Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

РОЗРОБКА 2D-ГРИ «DAYLIGHTER'S CHAMBER» В ЖАНРІ HORROR-LIKE З РІЗНИМИ ПОЛЯМИ ЗОРУ З ВИКОРИСТАННЯМ UNITY

DEVELOPMENT OF THE 2D GAME «DAYLIGHTER'S CHAMBER» IN THE HORROR-LIKE GENRE WITH VARYING FIELDS OF VIEW USING UNITY

спеціальність 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення»

Виконав: здобувач вищої освіти
групи ІПЗс-21
Ковальов Владислав Юрійович

Керівник:
Ящук Андрій Анатолійович

Кваліфікаційну роботу
допущено до захисту
«10» 06 2025 р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Наталія Миколаївна

_____

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти *бакалавр*
Галузь знань: *12 «Інформаційні технології»*
Спеціальність: *121 «Інженерія програмного забезпечення»*
Освітня програма: «*Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
«20» 12 2024 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Ковальову Владиславу Юрійовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи: *«Розробка 2D-гри «DayLighter's Chamber» в жанрі horror-like з різними полями зору з використанням Unity»*
Керівник роботи: *к.т.н., доцент Ящук А. А.*
затверджені наказом закладу вищої освіти від *«19» грудня 2024 р.* № *474/01-02*
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи бакалавра *«10» червня 2025 р.*
3. Вихідні дані до роботи: *ігровий рушій Unity, редактор програмного коду Visual Studio Code, приклади запозичених елементів та ідей з ігрових проектів, методичні вказівки до виконання кваліфікаційної роботи бакалавра*
4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити): *аналіз адаптивних можливостей існуючих аналогів ігор у жанрі horror-like, структуризація та вибір інструментів розробки, розробка архітектури гри, включаючи логіку процедурної генерації рівнів, навичок та поля зору, зміни перспективи (поля зору) у межах взаємодії з об'єктами на рівні, реалізація подачі сюжету у вигляді комікснх панелей, розробка базового інтелекту для ворогів, тестування гри, створення власного дизайну для гри, створення інсталяційного пакету для проекту, аналіз перспектив її просування*
5. Перелік графічного матеріалу: *29 рисунків, 2 таблиці, 3 додатки*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	*Ящук А. А.*		
Специфікація вимог до розробленої системи	*Ящук А. А.*		
Розробка об'єкта проектування	*Ящук А. А.*		
Нормоконтроль	*Вознюк А. В.*		
Гарант ОП	*Ліщина Н. М.*		
Показник запозичень тексту	1,15 %		
Академічна доброчесність	*Ящук А. А.*		

7. Дата видачі завдання *«20» грудня 2024 р.*

№	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	Огляд літературних джерел по темі кваліфікаційної роботи бакалавра	до 04.02.2025 р.	вик.
2	Аналіз проблеми розробки та впровадження об'єкту проектування	до 01.03.2025 р.	вик.
3	Обгрунтування вибору шляхів, технологій і засобів вирішення поставленого завдання	до 15.03.2025 р.	вик.
4	Розробка функціонально-структурної схеми роботи об'єкта проектування	до 29.03.2025 р.	вик.
5	Практична реалізація об'єкта проектування	до 26.04.2025 р.	вик.
6	Тестування та налагодження об'єкта проектування	до 03.05.2025 р.	вик.
7	Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру	до 10.06.2025 р.	вик.

Здобувач вищої освіти _____ Владислав КОВАЛЬОВ

Керівник кваліфікаційної роботи _____ Андрій ЯЩУК

АНОТАЦІЯ

Ковальов В. Ю. Розробка 2D-гри «DayLighter’s Chamber» в жанрі horror-like з різними полями зору з використанням Unity. Рукопис. Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення» спеціальності «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків.

У першому розділі здійснено аналіз сучасного стану проблематики проекту і сформульовано завдання. В другому розділі описано його структуру та ідеї, використані при розробці. У третьому розділі здійснено опис способів та методів розробки основних етапів проекту кваліфікаційної роботи. У висновках підведено підсумок про створений програмний продукт та зроблено оцінку результату та можливості подальшого розвитку проекту.

Ключові слова: ігровий рушій, Unity, мова програмування, C#, Visual Studio Code, ігровий проект, навички, зміна поля зору, ігровий жанр, horror-like, rogue-like, персонаж, ворог, префаб, спрайт, тайлмап.

ABSTRACT

Kovalov V. Y. Development of the 2D Game "DayLighter's Chamber" in the Horror-like Genre with Varying Fields of View Using Unity. Manuscript. Bachelor's qualification work of the OP "Software Engineering" specialty "Software Engineering". Lutsk National Technical University. Lutsk, 2025.

The bachelor's qualification work consists of an introduction, three sections, conclusions, a list of sources used and appendices.

The first section analyzes the current state of the project's problems and formulates the task. The second section describes its structure and ideas used in the development. The third section describes the methods and methods for developing the main stages of the qualification work project. The conclusions summarize the created software product and assess the result and the possibilities for further development of the project.

Keywords: game engine, Unity, programming language, C#, Visual Studio Code, game project, skills, changing the field of view, game genre, horror-like, rogue-like, character, enemy, prefab, sprite, tilemap.

ЗМІСТ

ВСТУП

Актуальність теми. Світ розробки комп’ютерних ігор, як один із найдинамічніших напрямів програмної інженерії, постійно змінюється та розвивається на Європейському ринку. Ще з моменту зародження перших 2D-ігор, що використовували найпростіші механіки, ця галузь за кілька десятиліть перетворилась на багатомільярдну індустрію, що поєднує програмування, дизайн, психологію, наратив і творчість. Сучасні гравці очікують не лише якісної графіки і апаратного навантаження, а й глибокого геймплею, атмосферності, сюжетного наповнення та нестандартних підходів до взаємодії з ігровим середовищем. Кожен гравець чекає чогось нового у власному досвіді. Це потрібно не тільки задля отримання досвіду, як розробника, але й для того, щоб гравець, як звичайна людина, затрималась саме у твоїй грі як-найдовше.

