи іншої проблемі в тестованому проекті. Це ще один невеликий документ, що має на меті зафіксувати та максимально деталізувати конкретну помилку в грі. Найбільш важливим при цьому є визначення кроків для відтворення та фіксація самого «багу» (фото, відео, логи, телеметрія тощо), для подальшого полегшення процесу виправлення виявлених помилок.

І останній шостий етап цього циклу – це етап аналізу отриманих даних та формування вихідної документації з результатами тестування. На цьому етапі всі процеси тестування вже завершено та їх результати оброблюються для визначення функціоналу який необхідно доопрацьовувати чи перероблювати.

Далі процес тестування запускається знову в тому ж порядку за необхідності, задля підтвердження якості розроблюваного продукту.

5.1 Використання вбудованих інструментів рушія для проведення тестування продуктивності

Особливу увагу при тестуванні ігрового програмного забезпечення прийнято також приділяти тестуванню продуктивності додатку. Для цього в тест-плані виділено та деталізовано окремий вид тестування. Основним інструментом для цього виду тестування програмного забезпечення є вбудований в рушій Unity профайлер, що дозволяє перевірити навантаження на систему, використання ресурсів, час виконання тих чи інших функцій, тощо.

Тож далі наведемо приклад застосування цього підходу для комплексної перевірки підсистем програми. Для цього запустимо профайлінг та виконаємо наступні дії:

- завантажимо головну сцену поселення;
- проведемо різнопланову взаємодію з камерою (переміщення, повороти, приближення);
- перейдемо в меню розвитку в таверні;
- нарахуємо додаткові очки розвитку;
- вивчимо нове уміння що дозволяє побудувати кам’яний кар’єр;
- повернемось до перегляду поселення;
- перейдемо в режим будівництва;
- розмістимо кар’єр;
- визначимо обсяги виробництва в кар’єрі.

Ці дії надають нам можливість перевірити продуктивність таких компонентів та функцій програми:

- загальна продуктивність відображення сцени в залежності від положення камери;
- продуктивність роботи менеджера вікон;
- продуктивність роботи системи розвитку персонажа та суміжних функцій системи будівель та збереження даних;

- продуктивність системи будівництва;

- продуктивність системи ресурсів, та її взаємодії з системою будівництва.

Після виконання цих дій в середовищі рушія з ввімкненим профайлером ми отримуємо детальний та інтерактивний графічний звіт з даними придатними для подальшого аналізу (див. рис. 5.2).

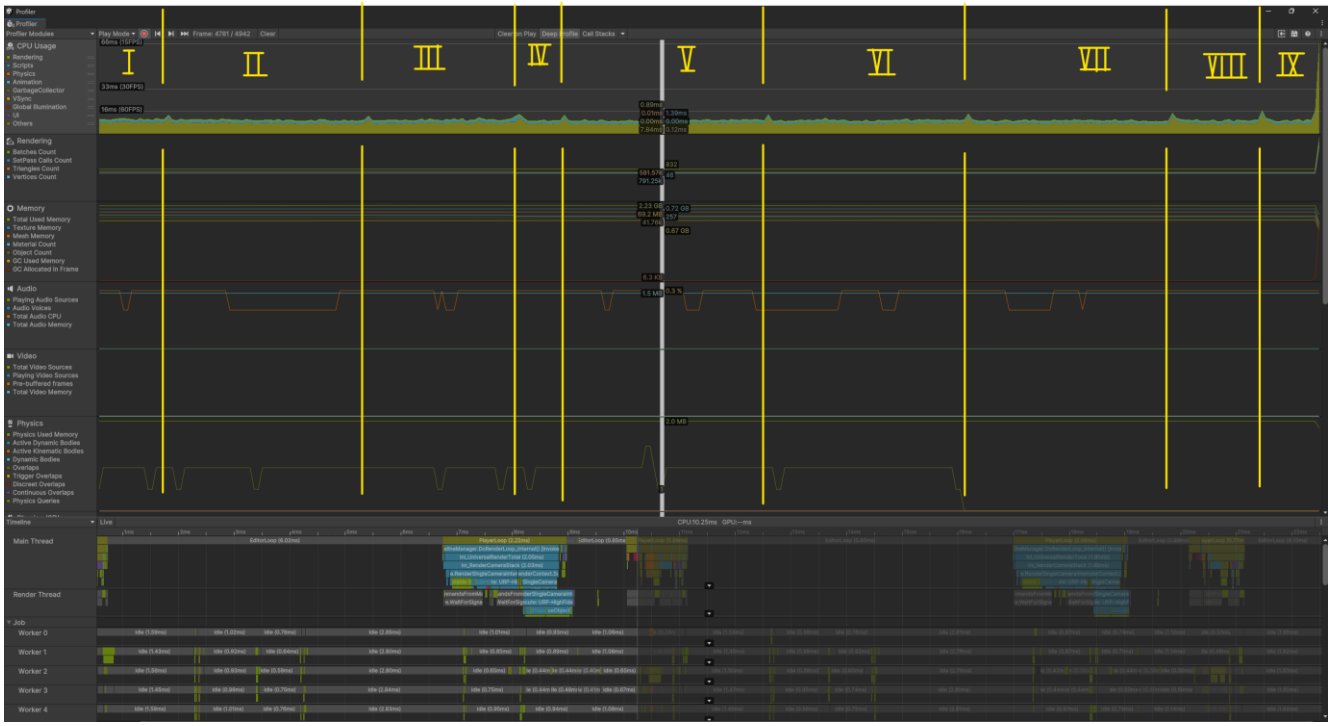


Рисунок 5.2 – Вигляд вікна профайлера після прогону тесту (власний рисунок)

Отримані дані у вікні профайлера мають вигляд графіків що позначають використання різних ресурсів системи. Також можна детально вивчати ці показники та статус виконання функцій у кожний окремий момент часу, перетягуючи білий маркер. На попередньому рисунку також жовтими маркерами відображено кожен з 9 кроків тестування за якими можна прослідкувати виділення ресурсів на ті чи інші задачі та функції.

Загалом за цими даними можна зробити висновок що додаток працює досить стабільно, нарівні вищому за 60 кадрів на секунду, за даної конфігурації програмного забезпечення. Виникають невеликі та не критичні скачки в використанні ресурсів при завантаженні та вивантаженні деяких компонентів.

6 ВПРОВАДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

На даному етапі розробки програмного продукту про повноцінне впровадження та продаж гри не йдеться. Адже загальна мета проекту є комплексною та не може бути охоплена в рамках однієї дипломної роботи. Але часткове впровадження здійснювалось протягом усього періоду розробки гри, про що і поговоримо у цьому розділі.

В першу чергу слід зазначити дослідження можливостей використання генеративних інструментів на основі штучного інтелекту для створення контенту гри, розглянуті у попередніх розділах. Перед цим було проведено дослідницьку та аналітичну роботу з дослідження можливостей використання штучного інтелекту в сфері розробки ігрового програмного забезпечення. Частина матеріалів цієї роботи представлена в цьому документі в розділі 4.4.4, інша ж частина була оформлена та опублікована у форматі тези на першій міжнародній науковій та теоретичній конференції «Scientific review of the actual events, achievements and problems», що проходила в Берліні, 01.12.2023 [13]. Підтвердження участі та публікації цього та інших матеріалів що пов’язані з даною роботою наведені в додатку И. Ця теза фокусувалась на пошуку можливих застосувань сучасних надбань у сфері штучного інтелекту у сфері розробки ігрового програмного забезпечення та контенту до нього і є логічним попередником дослідження про генерацію графічного матеріалу що розглядався у цій роботі.

Далі трохи відходячи від наукової сторони роботи, за занурюючись у практичну площину застосування результатів роботи зазначимо, що протягом розробки програмного продукту активно використовувались можливості соціальних мереж для інформування потенційних користувачів як про сам продукт так і про процес розробки. Для цього використовувались основні сучасні соціальні мережі зокрема Telegram та Instagram що показали не погані показники зацікавленості та активності аудиторії. За трохи більше ніж місяць існування та активної підтримки каналів у телеграмі та інстаграмі, без залучення сторонніх

методів просування контенту було здобуто 15 підписників, в середньому 40 переглядів та 15 читачів, в середньому по 22 перегляди відповідно (див. рис. 6.1).

Рисунок 5.2 – Шапка профілю каналу гри в Instagram (власний рисунок)

Ці медійні ресурси є не поганим початком для подальшого розвитку зацікавленості потенційних майбутніх користувачів у грі та може слугувати додатковим джерелом надходження коштів для розвитку проекті.

Також ще одним проявом впровадження проекту в більш практичній площині є участь команди розробки у виставці технічної творчості молоді 28-го міжнародного молодіжного форуму «Радіоелектроніка та молодь у XXI столітті» за напрямом «Ігрові технології» [14]. На цій виставці було представлено розроблений функціонал додатку в тому числі і ті компоненти роботи що детально розглядались у попередніх розділах. За результатами проведеної виставки команда отримала 3 місце серед усіх представлених на цьому напрямку виставки проектів.

Наступними кроками для впровадження розроблюваного ігрового продукту є продовження підтримки соціальних мереж та впровадження нових методів залучення майбутніх користувачів. На більш пізніх етапах розробки можливе впровадження нових осередків медійної активності з основним акцентом на залучення коштів для запуску гри на сервісах цифрової дистрибуції. Наприклад за допомогою сервісу Patreon, на якому розміщуватимуться ексклюзивні матеріали про гру та її розробку доступні лише для платних підписників. Така активність дозволить частково отримати кошти для випуску гри, або підвищить шанси отримати фінансування чи пропозиції просування.

ВИСНОВКИ

У попередніх розділах цієї роботи було розглянуто основні моменти що стосуються всіх основних етапів розробки програмного продукту. А саме підготовчий етап, з формуванням вимог та аналізом предметної області, а також проектування, розробка та тестування програмного забезпечення.

На етапі формування вимог та аналізу було пройдено шлях від абстрактної ідеї гри, до формування конкретних вимог до частини функціоналу програмного забезпечення, що є результатом даної роботи. Також на цьому етапі було проаналізовано загальний ринок ігрових застосунків та основні тенденції і вподобання користувачів, на основі чого було проведено аналіз конкурентів.

На наступному етапі проектування було визначено основні практичні риси майбутнього додатку за допомогою загальної архітектури системи та проектування її інтерфейсів. Також тут було пророблено роботу з декомпозиції окремих функцій застосунку на підсистеми та запропоновано можливі архітектурні рішення для їх реалізації. Найбільш цікаві моменти проектування було також висвітлено тут.

Далі на етапі розробки за допомогою рушія Unity та можливостей мови програмування C#, були впроваджені практичні рішення для виконання завдання та формування частини програмного продукту що виконувала б завдання визначені раніше. На цьому етапі активно використовувались попередні наробки щодо архітектурного проектування. Також на цьому етапі піднімалось питання створення та впровадження контенту в гру за допомогою ШІ.

Останній етап тестування дозволив провести аналіз отриманого продукту та сформувати уявлення про його якість та слабкі сторони його реалізації. Результатами цього етапу є ціна інформація та документація що може бути використана на майбутніх циклах розробки.

В результаті ми отримали програмний продукт, що містить в собі повноцінно реалізовану частину функціоналу фінальної гри і є основою для створення цікавого та перспективного застосунку.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Gaming Sector Bigger Than Films and Music? - Access Creative. Access Creative College. URL: https://www.accesscreative.ac.uk/blog/is-the-gaming-industry-bigger-than-films-and-music/ (дата звернення: 11.04.2024).

2. Top 10 RPGs To Play Right Now. Game Informer. URL: https://www.gameinformer.com/2024/01/26/top-10-rpgs-to-play-right-now (дата звернення: 11.04.2024).

3. Dungeons, Delves, and Dice. The Psychology of Why We Love Role Playing Games. Medium. URL: https://medium.com/@d3ballads/the-psychology-of-why-we-love-role-playing-games-29e9ae45efe4 (дата звернення: 11.04.2024).

4. Cult of the Lamb. Сервіс цифрової дистрибуції Steam. URL: https://store.steampowered.com/app/1313140/Cult_of_the_Lamb/(дата звернення: 11.04.2024).