На сьогодні особливої популярності набули інді-проекти, які дозволяють розробникам реалізовувати унікальні ідеї без обмежень великих студій. Однією з найпопулярніших платформ для створення таких ігор є ігровий рушій Unity, нині версії 6, що надає потужні інструменти для розробки 2D- та 3D-ігор із широким спектром можливостей: від фізики та анімацій – до мережевої взаємодії та підтримки мобільних платформ.

У межах цієї кваліфікаційної роботи було поставлено задачу створити 2D-гру в жанрі horror-like під назвою «DayLighter’s Chamber», у якій реалізовано ряд унікальних геймплейних рішень – випадкову генерацію рівнів, обмежені поля зору заради взаємодії з функцією зміни перспективи під час взаємодії з об’єктами та ворогами, а також нестандартну подачу міні-сюжету через комікс-панелі, стилізовані під мангу. Усі ці функції реалізовано засобами рушія Unity 6.

Метою кваліфікаційної роботи є проектування та розробка програмного продукту – 2D-гри, що поєднує технічні та художні рішення для створення глибокого досвіду гравця, який він ніколи не забуде.

Об’єктом кваліфікаційної роботи є технології розробки комп’ютерних ігор із процедурною генерацією, обмеженим полем зору та змінною перспективою задля того, щоб розширити ігровий досвід гравця.

Предметом кваліфікаційної роботи виступає процес створення 2D-гри на рушії Unity, включаючи архітектуру гри, особливості генерації рівнів, реалізацію механік огляду та наративного дизайну через візуальні засоби.

Для реалізації поставленої мети необхідно вирішити наступні завдання:

– виконати огляд та аналіз існуючих розробок 2d-horror-ігор на Unity, існуючих програмних рішень для реалізації коду поставлених задач на проект на мові C#;

– спроектувати майбутню структуру гри та подати її у вигляді схем;

– розробити архітектуру гри, включаючи логіку процедурної генерації рівнів та системи циклу навичок;

– реалізувати зміну перспективи (зміна поля зору) у межах взаємодії з об’єктами на рівні;

– створити механіку подачі сюжету у вигляді комікснних панелей, де це потрібно;

– реалізувати поведінку керованого компаньйона, що буде реагувати на поведінку ворогів, з урахуванням дистанції до них;

– здійснити тестування гри та виправити знайдені помилки;

– реалізувати дизайн (стилістику гри) у подальшій розробці за допомогою отриманого досвіду під час навчання;

– розробити інсталяційний пакет та документацію до гри;

– модернізувати код задля перспективи її просування і розширення при подальшому доопрацюванні головної ідеї.

РОЗДІЛ 1
АНАЛІЗ РОЗРОБКИ 2D-HORROR-ІГОР НА UNITY І ПОСТАНОВКА ЗАВДАНЬ НА КВАЛІФІКАЦІЙНУ РОБОТУ

1.1 Аналіз сучасного стану проблеми

Ігрова індустрія – одна з найперспективніших і найдинамічніших галузей сучасного програмного забезпечення. Щороку на ринок виходять тисячі нових ігор, які реалізуються як великими студіями, так і незалежними розробниками (інді). Серед великого розмаїття жанрів особливо виділяється horror – жанр, мета якого викликати у гравця напругу, страх або тривогу за рахунок аудіовізуальних та сценарних прийомів.

Horror-ігри мають давню історію, яка бере початок із таких проектів, як «Alone in the Dark» (1992) та «Resident Evil» (1996). Згодом вони трансформувались у цілі піджанри – survival horror, psychological horror, action horror, horror roguelike тощо. Однією з сучасних тенденцій є реалізація horror-елементів у 2D-середовищі з використанням процедурної генерації рівнів, обмеженого поля зору та подання сюжету через нестандартні візуальні прийоми.

Існує низка вже реалізованих проектів, які частково використовують ці механіки:

– «Darkwood» – 2D survival horror з видом зверху, де поле зору обмежене освітленням, а гравець має вижити в процедурно згенерованому лісі. Використовується атмосфера тривоги й аудіоефекти для посилення страху [3];

– «Lone Survivor» – піксельна 2D-horror пригода з психологічним ухилом, де гравець бачить світ від третьої особи, але з елементами «темного» сюжету та нестабільної психіки головного героя [7];

– «Don’t Starve» – гра, що поєднує виживання і процедурну генерацію, хоч і не є класичним horror, але використовує атмосферу тривоги та неспокою, особливо вночі [5];

– «Mad Father» [8], «The Witch’s House» [11] – японські RPG Maker horror-ігри з топ-даун видом і сюжетною подачею через відео та діалоги;

– «Otherworld Legends» – як приклад ідеальної коміксної подачі діалогів. Саме ним ми будемо надихатися при розробці власних комікснних панелей (рис. 1.1) [9].