5. XCOM® 2. Сервіс цифрової дистрибуції Steam. URL: https://store.steampowered.com/app/268500/XCOM_2/ (дата звернення: 11.04.2024).

6. Age of empires II. Сервіс цифрової дистрибуції Steam. URL: https://store.steampowered.com/app/813780/Age_of_Empires_II_Definitive_Edition/ (дата звернення: 11.04.2024).

7. Frostpunk. Сервіс цифрової дистрибуції Steam. URL: https://store.steampowered.com/app/323190/Frostpunk/ (дата звернення: 12.04.2024).

8. Fallout shelter. Сервіс цифрової дистрибуції Steam. URL: https://store.steampowered.com/app/588430/Fallout_Shelter/ (дата звернення: 12.04.2024).

9. Ніколаєнко І. В., Новіков Ю. С. Жанровая классификация видеоигр. 19-тий міжнародний молодіжний форум «Радіоелектроніка та молодь у XXI столітті». 2015.

10. Kryzarel. Character stats (aka attributes) system. Unity Forum. URL: https://forum.unity.com/threads/tutorial-character-stats-aka-attributes-system.504095/ (дата звернення: 12.04.2024).

11. Neural love. Найпростіший генератор зображень. URL: https://neural.love/ai-art-generator (дата звернення: 11.05.2024)

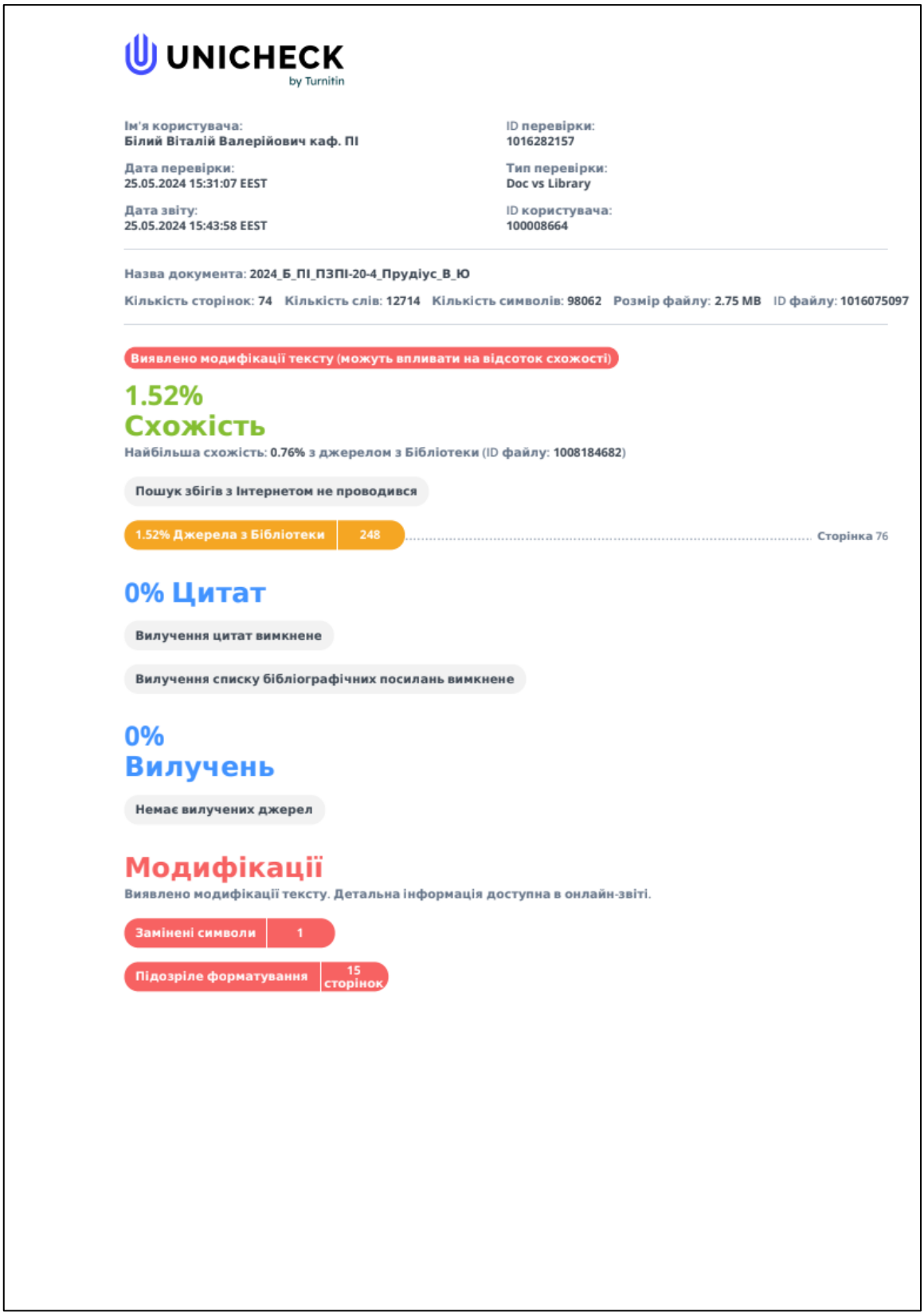
12. What is Software Testing Life Cycle? Optimumbrew. URL: https://optimumbrew.com/what-is-software-testing-life-cycle/ (дата звернення: 13.05.2024).

13. Прудіус В. Ю. Перспективи використання генеративного штучного інтелекту в розробці ігрових застосунків. I International Scientific and Theoretical Conference «Scientific review of the actual events, achievements and problems» : матеріали Міжнар. наук. концеренцій, м. Берлін, 1 груд. 2023 р. 2023. С. 159–160. URL: https://doi.org/10.36074/scientia-01.12.2023 (дата звернення: 18.05.2024).

14. Радіоелектроніка та молодь у XXI столітті. Каталог виставки технічної творчості молоді : матеріали 28-го Міжнар. молодіж. форуму, 16–18 квіт. 2024 р. / М-во освіти і науки України, Харків. нац. ун-т радіоелектроніки. – Харків : ХНУРЕ, 2024. С. 14. URL: https://openarchive.nure.ua/entities/publication/50c553ae-4c5c-4ca5-b137-52314123e2cb (дата звернення: 18.05.2024).

ДОДАТОК А

Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ



UNICHECK
by Turnitin

Ім'я користувача:
Білий Віталій Валерійович каф. ПІ

ID перевірки:
1016282157

Дата перевірки:
25.05.2024 15:31:07 EEST

Тип перевірки:
Doc vs Library

Дата звіту:
25.05.2024 15:43:58 EEST

ID користувача:
100008664

Назва документа: 2024_Б_ПІ_ПЗПІ-20-4_Прудіус_В_Ю

Кількість сторінок: 74 Кількість слів: 12714 Кількість символів: 98062 Розмір файлу: 2.75 MB ID файлу: 1016075097

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

1.52%
Схожість

Найбільша схожість: 0.76% з джерелом з Бібліотеки (ID файлу: 1008184682)