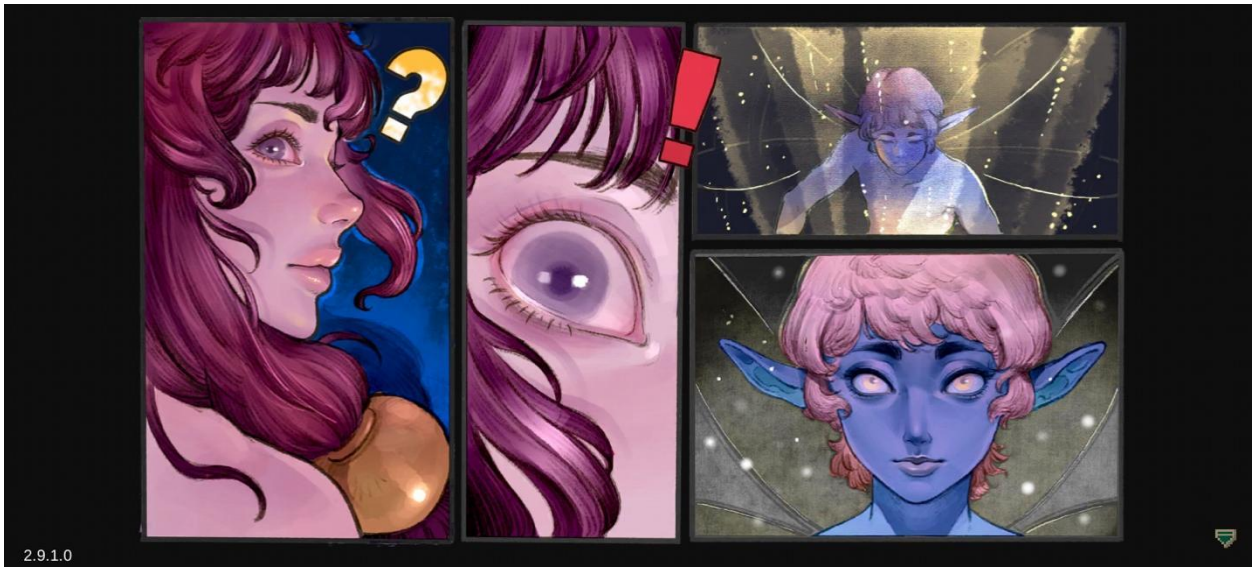

Рисунок 1.1 – Подача панельних сюжетних діалогів
у грі «Otherworld Legends» [9]

«Endless War – Roguelike RPG» – ще одна Android/IOS-гра, яка надає гравцю досвід взаємодії з процедурно згенерованими рівнями у рамках 2D-стилістики та випадковими персонажами. Саме цим проектом ми будемо надихатися при розробці процедурної зовнішності (рис. 1.2) [6].

Слід засвідчити, що процедурна генерація є фракталізованою – вона працює на будь-якому масштабі у таких проектах (навіть якщо формули можуть різнитися). Тобто, масштабний рівень – генерація рівнів (кімнат), середній рівень – генерація випадкових ходів в ці кімнати, зовнішності персонажа, його вмінь та інших деталей, та мінімальний рівень – генерація об’єктів та ворогів у кімнаті за заздалегідь визначеною формулою.

Рисунок 1.2 – Процедурно згенерований рівень у Android-грі «Endless War – Roguelike RPG» [6]

Певні елементи та вороги у приведених в прикладі іграх розміщені у випадковому порядку за певною формулою, що вказана у коді проекту. Такий код представляє собою роботу з встановленням змінних, що впливають на вигляд проекту (як-от встановлення спрайтів чи тайлмапів – тайлових карт – у проекті). Також реалізовано випадковість у розміщенні системи кімнат та проходів між ними. Це базові елементи rogue-like-ігор у проекті.

До базових елементів таких ігор відноситься і система навичок, яка вже реалізована у багатьох суміжних проектах, проте по-своєму (рис. 1.3).

Рисунок 1.3 – Система навичок у Android-проекті «Endless War - Roguelike RPG» [6]

Навички у даній грі подані як система пов’язаних один з одним аспектів – одна навичка викликає тригер, що необхідний задля роботи наступної.

Саме схожу ідею на описану вище взято до уваги при розробці кваліфікаційної роботи.

Можна зазначити, що кожен з прикладів має свої переваги, що включені у дану власну розробку: наявність процедурних формул, що використовуються для генерації та реалізації випадкових елементів у грі, унікальна атмосфера, подача сюжету через середовище (так звана мовчазна подача історії) діалоги, інтерфейс, що адаптований під жанр та особливо під сюжет (в основному, з мінімальною участю гравця, проте зрозумілий та доступний).

Однак у них є й відмінності як між собою, так і між нашим майбутнім проектом, які по своїй суті не є недоліками, але у питанні унікальності нашої розробки дають нам перевагу над ними:

– багато з них використовують лише один ігровий шар перспективи, без її зміни навіть під час сюжетних моментів;

– відсутня глибока взаємодія з атмосферою через перспективу (гравець бачить весь рівень за допомогою карти чи інших засобів контролю переміщення на рівні);

– сценарії реалізовані або як окремі діалоги, або як повноекранні заставки без елементів коміксної подачі.

Зокрема, окрім сценарію, необхідно мати чіткий план для того, щоб створити динаміку між ігровим досвідом та сюжетними моментами. Задля цього поставлено за ціль використовувати лише сірий спектр кольорів. Це задає певну динаміку будь-якому моменту гри.

Отже, на основі представлених прикладів зроблено висновок про те, що даний проект буде виділятися серед інших такими рисами:

– обмежене поле зору без доступу до карти задля нечіткого розуміння структури рівня (гравець не бачить всіх кімнат, що дозволяє збільшити вплив атмосфери гри на гравця);

– реалізація ідеї взаємодії з навичками в циклічному режимі (навички виконуються, при виклику, один за одним, а не всі в будь-якому незаданому порядку по одному);

– процедурна генерація рівнів, що робить кожне проходження унікальним через нестандартність розміщення кімнат, що також надає можливість розширювати кількість рівнів до будь-якого обсягу;

– унікальна сюжетна ідея, реалізована через комікcну подачу на окремій сцені (з перспективою розширення сюжету).

Вибір рушія для розробки даного ігрового проекту, як частини кваліфікаційної роботи, є настільки ж важливим, наскільки є важливим і вибір мови програмування.

Було прийнято рішення зупинитись на рушієві створення ігор Unity та мові програмування C#, яка є базовою мовою програмування для нього.

Ігровий рушій Unity є одним із найпопулярніших серед розробників 2D- та 3D-ігор, особливо в інді-середовищі. Його переваги та недоліки застосування представлені у таблицю 1.1. Дата таблиця обґрунтовує чому саме цей рушій був обраний для виконання поставлених завдань на розробку проекту [13].

Unity є найбільш зручним і доцільним серед усіх інструментів, особливо при розробці додатку для платформи Windows, задля якої реалізовано проект.

Таблиця 1.1 – Порівняння переваг та недоліків ігрового рушія Unity 6

Переваги вибору рушія	Недоліки вибору рушія
Підтримка як 2D, так і 3D розробки	Високе споживання ресурсів при масштабних проектах
Зручний інтерфейс та інтуїтивна система компонування об'єктів	Може виникати плутанина у великій кількості об’єктів на сцені
Широкі можливості C# для програмування логіки гри	Потрібна гарна оптимізація при роботі з фізикою чи великою кількістю елементів
Величезна бібліотека готових рішень та активів (Asset Store) [12]	Деякі пакети у Asset Store платні або застарілі (тому ми будемо малювати їх власноруч)
Активна спільнота, велика кількість навчальних матеріалів	Часті оновлення можуть ламати сумісність між версіями

Продовження таблиці 1.1

Кросплатформеність (можливість експорту під Windows, Android, WebGL тощо)	WebGL-версія має обмеження у функціональності та важкий для оптимізації
Безкоштовна версія підтримує повноцінну розробку	Для комерційних проектів може знадобитись платна ліцензія

Вибір рушія Unity є обґрунтованим та затвердженим з керівником кваліфікаційної роботи, особливо з урахуванням досвіду роботи з даним рушієм, отриманого на переддипломній практиці.

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

Для досягнення поставленої мети розробки розробка гри «DayLighter’s Chamber» у жанрі horror-like з використанням рушія Unity 6, яка реалізовує унікальні механіки зміни поля зору, подачі сюжету через панелі в стилі коміксів та процедурної генерації рівнів з циклічним викликом навичок, потрібно виконати наступні завдання:

– спроектувати майбутню структуру гри та подати її у вигляді схем;

– розробити архітектуру гри, включаючи логіку процедурної генерації рівнів та системи циклу навичок;

– реалізувати зміну перспективи (зміна поля зору) у межах взаємодії з об’єктами на рівні задля конкретної цілі (у нашому випадку, на одному рівні – зверху, на іншому – збоку);

– створити механіку подачі сюжету у вигляді комікснх панелей, де це потрібно, які з’являються у визначені моменти ігрового часу, або при взаємодії з об’єктами на рівні;

– реалізувати поведінку керованого компаньйона (з іменем Арекс), що буде реагувати на поведінку ворогів, з урахуванням дистанції до них за допомогою спеціальних функцій;

– здійснити тестування гри та виправити знайдені помилки;

– розробити дизайн (стилістику гри) у подальшій розробці за допомогою отриманого досвіду під час навчання задля уникання зіткнення з авторським правом у разі використання запозичених текстур об’єктів від інших авторів (наприклад, певні запозичені спрайти);

– розробити інсталяційний пакет та документацію до гри;

– модернізувати код задля перспективи розширення гри при подальшому доопрацюванні головної ідеї, а також можливого залучення спів розробників до проекту.

Стилістика гри повинна відповідати зазначеному жанру у темі проекту.

РОЗДІЛ 2
СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проектування програмного забезпечення

Проект у Unity можна поділити на сцени. Таким чином, задля оптимізації взаємодії з кодом, отримаємо три:

– сцена «Головне меню гри» (MenuScene.unity у структурі файлів проекту);

– сцена «Комікcний сюжет» (LoreMetaScene.unity у структурі);

– сцена «Рівень підземелля» (StartDungeonScene.unity у структурі проекту).

Схема структури взаємодії сцен даного проекту між собою у цій системі буде реалізована через об’єкти кнопок, через клавіші на клавіатурі та об’єкти на сцені.

Вона відображає сутність взаємодії користувача з інтерфейсом даного застосунку та функції, що прив’язані до елементів інтерфейсу.

Елементи інтерфейсу розташовані за певним зразком, який передбачений чималою кількістю прикладів у Інтернетному просторі.

Зокрема, проект «Endless War – Roguelike RPG» використовує саме такий вид подачі візуальних елементів, що було запозичено та, через різність платформ розробки, адаптовано до обрані платформи проекту [6].

Чітка структуризація сцен у межах одного проекту спрощує підтримку та масштабування гри в майбутньому – зокрема, додавання нових рівнів, сюжетних вставок або інтерактивних елементів.

Така структура інтерфейсу досить оптимізована і зі сторони вона виглядає достатньо просто та схематично для будь-якого користувача, що буде тестувати проект у подальшому, або ж пізнавати його вперше (рис. 2.1).