Пошук збігів з Інтернетом не проводився

1.52% Джерела з Бібліотеки 248 ..... Сторінка 76

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи 1

Підозріле форматування 15 сторінок

ДОДАТОК Б

Слайди презентації

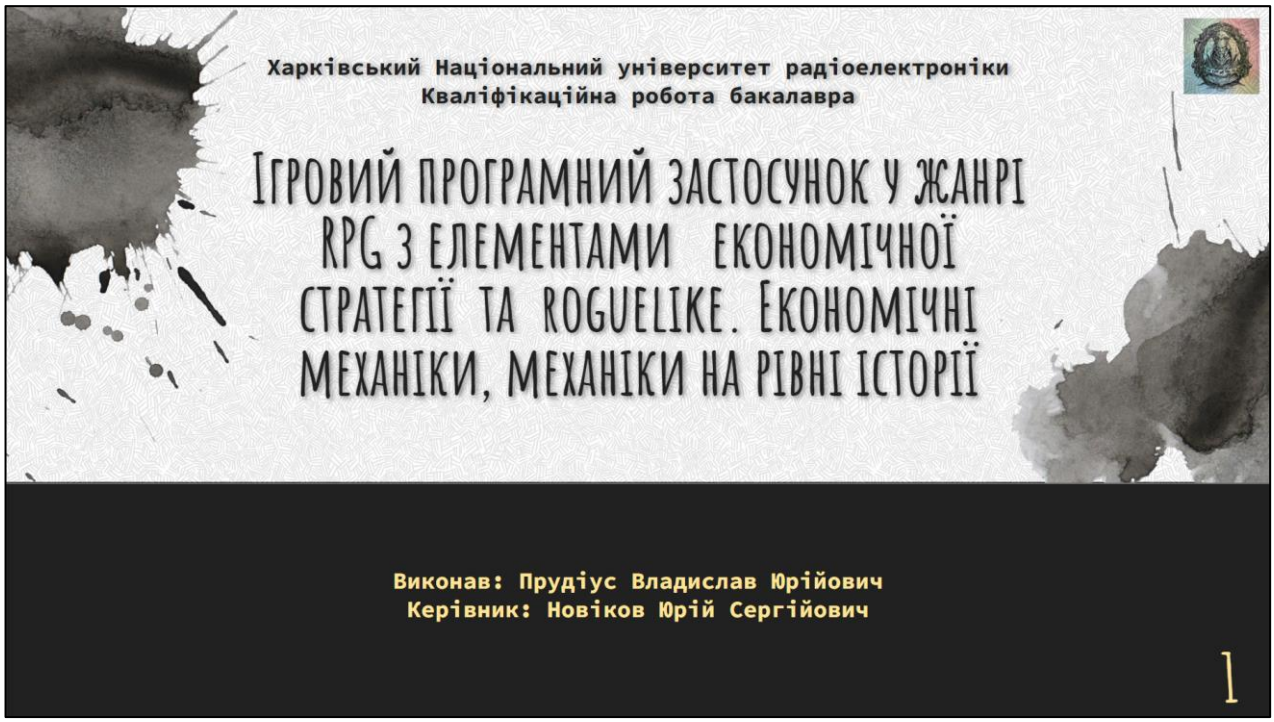


Рисунок М.1 – Слайд №1 презентації

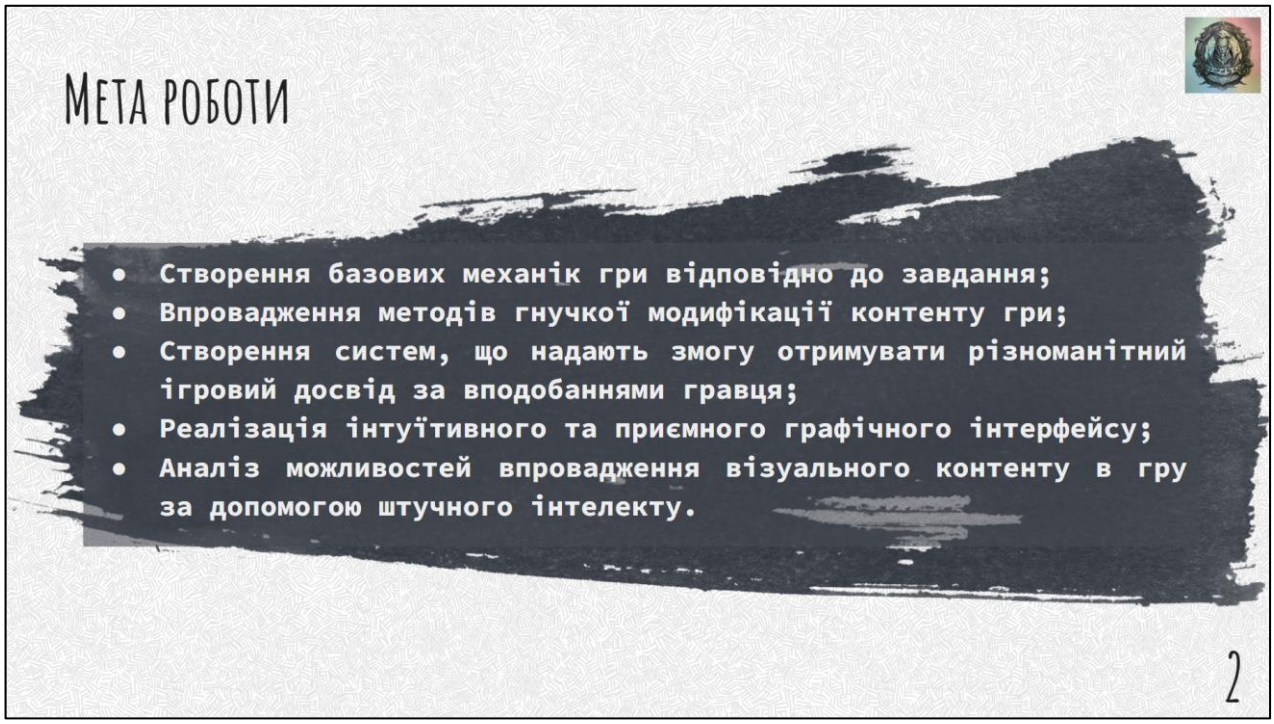


Рисунок М.2 – Слайд №2 презентації

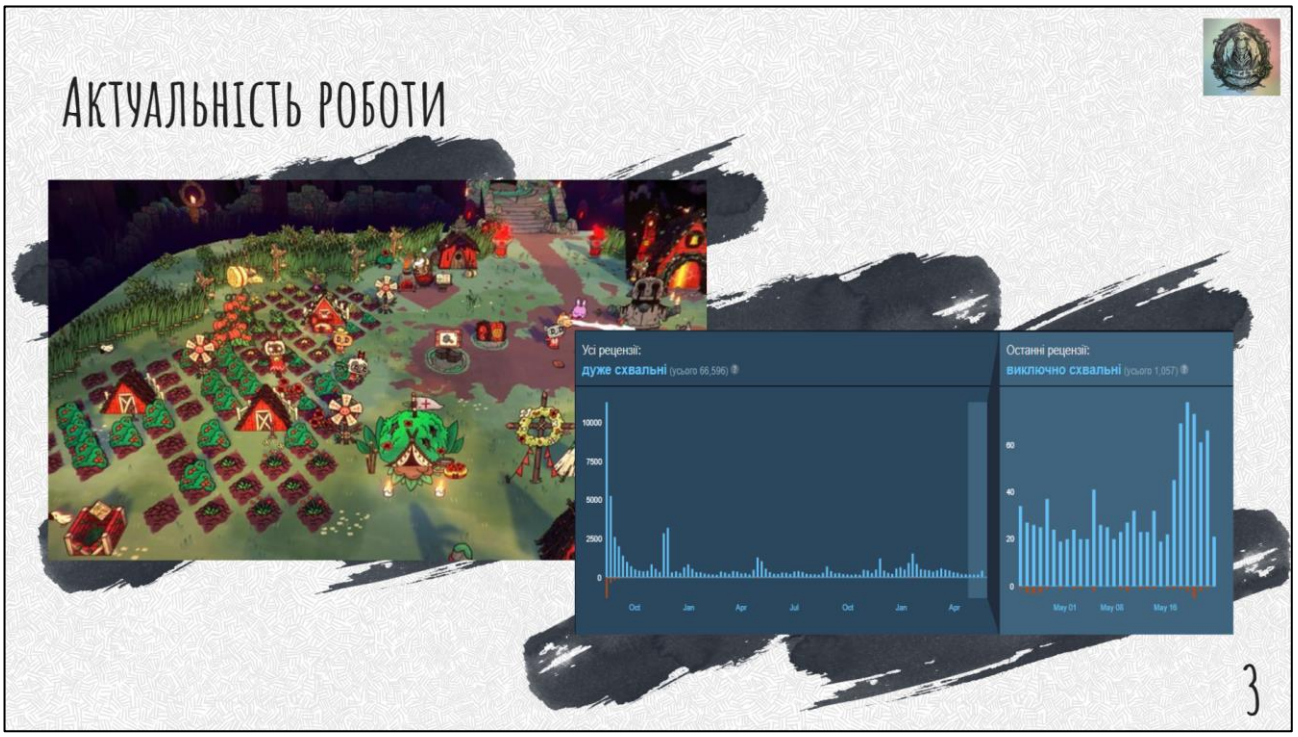


Рисунок М.3 – Слайд №3 презентації

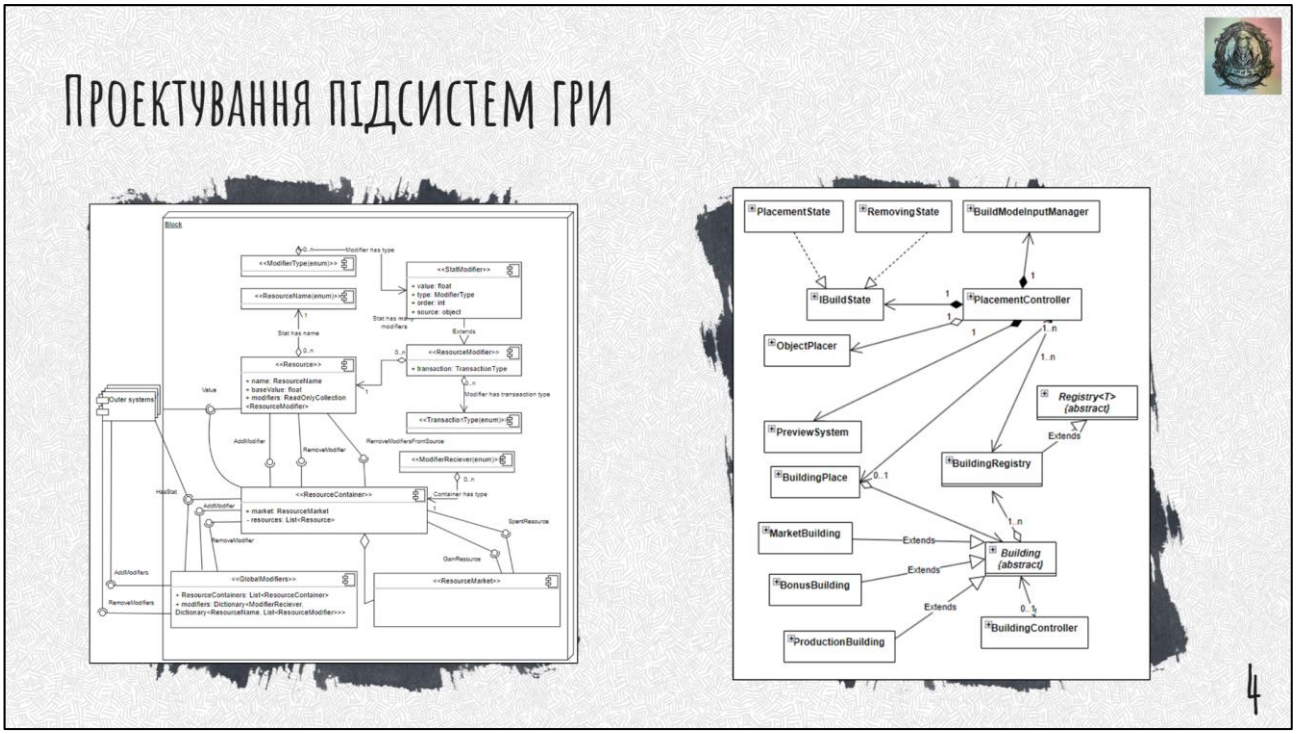


Рисунок М.4 – Слайд №4 презентації

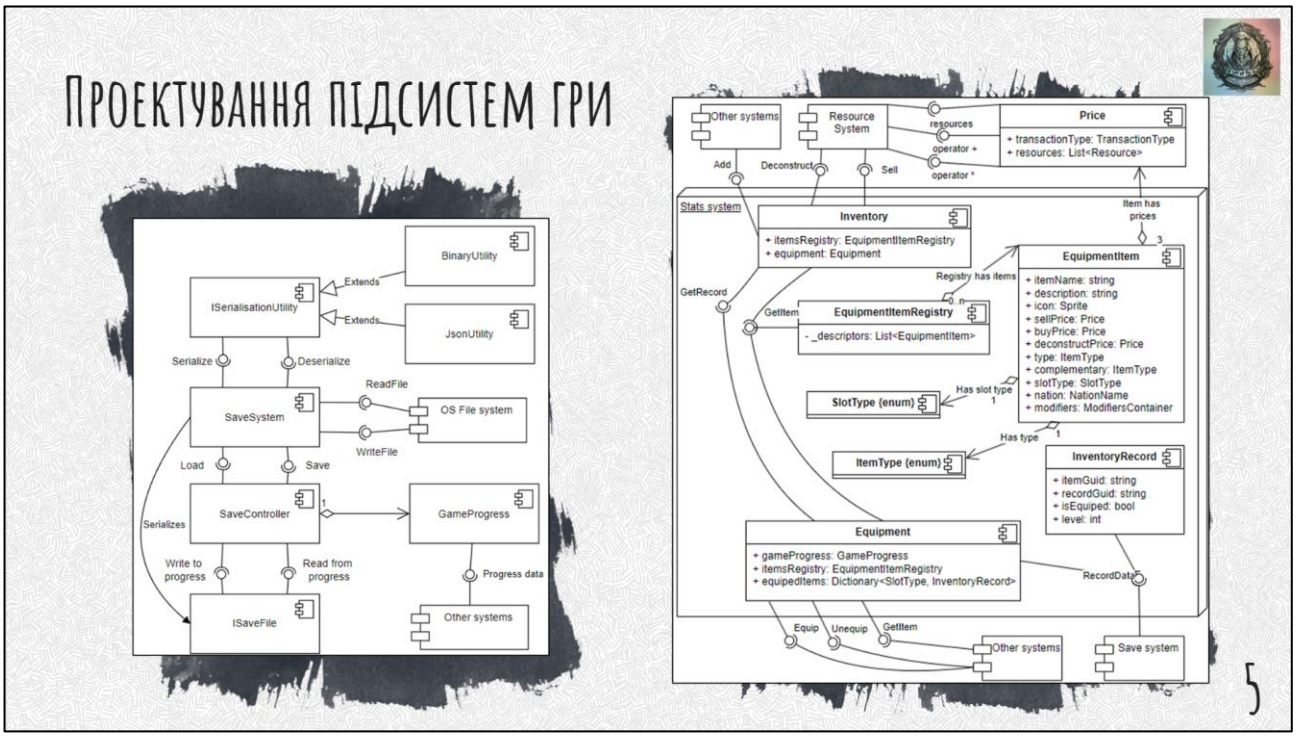


Рисунок М.5 – Слайд №5 презентації

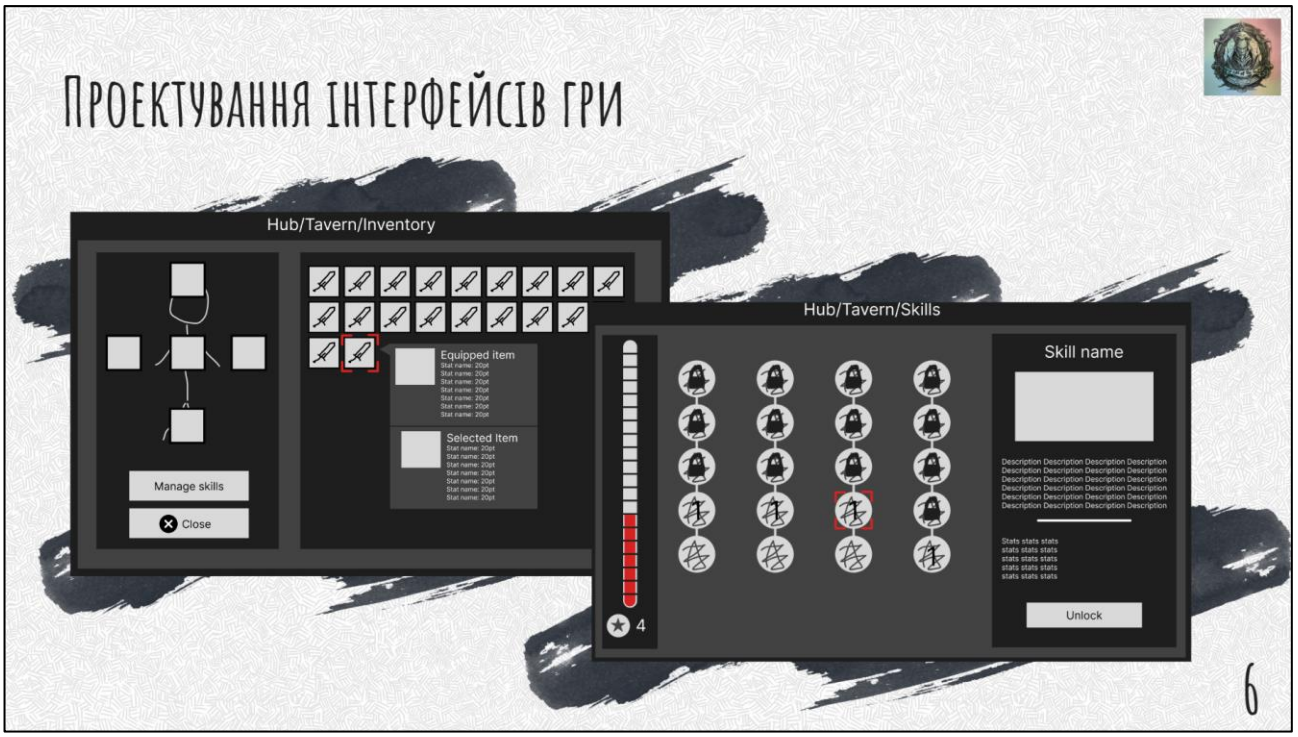


Рисунок М.6 – Слайд №6 презентації

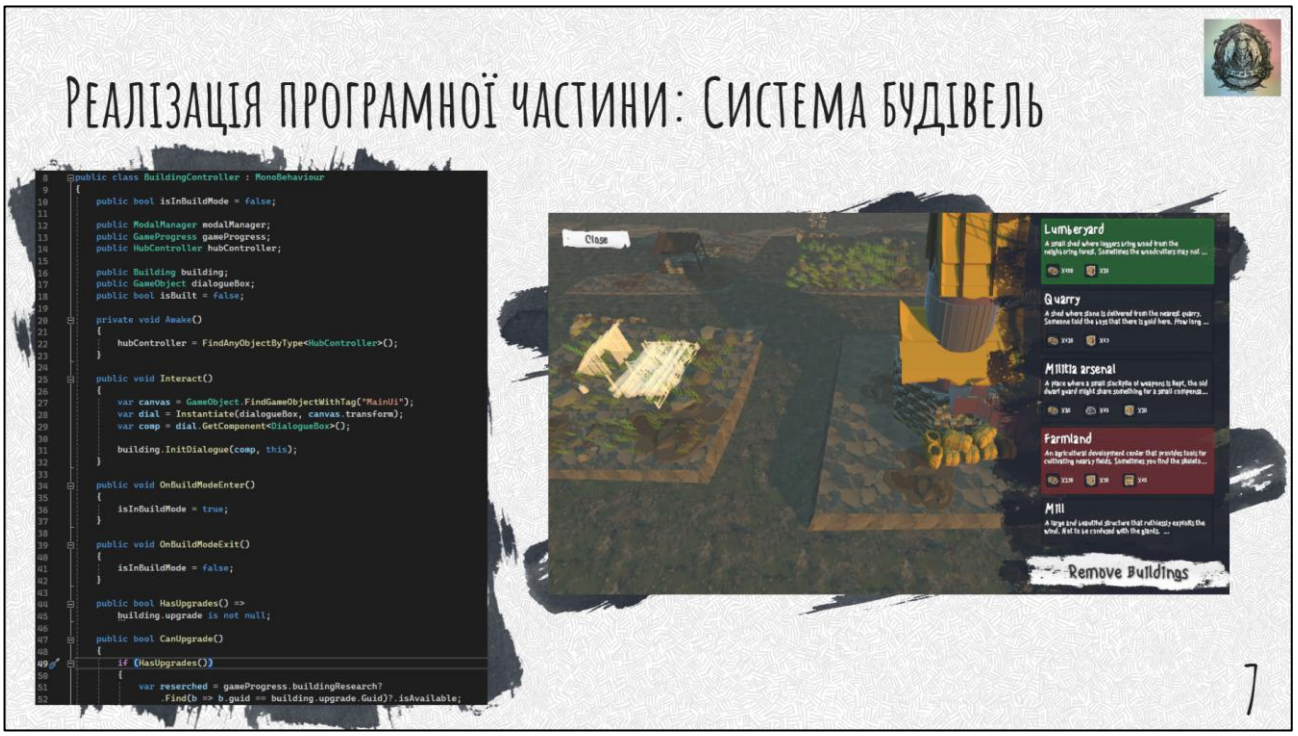


Рисунок М.7 – Слайд №7 презентації

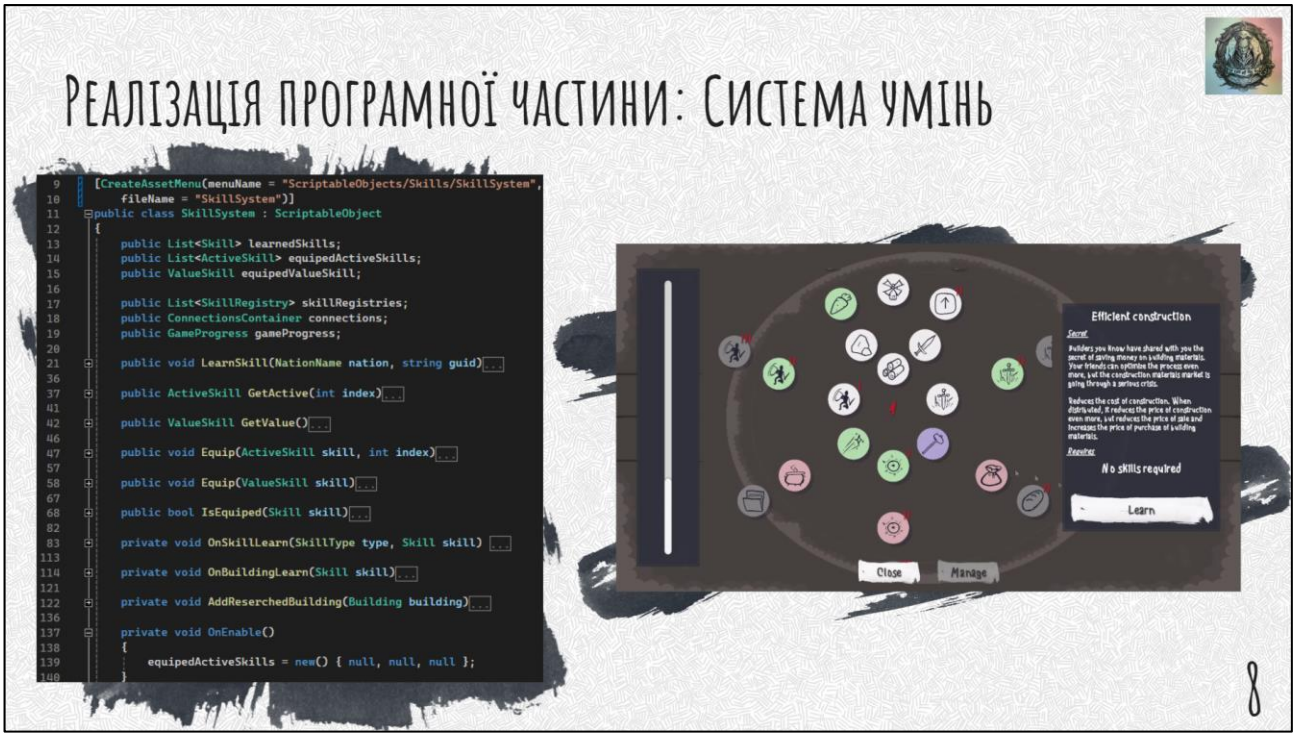


Рисунок М.8 – Слайд №8 презентації