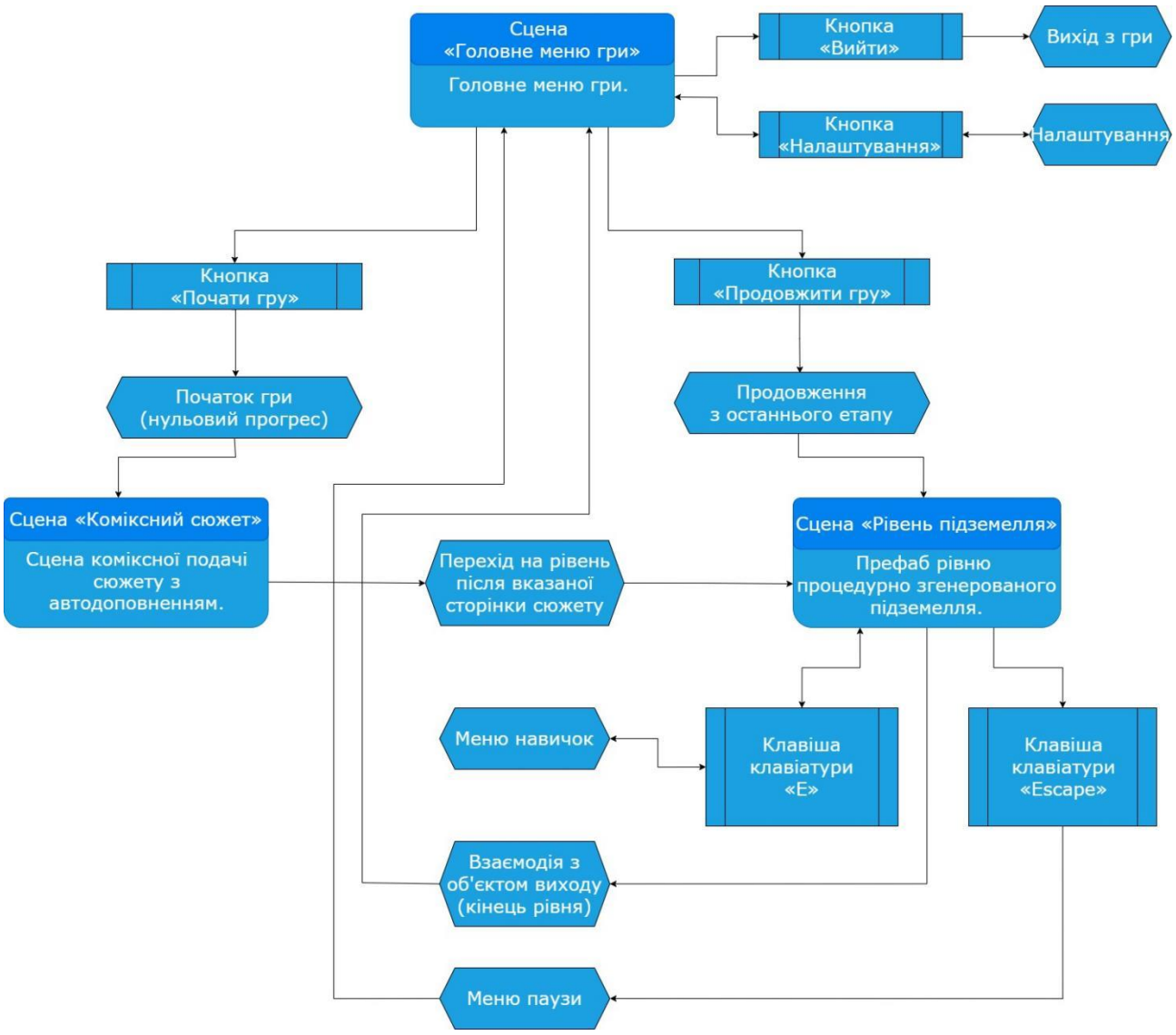

Рисунок 2.1 – Схема взаємодії сцен між собою у проекті та короткий опис їх призначення

Розглянемо архітектуру самих сцен у проекті. На рисунку 2.2 зображено порядок переходу зі сцени на сцену та їх наповнення – об’єкти, елементи інтерфейсу, ідейне призначення.

Саме тут можна виділити та ідейно представити важкість реалізації кожної сцени, а також побудувати їх структуру для реалізації.

Це дає змогу не лише візуалізувати логіку переходів між сценами, а й оцінити складність кожної з них з точки зору функціонального навантаження та взаємодії з користувачем.

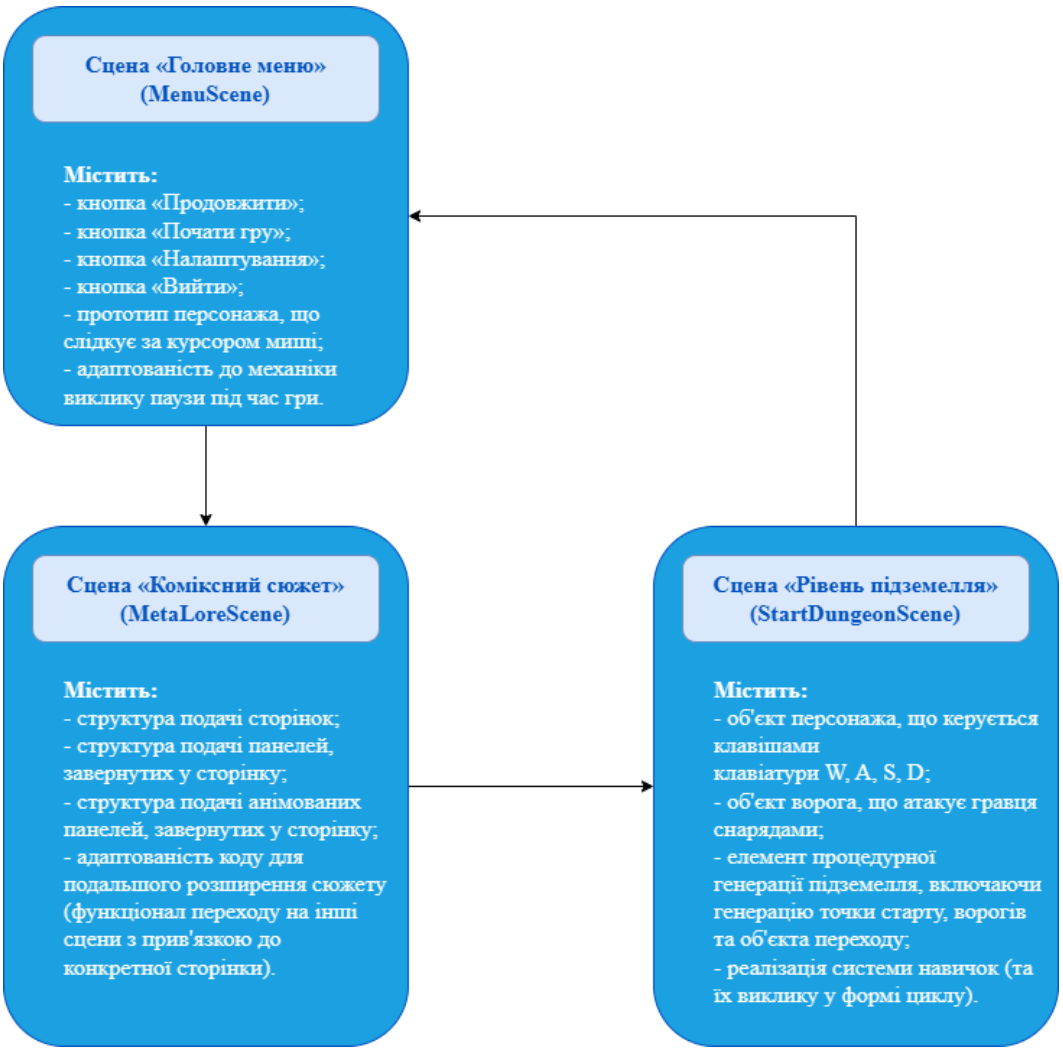

Рисунок 2.2 – Схема архітектури сцен у проекті, а також порядок взаємодії з ними

Перспектива розвитку кваліфікаційної роботи пов’язана з долученням інших розробників до роботи над цим проектом у подальшому часі, що на етапі планування структури вимагає оптимізувати код проекту.