Рисунок М.9 – Слайд №9 презентації

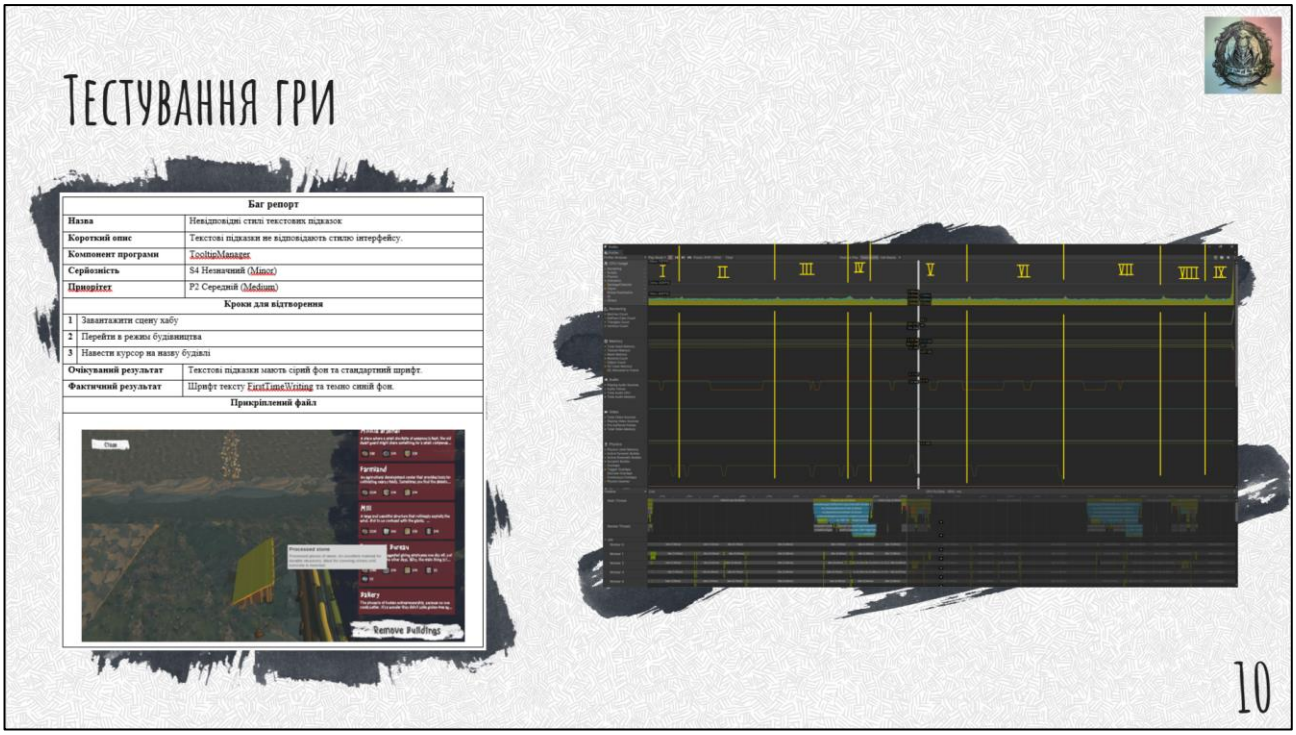


Рисунок М.10 – Слайд №10 презентації

Рисунок М.11 – Слайд №11 презентації

Рисунок М.12 – Слайд №12 презентації

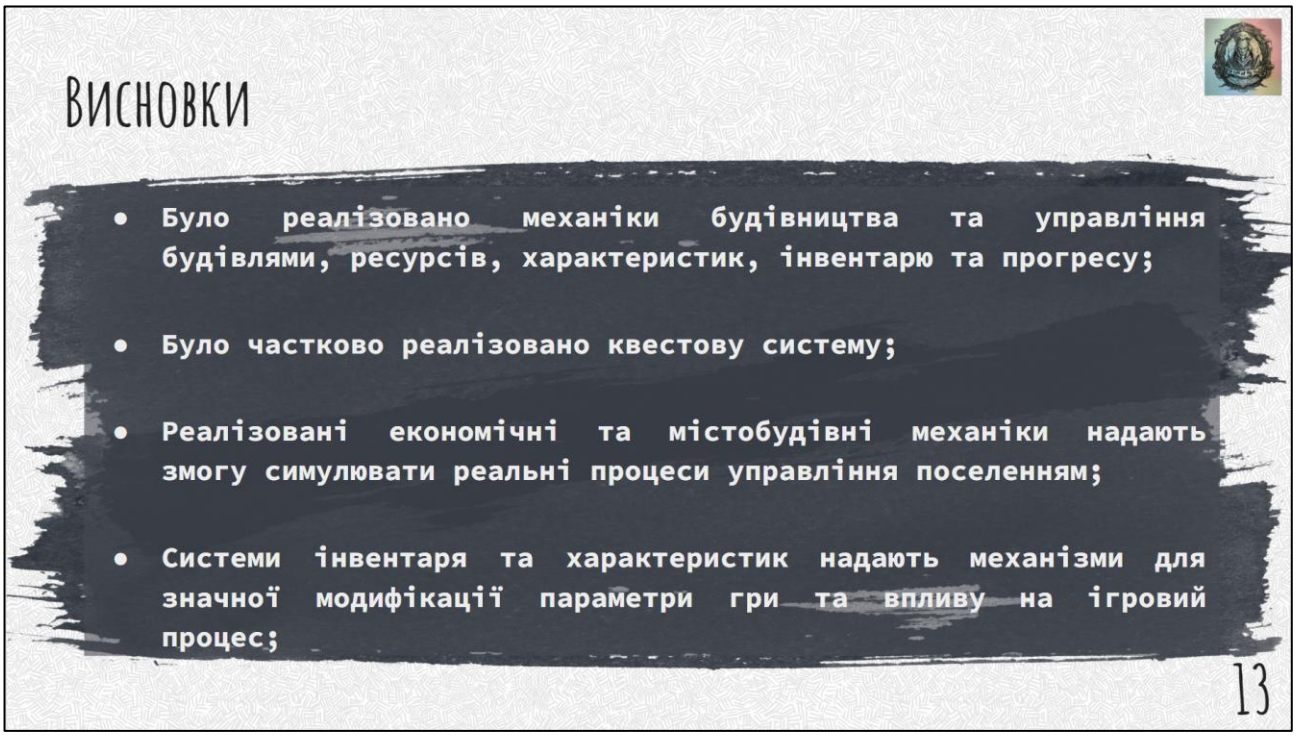


Рисунок М.13 – Слайд №13 презентації

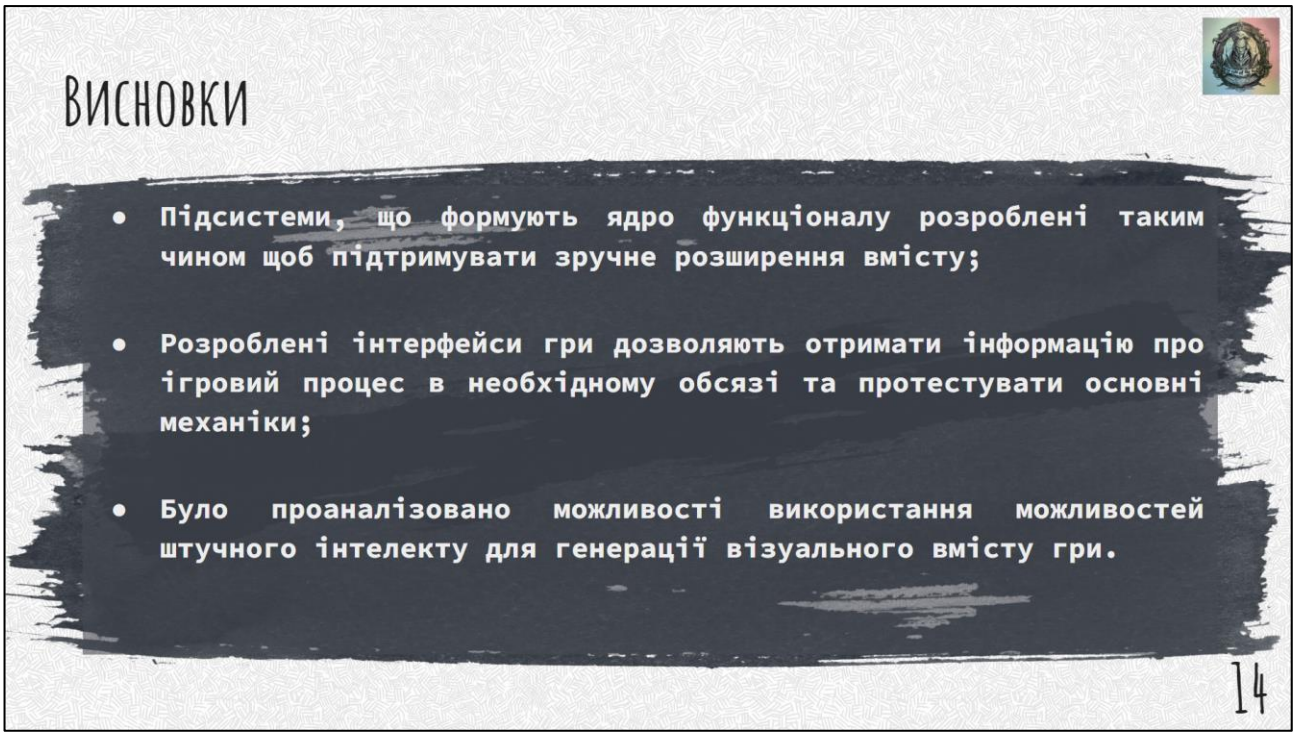


Рисунок М.14 – Слайд №14 презентації

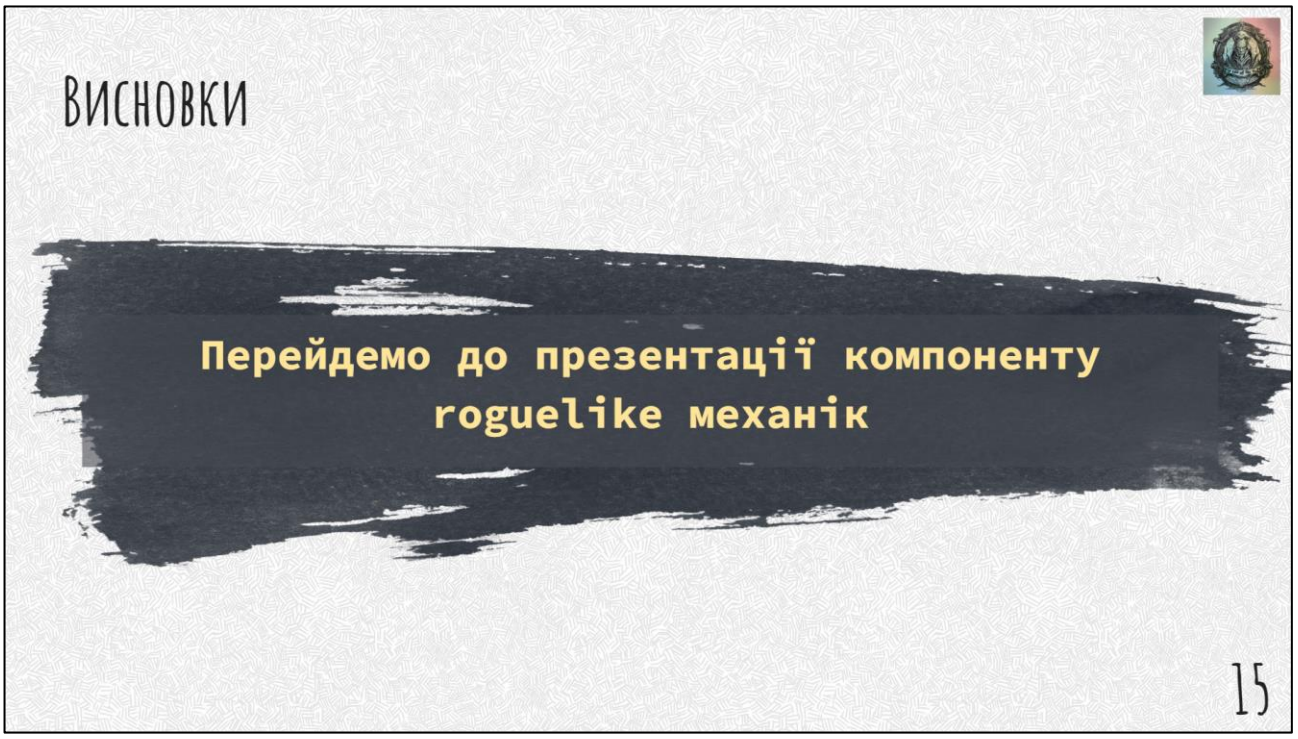
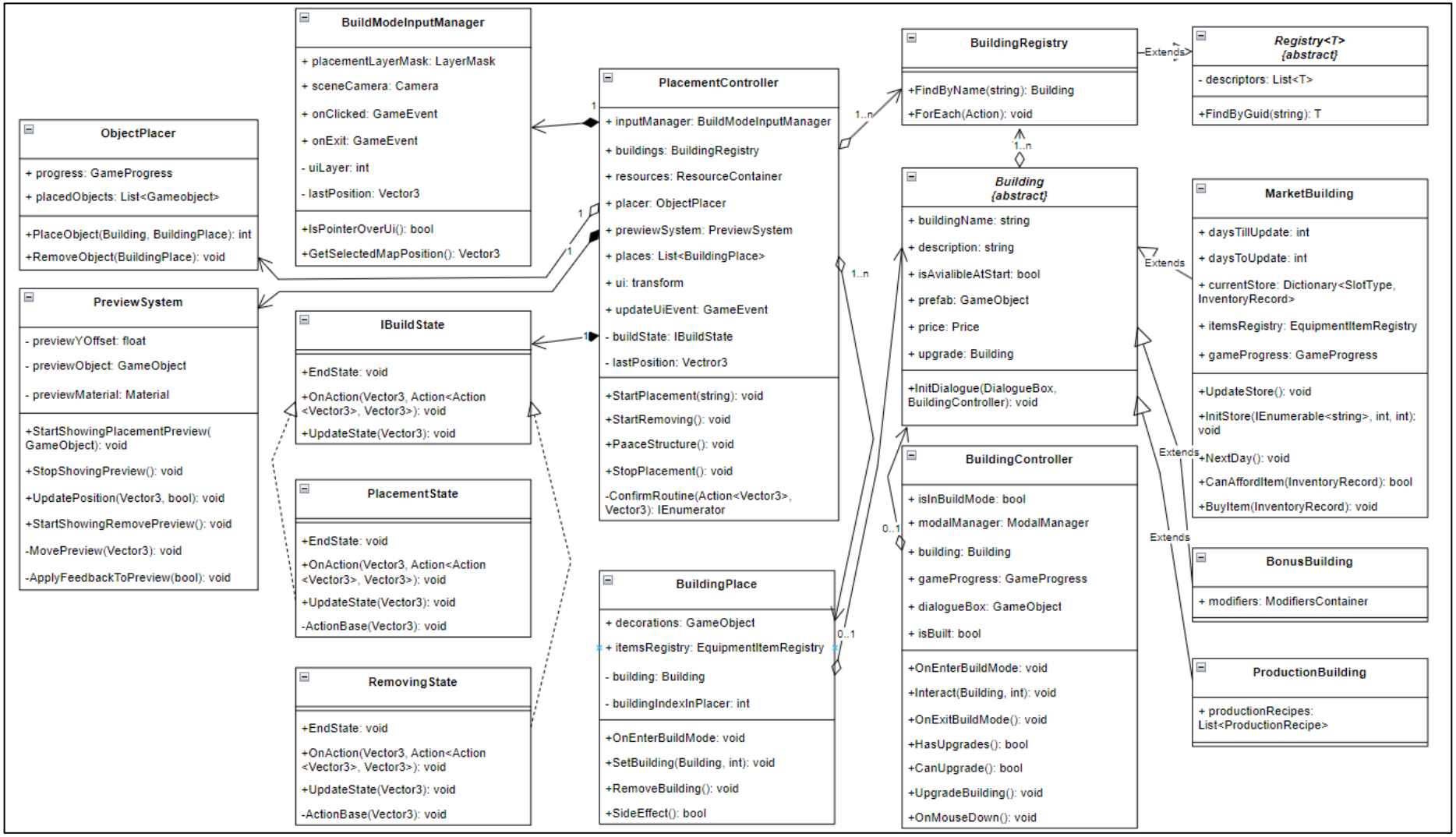


Рисунок М.15 – Слайд №15 презентації

ДОДАТОК В

Детальна діаграма класів системи будівель



ДОДАТОК Г

Вихідний код деяких компонентів системи

Код класу що описує контейнер ресурсів

```
[CreateAssetMenu(fileName = "ResourceContainer",
    menuName = "ScriptableObjects/ResourceSystem/Container")]
public class ResourceContainer : ScriptableObject
{
    [SerializeField]
    private List<Resource> resources;
    private Dictionary<ResourceName,
        Dictionary<TransactionType, ResourceDynamic>> statistics;
    public List<Production> production;
    public ResourceMarket resourceMarket;
    public List<Resource> Resources => resources;

    public Resource GetResource(ResourceName name)
    {
        return resources.Find(r => r.name == name);
    }

    public int GetResourceAmount(ResourceName name)
    {
        return GetResource(name).amount;
    }

    public IEnumerable<string>
        GetProductionBuildingsGuids(ResourceName resource)
    {
        return GetActiveProductions(resource)
            .Select(p => p.buildingGuid);
    }

    public IEnumerable<Production> GetActiveProductions(ResourceName
resource)
    {
        return production.Where(p => p.workers > 0 &&
            p.recipe.resources.FindIndex(r => r.name == resource) >
0);
    }

    public Dictionary<TransactionType, ResourceDynamic>
        GetStatistics(ResourceName resource)
    {
        if (statistics is not null
            && statistics.ContainsKey(resource))
        {
            return statistics[resource];
        } else return new();
    }
```

```
    public void AddStatistics(ResourceName resource,
        Dictionary<TransactionType, ResourceDynamic> record)
    {
        if (statistics is null)
            statistics = new();

        if (!statistics.ContainsKey(resource))
            statistics[resource] = record;
        else
            throw new Exception("there is already statistic data
about: "
                + Enum.GetName(typeof(ResourceName), resource));
    }

    public void GainResource(ResourceName name,
        int amount, TransactionType transactionType)
    {
        var gained = GetResource(name).GainResource(amount,
transactionType);
        InitDict(name, transactionType);

        statistics[name][transactionType].gained += gained;
        statistics[name][TransactionType.Any].gained += gained;
    }

    public void SpendResource(ResourceName name,
        int amount, TransactionType transactionType)
    {
        var spent = GetResource(name).SpendResource(amount,
transactionType);

        InitDict(name, transactionType);

        statistics[name][transactionType].spent += spent;
        statistics[name][TransactionType.Any].spent += spent;
    }

    public void AddResource(Resource resource)
    {
        resources.Add(resource);
    }

    public void AddResources(IEnumerable<Resource> resources)
    {
        this.resources.AddRange(resources);
    }

    public bool CanAfford(Price price)
    {
        foreach (Resource r in price.resources)
        {
            if (!GetResource(r.name).CanAfford(r.amount,
                price.transactionType))
            {
                return false;
            }
        }
```

```
        return true;
    }

    public void Spend(Price price)
    {
        if (CanAfford(price))
        {
            foreach (var r in price.resources)
                SpendResource(r.name, r.amount,
price.transactionType);
        }
        else
            Debug.Log("Can't afford!");
    }

    public void Gain(Price price)
    {
        foreach(var r in price.resources)
            GainResource(r.name, r.amount, price.transactionType);
    }

    public int SpentPerWeek(ResourceName resource)
    {
        int value = 0;
        var res = GetResource(resource);
        var releventProductions = production.Where(p => p.workers > 0
&&
            p.recipe.price.resources.FindIndex(r => r.name ==
resource) > 0);

        foreach (var prod in releventProductions)
        {
            var priceRes = prod.recipe.price
                .resources.Find(r => r.name == resource);

            if (priceRes is not null)
            {
                value += res.ValueWithModifiers(priceRes.amount,
                    TransactionType.Production, false) *
prod.workers;
            }
        }
        value += resourceMarket.GetTransaction(resource).spent;
        return value;
    }

    public int GaindedPerWeek(ResourceName resource)
    {
        int value = 0;
        var res = GetResource(resource);

        foreach (var prod in GetActiveProductions(resource))
        {
            var gain = prod.recipe.resources.Find(r => r.name ==
resource);

            if (gain is not null)
```

```
            {
                value += res.ValueWithModifiers(gain.amount,
                    TransactionType.Production, true) * prod.workers;
            }
        }

        value += resourceMarket.GetTransaction(resource).gained;
        return value;
    }

    private void InitDict(ResourceName resource, TransactionType
transaction)
    {
        if (statistics is null)
            statistics = new();

        if (!statistics.ContainsKey(resource))
            statistics.Add(resource, new());

        if (!statistics[resource].ContainsKey(transaction))
            statistics[resource].Add(transaction,
                new ResourceDynamic { gained = 0, spent = 0 });

        if (!statistics[resource].ContainsKey(TransactionType.Any))
            statistics[resource].Add(TransactionType.Any,
                new ResourceDynamic { gained = 0, spent = 0 });
    }

    private void OnEnable()
    {
        statistics = new();
    }
```

Код класу що описує компонент для динамічного відображення списків

```
public class ListViewController : MonoBehaviour
{
    private List<object> data;
    private object selectedData;
    private IListItem selectedItem;
    private List<IListItem> items;

    public Action selectionChanged;

    public GameObject prefab;
    public GameObject contentParent;
    public float childScale = 1;

    public bool allowSelection = true;

    public object Selected
    {
        get => selectedData;
        set
        {
            selectedData = value;
```

```
            selectedItem = items.Find(i => i.HasData(value));
            selectionChanged?.Invoke();
        }
    }

    public void InitView(List<object> data,
        object selection = null)
    {
        this.data = data;
        items = new List<IListItem>();
        RefreshList();

        Selected = selection;

        if (selection != null)
            ChangeSelection(Selected);
    }

    public void InitView(IListViewExtendedRegistry reg,
        object selection = null)
    {
        data = new();
        reg.ForEach(i => data.Add(i));
        items = new List<IListItem>();
        RefreshList();

        Selected = selection;

        if (selection != null)
            ChangeSelection(Selected);
    }

    public void RefreshList(List<object> newData = null)
    {
        items.Clear();

        foreach (Transform child in contentParent.transform)
            Destroy(child.gameObject);

        if (newData != null)
            data = newData;

        data.ForEach(item =>
        {
            var obj = Instantiate(prefab, contentParent.transform);
            var comp = obj.GetComponent<IListItem>();

            obj.transform.localScale *= childScale;

            if (allowSelection)
                comp.Selection += () => ChangeSelection(item);

            comp.FillItem(item);
            items.Add(comp);
        });

    }
```

```
    public void ChangeSelection(object obj)
    {
        Selected = data.Find(i => i == obj);

        selectedItem?.OnSelected();
    }
}
```

Код класу модального вікна

```
public class DialogueBox : TooltipDataProvider
{
    [NonSerialized]
    public string header;
    [NonSerialized]
    public string body;

    public List<GameObject> buttons;

    [NonSerialized]
    public DialogueBoxResult result = DialogueBoxResult.None;

    public TextMeshProUGUI headerSection;
    public TextMeshProUGUI bodySection;
    public GameObject buttonsSection;

    public bool discardOtherWhileOpen = false;
    public bool allowInBuildMode = false;
    public bool ignoreConstraints = false;
    public bool requireResultToClose = false;

    public ModalManager modalManager;

    private void Awake()
    {
        gameObject.SetActive(false);
    }

    public virtual bool InitDialogue()
    {
        if (ignoreConstraints)
            Init();
        else if (modalManager.isDiscarding ||
            (modalManager.isInBuildMode && !allowInBuildMode))
        {
            DestroyDialogue();
            return false;
        }
        else
        {
            Init();
        }

        return true;
    }
```

```
    public virtual void EndDialogue()
    {
        //if (!ignoreConstraints)
        //runtimeData?.DialogueClose();

        if (requireResultToClose && result == DialogueBoxResult.None)
            return;

        if (discardOtherWhileOpen)
            modalManager.isDiscarding = false;

        gameObject.SetActive(false);

        Invoke(nameof(DestroyDialogue), 2);
    }

    protected virtual void DestroyDialogue()
    {
        Destroy(gameObject);
    }

    public override string GetHeader(string tag = null)
    {
        return header;
    }

    public override string GetContent(string tag = null)
    {
        return body;
    }

    private void Init()
    {
        gameObject.SetActive(true);

        headerSection?.SetText(header);
        bodySection?.SetText(body);

        foreach (var button in buttons)
        {
            var btnObj = Instantiate(button.gameObject,
buttonsSection.transform);
            var btn = btnObj.GetComponent<Button>();
            var dialogueBtn = btnObj.GetComponent<DialogueButton>();

            btn.onClick.AddListener(() => result =
dialogueBtn.result);
            btn.onClick.AddListener(() =>
modalManager.DialogueClose());
        }

        modalManager.isDiscarding = discardOtherWhileOpen;
    }
}
```

ДОДАТОК Д

Майстер тест-план

The fate of the liar

План випробувань і тестування
(Test Plan)

Версія 1.0
(Version 1.0)

Revision History

Date	Version	Description	Author
24.02.2024	1.0	Початкова версія	Прудіус В. Ю., Долгий А. С.

1. Introduction

1.1. Purpose

Мета цього документу - це забезпечення систематичного підходу до тестування функціоналу гри Fate of the liar, що пов’язаний з механіками розвитку персонажа та поселення, щоб впевнитися в її якості та функціональності перед випуском на ринок або перед офіційним представленням. Основні цілі включають:

- Переконатися, що всі основні функції гри працюють належним чином, включаючи економічні моделі, управління ресурсами, торгівлю, дипломатію тощо.
- Визначити стабільність гри шляхом виявлення та вирішення будь-яких помічених проблем або збоїв, таких як затримки, краші або витоки пам'яті.
- Перевірити рівень складності та ігровий досвід, включаючи перевірку наявності достатньої кількості викликів для гравців на різних рівнях майстерності.
- Переконатися, що гра відповідає стандартам якості, включаючи графіку, звук, інтерфейс користувача та загальний відчутний ігровий досвід.

Цей документ буде використовуватися для організації та виконання тестування гри з метою забезпечення її успішного впровадження її базових механік та надання бази для подальшого зростання проекту без відчутних проблем.

1.2. Background

Тестування проводитиметься для гри Fate of the liar, що є рольовою грою з елементами механік економічних стратегій та ігор симейства roguelike. Тестуватись будуть основні механіки пов’язані з економічними процесами в грі, а також механіки що сприяють розвитку персонажа та його поселення.

Механіки стратегічних ігор, що підлягають тестуванню представлені у вигляді управління ресурсами та поселенням що доповнюють механіки

управління інвентарем, розвитку персонажа та базова реалізація механік квестової системи, зокрема її частини, що пов’язані з нарахуванням нагород.

Більш детально з інформацією про проект можна ознайомитись у ГДД за посиланням: https://docs.google.com/document/d/1Y9_I0Mn20nhqRkoJ-gg1L4RRCzr9LS4yb1twfXA1SHc/edit?usp=sharing

1.3 Scope

Цей документ має застосовуватись для систематизації підходу до процесу тестування основних механік гри на ранніх етапах розробки програмного продукту.

1. В процесі тестування мають бути перевірені наступні елементи гри:
1. Користувацький інтерфейс;
2. Коректність роботи впроваджених ігрових механік:
    a. Створення персонажа;
        i. Ручне
        ii. Полегшене
    b. Механік поселення;
        i. Система ресурсів
        ii. Система будівель
        iii. Система будівництва
        iv. Система ринків
        v. Система квестів (економічна частина)
    c. Механік розвитку персонажа;
        i. Система предметів та інвентаря
        ii. Система екіпірування
        iii. Система розвитку
        iv. Система характеристик

1.4 Project Identification

У таблиці 1.4.1 наведено документацію та її готовність, для розробки плану тестування.