Це означає, що потрібно заздалегідь розпланувати кількість сцен, скриптів (відмежованих програмних рішень, до яких відносяться класи та посилання у мові програмування C#) [1].

При умові доручення сторонніх розробників, вони вже будуть орієнтуватись на лістинги коду, на методичні вказівки до роботи, а також на поради головного розробника.

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

Для реалізації задачі процедурної генерації приміщень та розміщення об’єктів у середовищі гри обрано сучасні інструменти розробки, що забезпечують гнучкість, масштабованість та візуальну наочність.

Середовище розробки – Visual Studio Code, яке забезпечує комфортне написання коду завдяки підсвічуванню синтаксису, автодоповненню та інтеграції з системою контролю версій, особливо під ігровий рушій Unity 6, що дозволяє реалізовувати складну логіку генерації рівнів, використовуючи компонентно-орієнтований підхід, систему префабів та вбудовану підтримку 2D/Tilemap для генерації підлоги [4].

Для вирішення задачі була реалізована система процедурної генерації за допомогою скрипту DungeonGenerator.cs, повний код якого продемонстровано у додатку А. Основні алгоритмічні підходи та методи було сформовано з перспективою розвитку проекту та оптимізації роботи над ним.

Використано метод рекомбінування приміщень на основі шаблонів: використовується колекція RoomPrefab, кожен елемент якої містить префаб кімнати та опис можливих напрямків виходів (напр., «U», «D», «L», «R»).

На основі сумісності з сусідніми кімнатами у наведеному методі виконується добір допустимих конфігурацій.

Приклад реалізації у мові C# такого методу, який долучено до коду процедурної генерації, наведеній у лістингу 2.1.

При заповненні кожної клітинки сітки рівня виконується перевірка на допустимість конфігурації кімнати, враховуючи наявні сусідні приміщення. Це дозволяє уникнути конфліктів між приміщеннями.

Для візуального оформлення меж рівня по периметру також додатково генеруються кімнати без проходів.

Лістинг 2.1 – Метод рекомбінування приміщень на мові програмування C#

```
List<RoomPrefab> GetValidRoomsForPosition(int x, int y)
{
    List<RoomPrefab> validRooms = new List<RoomPrefab>();

    foreach (var room in roomPrefabs)
    {
        if ((y == height - 1 && room.exits.Contains('U')) ||
            (y == 0 && room.exits.Contains('D')) ||
            (x == 0 && room.exits.Contains('L')) ||
            (x == width - 1 && room.exits.Contains('R')))
        {
            continue;
        }

        if (IsCompatible(room, x, y))
            validRooms.Add(room);
    }

    return validRooms;
}
```

кінець лістингу 2.1

Забезпечення зв’язності рівня реалізовано за допомогою алгоритму пошуку в ширину (BFS), який визначає, які приміщення є досяжними зі стартової точки. Всі недосяжні кімнати замінюються на пусті.

На основі сформованої структури рівня далі виконуються виклики методів створення персонажу у випадковій точці, ворогів (у кожній кімнаті), та виходів у випадковій, що не містить ключа виходу «U».

По ідеї на самій сцені в результаті не будемо мати майже ніяких елементів окрім інтерфейсу, оскільки вся генерація реалізована префабами об’єктів і сам рівень видно лише коли розпочато тестування проекту.

Такий підхід дозволяє динамічно створювати ігрові рівні зі збереженням логіки, зв'язності та випадковості. Це забезпечує реіграбельність гри, кожен запуск якої може генерувати унікальний підземелля.

РОЗДІЛ 3
РОЗРОБКА 2D-ГРИ «DAYLIGHTER'S CHAMBER»

3.1 Практична реалізація об’єкта проектування

При старті проекту, перша сцена для гравця – це меню. В ньому він може почати гру, налаштувати звук для подальшої гри, або вийти. Також, ця ж сама сцена використовується як меню паузи під час гри. Оскільки отримано досвід роботи з дизайном спрайтів, тайлмапів та інших елементів гри, було вирішено розробити власний стиль (рис. 3.1).

Отже, усі спрайти для проекту були намальовані власноруч, та встановлені, як об’єкти сцени.

Рисунок 3.1 – Зовнішній вигляд меню після розробки власного дизайну

Щоб надати зовнішньому вигляду меню більшої динаміки, було розроблено скрипт, що надає ефект плавного входу на сцену – фейдінг. А також ефект плавного в’їзду на сцену для кнопок меню.

Код цього скрипту представлено у лістингу 3.1.

Лістинг 3.1 – Код скрипту, що відповідає за динамічне відображення меню

```
[RequireComponent(typeof(RectTransform), typeof(CanvasGroup))]
public class LogotypicTransform : MonoBehaviour
{
    public float delay = 1f;
    public float slideDuration = 0.5f;
    public float offsetX = -150f;
    private RectTransform rectTransform;
    private CanvasGroup canvasGroup;
    private Vector2 startPos;
    private Vector2 targetPos;
    private float timer;
    private bool sliding = false;
    void Start()
    {
        rectTransform = GetComponent<RectTransform>();
        canvasGroup = GetComponent<CanvasGroup>();

        targetPos = rectTransform.anchoredPosition;
        startPos = new Vector2(targetPos.x + offsetX, targetPos.y);
        rectTransform.anchoredPosition = startPos;
        canvasGroup.alpha = 0f

        Invoke(nameof(StartSliding), delay);
    }
    void StartSliding()
    {
        timer = 0f;
        sliding = true;
    }
    void Update()
    {
        if (!sliding) return;

        timer += Time.deltaTime;
        float t = Mathf.Clamp01(timer / slideDuration);
        float smoothT = Mathf.SmoothStep(0, 1, t);

        rectTransform.anchoredPosition      =      Vector2.Lerp(startPos,
targetPos, smoothT);
        canvasGroup.alpha = smoothT;

        if (t >= 1f) sliding = false;
    }
}}
```

кінець лістингу 3.1

Оскільки більшість об’єктів на сцені меню – це зображення (Image), кодово можна задати їм прозорість, саме яка й змінюється за таймером у функції Update(). По дефолту дана функція у рамках рушія Unity та мови C# використовується як та, що реєструє задані оновленя та миттєво їх застосовує.