Document	Created	Revised	Author	Note
Гейм дизайн документ v1.2	13.09.2023	24.02.2024	Прудіус В. Ю., Долгий А. С.	-
Таблиця контенту v1.1	12.11.2023	24.02.2024	Прудіус В. Ю.	-

2. Requirements for Test

Перелік нижче формує детальний список функцій що мають бути протестовані:

1. створення нового персонажа з вказанням такої інформації:
    a. ім’я;
    b. стать;
    c. аватар, іконка персонажа у грі;
    d. розподіл характеристик персонажа за 5 основними групами
2. спрощений процес створення персонажа, що включає в себе вибір одного з запропонованих досьє, що були згенеровані штучним інтелектом з попередньо розподіленими значеннями;
3. управління ресурсами, що включає в себе:
    a. перегляд наявних ресурсів;
    b. перегляд статистики по використанню та отриманню ресурсів;
    c. ринок ресурсів;
    d. вільне використання ресурсів на різні потреби (купівля спорядженння, будівництво, тощо);
4. управління поселенням, що включає в себе:
    a. перегляд поселення в режимі вільної камери;
    b. перегляд можливостей для будівництва;
    c. розміщення нових будівель;
    d. знесення старих будівель;
    e. взаємодія з розміщеними будівлями;
5. управління інвентарем, що включає:
    a. перегляд наявного екіпірування, та його деталей;
    b. управління екіпіруванням що використовується ігровим персонажем;
    c. ринок екіпірування, що включає розбір, продаж та купівлю екіпірування;
6. відстеження розвитку персонажа, що включає:

   a. перегляд прогресу, та визначення впливу основних характеристик прогресу: рівнем відомості, рівнем лояльності долі та взаємовідносин з народами;
   b. перегляд основних характеристик персонажа;
   c. перегляд та розподіл очок по деревам розвитку кожного з народів;
7. впровадження системи завдань, що включає в себе:
   a. перегляд наявних завдань, відслідковування обраних та відмова від їх виконання;
   b. відслідковування прогресу проходження квестів;
   c. видача нових квестів та нарахування винагороди при виконанні завдань;
8. збереження та завантаження ігрового прогресу;

Тестована функціональність має перевірятись на відповідність вимогам та описанню ігрового процесу що зазначені у ГДД, посилання на яке є у попередніх розділах документу.

3. Test In Time

Тестування програмної системи проводитиметься на наступних рівнях:

1. компонентне тестування - окреме тестування функцій різних компонентів програми.
2. інтеграційне тестування - перевірка правильності взаємодії між підсистемами.
3. системне тестування - перевірка усієї композиції тестованих механік для виявлення загальних комплексних проблем з продуктом.

Для тестування програмної системи використовуватимуться такі види тестування:

- Функціональне тестування
- Тестування взаємодії
- Тестування продуктивності
- Тестування зручності використання
- Тестування користувацького інтерфейсу

3.1 Testing Types

3.1.1 Functional testing

Цілі:

- Перевірка того, що всі основні функції гри працюють належним чином, включаючи управління ресурсами, торгівлю, дипломатію, дослідження, будівництво тощо.
- Перевірка правильності реалізації різних ігрових механік, таких як управління камерою, взаємодії, економічні моделі та інші ігрові аспекти.
- Тестування коректності вводу та виводу даних. Перевірка правильності обробки введених даних гравцем та коректності відображення результатів гри.
- Визначення реакції на некоректний ввід. Тестування гри на виявлення некоректного введення або невірного використання, і визначення того, як гра реагує на такі ситуації.

Опис процесу:

- Експлораторне тестування. Активне дослідження гри для виявлення непередбачених сценаріїв та проблем;
- Вибір функцій та компонентів які мають бути протестовані, та відтворення вводу та команд, що імітують різноманітні ігрові ситуації;
- Тестування у реальному середовищі. Відтворення реальних умов гри для перевірки реакції програмного забезпечення на різні фактори, такі як мережеві затримки або обмеження ресурсів апаратного забезпечення;
- Тестування користувацького досвіду. Оцінка вражень та виявлення можливих покращень шляхом активної взаємодії з компонентами гри.

Критерії виходу:

- Компонент гри пройшов тестування без виявлення критичних помилок.
- Виявлені критичні помилки в роботі додатку і необхідне доопрацювання
- Реалізований функціонал не відповідає очікуванням користувачів чи специфікації, компонент повертається на доопрацювання

3.1.2 Integration testsing

Цілі:

- Перевірка того, що всі тестовані компоненти гри взаємодіють у відповідності до їх специфікації.
- Перевірка правильності реалізації різних ігрових механік, на перетині різних ігрових підсистем, таких як система ресурсів, будівництва, розвитку тощо.
- Тестування коректності вводу та виводу даних. Перевірка правильності обробки введених коректних даних гравцем, в контексті взаємодії між підсистемами гри.
- Визначення реакції на некоректний ввід. Тестування гри на виявлення некоректного введення або невірного використання функціоналу різних підсистем гри.

Опис процесу:

- Експлораторне тестування. Активне дослідження гри для виявлення непередбачених сценаріїв та проблем;
- Вибір компонентів які мають бути протестовані, та відтворення вводу та команд, що імітують різноманітні ігрові ситуації для перевірки взаємодії підсистем гри;
- Тестування у реальному середовищі. Відтворення реальних умов гри для перевірки реакції програмного забезпечення на різні фактори, такі як мережеві затримки або обмеження ресурсів апаратного забезпечення;
- Тестування користувацького досвіду. Оцінка вражень та виявлення можливих покращень шляхом активної взаємодії різними з механіками що включають в себе використання різних ігрових підсистем.

Критерії виходу:

- При тестуванні взаємодії між компонентами не виявлено критичних помилок
- Тестування виявило критичні промилки у взаємодії або в одному з компонентів, необхідне доопрацювання
- Тестування виявило проблеми пов’язані невідповідністю взаємодії між компонентами з точки зору специфікації та запланованого ігрового досвіду, необхідне доопрацювання

3.1.3 Performance testing

Цілі:

- Оцінка продуктивності. Вимірювання частоти кадрів (FPS), часу завантаження та інших показників продуктивності гри при різних умовах;
- Перевірки масштабованості. Визначення того, як гра працює при різних рівнях навантаження, включаючи велику кількість 3D моделей, важкий рендеринг або складні сцени;
- Виявлення проблем з ресурсами. Виявлення проблем з використанням пам'яті, процесором або іншими ресурсами системи під час гри.

Опис процесу:

- Тестування гри з використанням складних обчислень за допомогою застосування складного функціоналу, запуску важкого рендерингу чи відтворення складних сцен;
- Запуск гри на довгий час для виявлення проблем пов’язаних з виділенням пам'яті або використанням ресурсів, протягом довгого періоду активності додатку.
- Тестування гри в нормальному режимі для порівнянь характеристик та показників, а також визначення проблемних моментів.

Критерії виходу:

- Стабільна продуктивність;
- Відсутність проблем з ресурсами;
- Масштабованість.

3.1.4 Convenience testing

Цілі:

- Оцінка зручності використання тестованих механік в ігровому процесі;
- Перевірка визначених методів взаємодії з механіками та визначення їх зручності;
- Виявлення проблем пов’язаних з комплектацією та розміщенням графічних елементів користувацького інтерфейсу в процесі використання тестованих механік.

Опис процесу:

- Перевірка можливості використання тестованих механік в середині гри;
- Оцінка зручності використання тих чи інших механік.
- Тестування гри в нормальному режимі для визначення моментів які виділяються чи викликають особливі проблеми при взаємодії.
- Визначення дій та механік, на використання яких витрачається найбільше часу та оцінка впливу якості інструментів взаємодії в цьому контексті.

Критерії виходу:

- Не виявлено значних відхилень від запланованих норм;
- Виявлено деякі невідповідності та проблеми зі зручністю тестованих компонентів, необхідне доопрацювання.

3.1.5 User interface testing

Цілі:

- Оцінка відповідності тих чи інших візуальних елементів користувацького інтерфейсу специфікаціям;
- Перевірка відповідності різних графічних елементів тим функціям які закладені в їх основу;
- Виявлення проблем пов’язаних з реалізацією тих чи інших візуальних елементів.
- Виявлення невідповідностей тих чи інших елементів інтерфейсу визначеному візуальному стилю проекту

Опис процесу:

- Активна взаємодія з тестованими елементами інтерфейсу гри;
- Порівняння тестованих компонентів інтерфейсу з макетами розробленими на етапі проектування та перевірка відповідності специфікаціям.
- Тестування гри в нормальному режимі для визначення моментів які виділяються чи викликають дисонанс на загальному плані.

Критерії виходу:

- Не виявлено значних відхилень від запланованих норм;
- Виявлено невідповідності щодо загального дизайну чи макету, необхідне доопрацювання ;
- Елементи інтерфейсу не відповідають цілям які перед ними ставляться або не виконують поставлену задачу відповідно до специфікації, необхідне додаткове доопрацювання.

4. Resources
4.1. Roles

Наступна таблиця показує приблизні потреби у людських ресурсах для повноцінного функціонування команди розробки та проведення тестування та виправлення помилок знайдених в цьому процесі.

Human Resources		
Worker	Minimum Recommended number of full-time roles allocated	Specific Responsibilities or Comments
Developer	2	- розробка програмних рішень відповідно до вимог та специфікацій завдань. - первинне тестування розроблених компонентів та виправлення помилок.
PM	1	- керування процесами розробки - ведення документації.
Artist	2	- розробка візуального контенту для гри, (3д моделі, зображення).
QA	1	- Виконання тестування програмного забезпечення для виявлення помилок та дефектів.
UI/UX Designer	1	- проектування та покращення користувацьких інтерфейсів.

4.2. Система (System)

Наступна таблиця націлена на формування контексту тестування та контексту впровадження продукту що тестується стосовно технічного та програмного забезпечення.

System Resources	
Resource	Note
Testing client version	v. 1.0
Include special configuration requirements	Minimum Requirements: ОС: Windows 7 x64 or later Процесор: Intel Core i3-3240 (2 * 3400); Оперативна пам’ять: 4GB ОП Відеокарта: GeForce GTX 560 Ti (1024 VRAM); Radeon HD 7750 (1024 VRAM) Місце на диску: 4GB доступного місця
	Recommended Requirements: ОС: Windows 10 x64 Процесор: Intel Core i5-3470 Оперативна пам’ять: 8GB ОП Відеокарта: GeForce GTX 1050 (2048 VRAM); Radeon R9 380 (2048 VRAM) Місце на диску: 4GB доступного місця
Test Development PC's	ОС: Windows 11 DirectX: 11 Процесор: Intel(R) Core i5-12400F Оперативна пам’ять: 32GB Відеокарта: NVIDIA GeForce RTX 3060

5. Project Milestones

Тестування проводитиметься поетапно, до моменту в часі коли тестування не виявить проблем. Кожен етап включатиме повний цикл від підготовки та планування, до виконання та оновлення документації. Наступна таблиця візуалізує кожен з пунктів циклу проведення тестування.

Milestone Task	Effort	Start Date	End Date	Note
Plan Test	1 д	22.02.2024	23.02.2024	Загальне планування тесту
Design Test	2д	23.02.2024	25.02.2024	Детальне планування
Implement Test	2д	25.02.2024	27.02.2024	Підготовка матеріалів для тестування
Execute Test	7д	27.02.2024	5.03.2024	Виконання тестування
Evaluate Test	3д	5.03.2024	8.03.2024	Оцінка результатів та ведення документації

6. Deliverables
6.1. Review

Результатами проведення кожного з етапів тестування має бути тест план етапу, що детально описуватиме конкретні функції що тестуються, підходи та масштаби тестування а також контекст тестування. Поряд з тест планом етапу, результатами тестування також є тест-кейси, що у формалізованому детальному вигляді визначають які проблеми було виявлено, в яких компонентах програми, та яким чином ці проблеми можна виправити.

ДОДАТОК Е

Приклад заповненого тест-кейса

Інформація про тест-кейс			
Ідентифікатор	TAA0008		
Назва тесту	EquipItemOnOccupiedSlot		
Власник тесту	Прудіус Влад		
Місцезнаходження тесту(шлях)	TestServer: D:\FotL\Testing\TestCases\TAA0008.doc		
Дата останнього перегляду	13/05/2024		
Технічна вимога, що тестується	- екіпірування предмету - відміна екіпірування вже використаного предмету - відміна екіпірування суміжного предмету - неможливість екіпірувати несуміжний аксесуар		
Конфігурація засобів тестування	базова		
Взаємозалежність тестових випадків	Виконати прогін тесту і його налаштування перед прогоном даного тесту.		
Мета тесту	Перевірити те, що при екіпіруванні предмету в слот що вже зайнятий, поточний предмет займає його місце, і усі суміжні предмети, що не підходять поточному, також знімаються.		
Методика тестування			
Налаштування	Перед тестуванням модифікувати файли збереження таким чином щоб: - в інвентарі знаходилось мінімум 2 предмети зброї різного типу - в інвентарі знаходився мінімум 1 аксесуар суміжний не більше ніж з одним предметом зброї - предмет зброї та суміжний до нього аксесуар екіпіровано		N/A *
Крок	**Дія**	**Очікуваний результат**	**Відмітка(V)***
1.	Натиснути на тривимірну модель таверни.	Відкривається вікно таверни.	(V)
2.	Натиснути на кнопку інвентаря	Відкривається вікно інвентаря, в якому видно список предметів та видно які з них екіпіровано.	(V)
3.	Перетягнути не екіпірований предмет в зону екіпірування	Предмет в інвентарі і зоні екіпірування відмічається як одягнутий, предмет що був у відповідному слоті та аксесуар перестають відмічатись як одягнуті.	(V)
4.	Перетягнути в зону екіпірування аксесуар що не суміжний з	Предмет залишається екіпірованим, аксесуар	(V)

	попереднім предметом	залишається не екіпірованим.	
5.	Перетягнути в зону екіпірування попередній предмет	Предмети обмінюються станами, новий предмет екіпіровано, старий знято з відповідними індикаціями у інвентарі і екпіруванні.	(V)
6.	Перетягнути в зону екіпірування аксесуар що суміжний з попереднім предметом	Аксесуар екіпіровано як і предмет	(V)
Очистка	Очистити попередньо додані дані з файлу збереження. Видалити нові файли збережень		
Результати тесту			
Тестувальник: Прудіус Влад	**Дата прогону тесту:** 13/05/2024		**Результат тесту(P/F/B): ПРОЙДЕНО (P)**

ДОДАТОК Ж

Приклад баг-репорту

Баг репорт	
Назва	Невідповідні стилі текстових підказок
Короткий опис	Текстові підказки не відповідають стилю інтерфейсу.
Компонент програми	TooltipManager
Серйозність	S4 Незначний (Minor)
Приорітет	P2 Середній (Medium)
Кроки для відтворення	
1	Завантажити сцену хабу
2	Перейти в режим будівництва
3	Навести курсор на назву будівлі
Очікуваний результат	Текстові підказки мають сірий фон та стандартний шрифт.
Фактичний результат	Шрифт тексту FirstTimeWriting та темно синій фон.
Прикріплений файл	

ДОДАТОК И

Матеріали публікацій пов’язаних з роботою

Рисунок К.1 – Сертифікат учасника конференції

Рисунок К.2 – Зміст видання конференції

December 3, 2023 • Berlin, Federal Republic of Germany • Collection of scientific papers «SCIENTIA»

Прудіус Владислав Юрійович
здобувач вищої освіти факультету комп'ютерних наук
Харківський національний університет радіоелектроніки, Україна

ПЕРСПЕКТИВИ ВИКОРИСТАННЯ ГЕНЕРАТИВНОГО ШТУЧНОГО ІНТЕЛЕКТУ В РОЗРОБЦІ ІГРОВИХ ЗАСТОСУНКІВ

Загальним світовим трендом за останні кілька років є потужний стрибок у розвитку загальнодоступних мовних моделей, на основі яких будуються різноманітні, більш складні інструменти, що покликані перетворити запит користувача на різного роду контент.

Впровадження штучного інтелекту у різні сфери діяльності, може значно підвищити ефективність роботи. Так згідно з дослідженнями, проведеними Nilsen Norman Group [1], працівники що відповідають за роботу з клієнтами, у середньому підвищують свою продуктивність на 13.8% при використанні штучного інтелекту на робочому місці. Хоча слід зазначити, незначний приріст, у якості виконаної роботи (приблизно 1.3%), та значний розбіг у значеннях відповідно до попередньої ефективності працівника (найбільш ефективні працівники отримували приріст лише у декілька відсотків, в той час, коли найменш ефективні могли покращити свої показники у 2.5 рази).

Але як щодо сфер, де штучний інтелект, у різних його проявах, використовується вже протягом багатьох років. Зокрема найбільше нас цікавить розробка ігрових застосунків, адже одні з перших, найбільш примітивних моделей штучного інтелекту були використані в іграх ще у 80-х, а широкого розповсюдження набули у 2000-х [2]. Тож поглянемо як сучасні генеративні мовні моделі можуть стати в пригоді в цій індустрії.

Для початку розглянемо, яким чином штучний інтелект використовується в сучасній ігровій розробці. Найбільш очевидним варіантом є використання моделей штучного інтелекту для управління не ігровими персонажами (NPC). Це найбільш старий випадок використання, що дозволяє урізноманітнити ігровий процес, надати доступ до більш реалістичного ігрового досвіду та коригувати складність гри. Але чи можуть сучасні мовні моделі (на кшталт ChatGPT або Bard) бути тут корисними? Якщо враховувати пряме використання, то відповідь ні, через різну цільову спрямованість інструментів. Хоча опосередковано мовні моделі можуть допомагати у навчанні іншого штучного інтелекту, пропонувати алгоритми та підтримувати розробника на цьому шляху.

У ще одному розповсюдженому сценарії використання – процедурній генерації, ситуація дещо схожа. В загальному випадку стандартні мовні моделі, без додаткових інструментів складно використовувати для цього сценарію. Винятками є досить прості алгоритми, які приймають форматований текстовий ввід для генерації, хоча навіть в цьому випадку треба приділити достатньо часу для навчання моделей правильному сприйняттю та формуванню виводу. При цьому, навіть велика кількість вкладених зусиль, не гарантує стовідсоткового показника успішності використання, що також слід брати до уваги.

Далі перейдемо до сценаріїв використання сучасного генеративного штучного інтелекту напряму [3]. Використання цих інструментів для впровадження систем взаємодії між гравцем і не ігровими персонажами. Це один з найбільш цікавих та перспективних шляхів застосування мовних моделей, адже таким чином ігри що містять розгорнуті діалогові системи можуть продукувати з високою долею вірогідності унікальний досвід, не лише на рівні кожного окремого гравця, а й усієї гри в цілому. Ще одним плюсом у використанні таких інструментів, є значне зниження часу, потрібного наративним

Рисунок К.3 – Текст першої сторінки публікації

дизайнерам на розробку контенту для діалогових систем, але водночас з'являється додаткове навантаження необхідне для правильного налаштування та навчання нейронних мереж.

Ще одним сценарієм є обробка та генерація графічного контенту. На основі базових мовних аналітичних моделей створено багато інструментів, за допомогою яких можна перетворити простий текстовий запит на зображення. Використовуючи ці інструменти, можна покращувати, модифікувати та генерувати контент для впровадження в ігри. Це особливо актуально для невеликих студій з обмеженими ресурсами, адже значно пришвидшує генерацію контенту графічними дизайнерами, або також, дозволить перенести їх обов'язки на інших членів команди без особливої втрати в якості, але з необхідністю в доопрацюванні.

Також слід згадати низку сценаріїв притаманних усій IT індустрії, зокрема генерація коду, поради та допомога в проєктуванні та прийнятті рішень, аналіз текстових даних, робота по зв'язку з клієнтами тощо. Кожен з описаних вище варіантів використання сучасних мовних моделей також актуальний і для ігрової розробки.

Отже, перейдемо до висновків. У світі зростаючого застосування штучного інтелекту, особливий акцент робиться на розвиток загальнодоступних мовних моделей, що відкриває нові перспективи.

У контексті ігрової розробки, сучасні генеративні мовні моделі виявляються корисними для різних сценаріїв використання, таких як покращення імерсивності та генерація графічного контенту на основі текстових запитів, спрощуючи процес розробки для проєктів різного об'єму.

Однак, варто відзначити, що використання стандартних мовних моделей у деяких сценаріях, зокрема процедурній генерації, може бути досить складним і вимагати додаткового часу та зусиль для налагодження та навчання. Тому в деяких випадках від використання мовних моделей слід відмовитись, на користь більш спеціалізованих інструментів.

В цілому, використання сучасного генеративного штучного інтелекту в ігровій розробці відкриває нові можливості для творчості та оптимізації процесів, проте вимагає уважної адаптації та налагодження залежно від конкретного сценарію використання.

Список використаних джерел:

1. Nielsen J. AI Improves Employee Productivity by 66%. *Nielsen Norman Group*. URL: https://www.nngroup.com/articles/ai-tools-productivity-gains/ (date of access: 21.11.2023).
2. The Evolution of AI in Gaming - Big Cloud. *Big Cloud*. URL: https://bigcloud.global/the-evolution-of-ai-in-gaming/ (date of access: 21.11.2023).
3. Srivastava S. How AI in Gaming is Redefining the Future of the Industry. *Appinventiv*. URL: https://appinventiv.com/blog/ai-in-gaming/ (date of access: 21.11.2023).

Рисунок К.4 – Текст другої сторінки публікації

Рисунок К.5 – Сертифікат учасників виставки

Ігрові технології

1. Ігровий програмний застосунок в жанрі RPG з елементами Roguelike та економічної стратегії

Автори: ***Прудіус Владислав Юрійович, Долгий Андрій Іванович,*** ст. гр. ПЗПІ-20-4, ХНУРЕ.

Науковий керівник: Новіков Юрій Сергійович, старший викладач. каф. ПІ, ХНУРЕ.

Розроблено ігровий програмний застосунок що комбінує в собі елементи популярних жанрів. В грі присутні елементи розвитку поселення, управління ресурсами, розвитку персонажа, участі у економічних та соціальних процесах ігрового світу, а також можливості дослідження процедурно генерованих підземель та динамічна бойова система.

Основною ціллю розробки є заохочення гравця до повторного проходження з мінімізацією повторного отримання ідентичного ігрового досвіду.

Для цього впроваджено низку ігрових механік та підходів які допомагають змінювати деякі елементи ігрового процесу за бажанням гравця або автоматично при повторному проходженні. Таким чином гравцеві для того щоб повноцінно пройти гру, випробувавши увесь арсенал можливих функцій, не вистачатиме одного проходження, а повторні проходження стимулюватимуть зміну ігрового стилю.

2. Ігровий додаток «Шахи»

Автор: ***Бєлєвцова Олена Сергіївна***, ст. гр. КІУКІ-22-8, ХНУРЕ.

Науковий керівник: Малькова Ірина Анатоліївна, ас. каф. ІУС, ХНУРЕ.

Інтерактивна гра "Шахи" надає гравцям можливість взаємодіяти з шаховою дошкою, здійснювати ходи та виконувати різноманітні дії, такі як рокіровка, взяття на проходженні, атаки та захисти фігур.

Основною функціональністю гри є перевірка правильності здійснених гравцем ходів та визначення матових ситуацій. Гравці можуть піддаватися мату або завдавати його опоненту, використовуючи різні шахові стратегії та тактики.

Гра розроблена з використанням мови програмування C++ для написання логіки гри та графічної бібліотеки для створення візуального інтерфейсу.

3. Ігровий додаток «ColorMix»

Автори: ***Двінов Ілля Юрійович, Кіров Микита Русланович***, ст. гр. КНТ-21-3, ХНУРЕ.

Науковий керівник: Малькова Ірина Анатоліївна, ас. каф. ІУС, ХНУРЕ.

Ігровий додаток "ColorMix" призначений для розваги та цікавого проведення вільного часу. Гру виконано у жанрі аркада-головоломка, що характеризується коротким часом, але інтенсивним ігровим процесом. Ігровий додаток допоможе гравцеві зняти стрес, а також розвинути свої інтелектуальні здібності та увагу.

Гру розроблено з використанням ігрового рушія Unity, для написання скриптів використано мову програмування C#. Ігрову графіку створено за допомогою програми Blender.

Рисунок К.6 – Сторінка з матеріалами в каталозі виставки