У функції Start() натомість реєструються лише початкові значення для об’єктів, які не задані у Інспекторі Unity як змінні.

Unity має власний Інспектор, куди задля збільшення ефективності під час тестування винесено деякі public-змінні. Це досить зручна функція, яка у подальшому дозволить збільшити швидкість та продуктивність реалізації структури проекту (рис. 3.2).

Саме у Інспектор занесені елементи, які викликаються у коді для взаємодії. Натомість, робота з ними проходить ще до їх встановлення у Інспекторі, оскільки Unity легко розрізняє типи об’єктів чи змінних, з якими він працює.

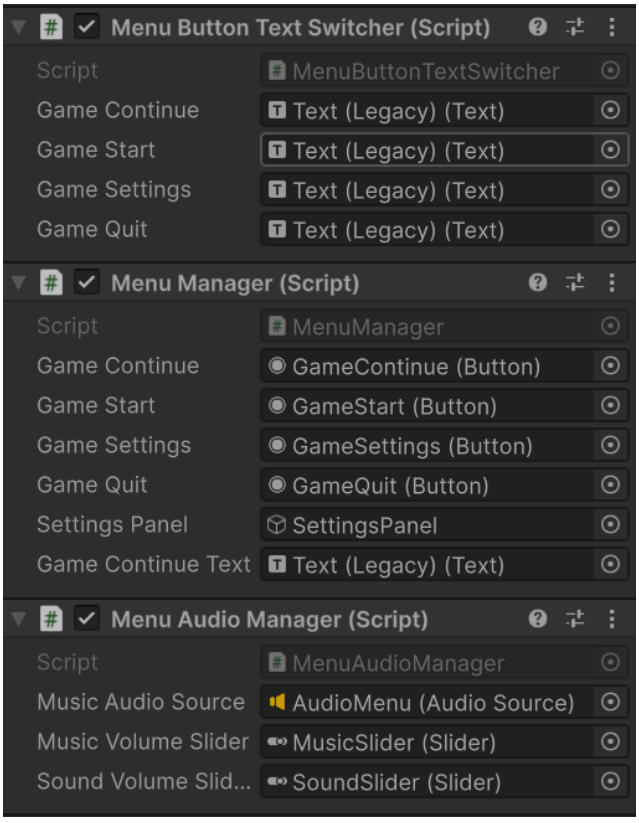

Рисунок 3.2 – Реалізація під’єднання посилань на об’єкти, що викликаються у коді, через Інспектор у Unity за допомогою навішування скриптів на об’єкт

На рисунку видно, що на ньому є ще кілька скриптів. При цьому кожен з них відповідає за свою частину роботи. Усі вони навішані на єдиний об’єкт MenuController (рис. 3.3).

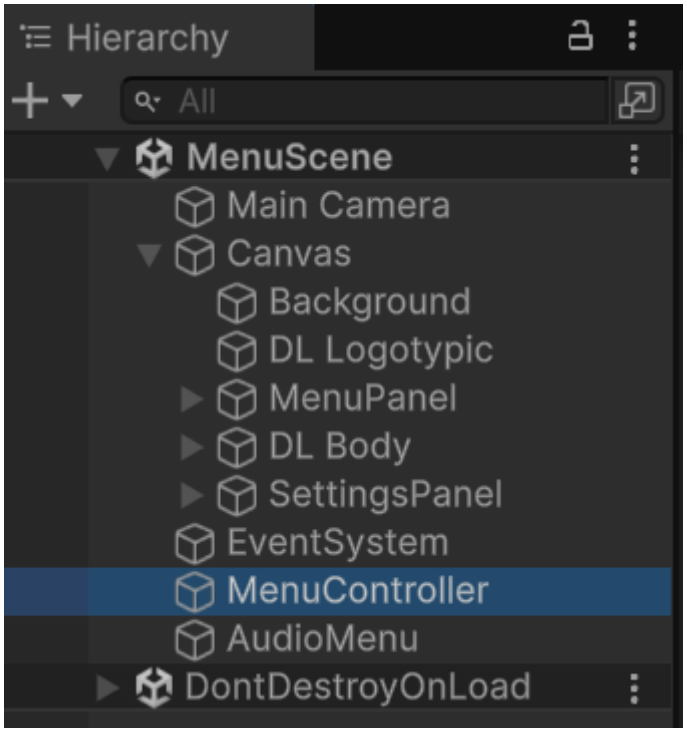

Рисунок 3.3 – Пустий об’єкт MenuController у ієрархії сцени меню задля навішування основних скриптів на нього

Розглянемо тепер ще один скрипт, що надає унікальності цьому меню. А саме, анімування прототипу персонажу, який знаходиться в центрі кадру. Це надає йому живості, а також у комбінації з фоном демонструє ідею ізольованості мислення персонажа. На рисунку 3.4 зображено як саме виглядає анімація.

Вона складається з восьми фреймів для очей, проте продемонстровано буде лише три з них, для прикладу. Всі вони також реалізовані власноруч, з урахуванням анатомічних особливостей персонажу, а також з використанням кольорів одного стилю: білого, сірого та чорного.

Такий мінімалістичний підхід до кольорової гами підкреслює емоційний стан героя та відповідає загальному стилю гри.

Рисунок 3.4 – Три фрейми з восьми, що демонструють анімацію обличчя у прототипу персонажа з меню

Звісно ж, у Unity анімування доступне через дві вкладки: Animation та Animator. Проте завдавши базову анімацію спрайтів через перше вікно, зручніше кодово керувати таймінгами та тригерами для неї.

Саме це й виконує код скрипту з лістингу 3.2.

Лістинг 3.2 – Код скрипту, що відповідає за управління анімацією обличчя

```
    void Start() {
        rectTransform = GetComponent<RectTransform>();
        canvasGroup = GetComponent<CanvasGroup>();
        targetPos = rectTransform.anchoredPosition;
        startPos = new Vector2(targetPos.x + offsetX, targetPos.y);
        rectTransform.anchoredPosition = startPos;
        canvasGroup.alpha = 0f;
        Invoke(nameof(StartSliding), delay); }
    void StartSliding() {
        timer = 0f;
        sliding = true; }
    void Update() {
        if (!sliding) return;
        timer += Time.deltaTime;
        float t = Mathf.Clamp01(timer / slideDuration);
        float smoothT = Mathf.SmoothStep(0, 1, t);
        rectTransform.anchoredPosition = Vector2.Lerp(startPos,
targetPos, smoothT);
        canvasGroup.alpha = smoothT;
        if (t >= 1f) sliding = false; }
```

кінець лістингу 3.2

Даний скрипт-код досить гнучкий, хоч і короткий. Його можна було долучити і до коду з реалізацією фейдінгу, проте рішення їх відокремити демонструє бажання оптимізувати очну роботу зі скриптами, що дозволить більш гнучко в них орієнтуватися і дозволить уникнути нагромадження великої кількості коду у одному файлі.

Щоб довести ефект атмосфери до ідеалу, потрібно продемонструвати навички роботи з 2D-простором та шарами. Задля цього додамо рух обличчя за курсором миші, якщо ми водимо ним поблизу прототипу персонажа.

Цього можна досягнути поділивши об’єкт прототипу на батьківські та дочірні (рис. 3.4).

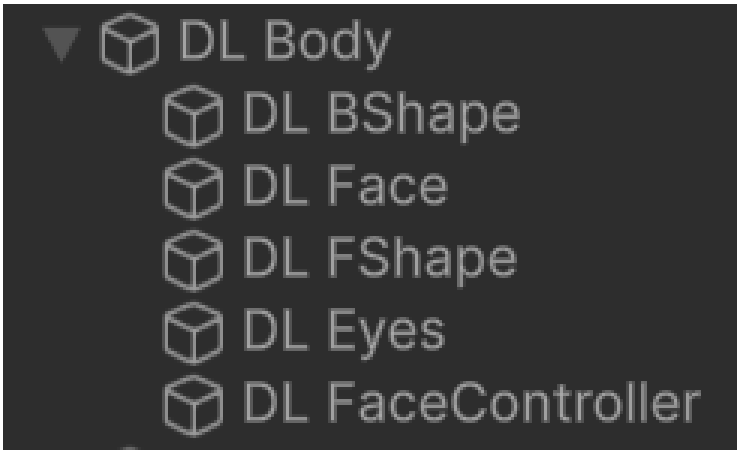

Рисунок 3.4 – Об’єкт, структурований пошарово задля реалізації плавної анімації обличчя

Хоч ідея і звучить важко, реалізація її вміщується в п’ятдесят три рядки коду.

Саме візуальне розділення елементів на шари можна представити, як головну позитивну рису цього підходу.

В чому ідея: рух кожного шару (шар очей, обличчя, заднього фону персонажу, його тіла) зумовлений різним ефектом паралаксного переміщення за курсором. Деякі – лише по координаті X, а інша більшість – за X та Y.

Описана взаємодія реалізована у коді і подана у лістингу 3.3.

Лістинг 3.3 – Код скрипту, що відповідає за рух обличчя за курсором

```
    void Start()
    {
        Canvas canvas = GetComponentInParent<Canvas>();
        if (canvas != null)
            canvasRect = canvas.GetComponent<RectTransform>();
        if (eyes != null) eyesInitialPos = eyes.anchoredPosition;
        if (face != null) faceInitialPos = face.anchoredPosition;
        if (fShape != null) fShapeInitialPos = fShape.anchoredPosition;
        lastMousePosition = Input.mousePosition;
    }
    void Update()
    {
        if (canvasRect == null) return;
        Vector2 currentMousePosition = Input.mousePosition;
        if ((currentMousePosition - lastMousePosition).sqrMagnitude > 1f)
        {
            idleTimer = 0f;
            lastMousePosition = currentMousePosition;
        }
        else
        {
            idleTimer += Time.deltaTime;
        }
        Vector2 dir = Vector2.zero;
        if (idleTimer < idleThreshold)
        {
      RectTransformUtility.ScreenPointToLocalPointInRectangle(canvasRect,
currentMousePosition, null, out Vector2 localMousePos);
            Vector2 canvasCenter = canvasRect.rect.center;
            dir = (localMousePos - canvasCenter).normalized;
        }
        Vector2 eyeTarget = eyesInitialPos + dir * eyeOffset;
        Vector2 faceTarget = faceInitialPos + dir * faceOffset;
        Vector2 fShapeTarget = fShapeInitialPos + dir * fShapeOffset;
        if (eyes != null)
            eyes.anchoredPosition =
Vector2.SmoothDamp(eyes.anchoredPosition, eyeTarget, ref eyesVelocity,
eyeSmoothTime);
        if (face != null)
            face.anchoredPosition =
Vector2.SmoothDamp(face.anchoredPosition, faceTarget, ref faceVelocity,
faceSmoothTime);
        if (fShape != null)
            fShape.anchoredPosition =
Vector2.SmoothDamp(fShape.anchoredPosition, fShapeTarget, ref
fShapeVelocity, fShapeSmoothTime);
    }
```

кінець лістингу 3.3

Перед тим, як перейти до сцен, що характеризують дану кваліфікаційну роботу, слід також зазначити функціонал та вигляд налаштувань звуку в меню.

Два елементи: саунд-слайдер та слайдер музики, їх вигляд відображений на рисунку 3.5. Вони зазначають подальші змінні, які характеризують відсоток гучності аудіо-елементів проекту, що також додані у цей проект.

Рисунок 3.5 – Вигляд кнопок та панелі налаштування звуку у меню

Також на цьому рисунку видно, що клавіша «Продовжити» неактивна. Вона буде активізована після початку гри.

Тепер можна вільно переходити до розбору наступної сцени, LoreMetaScene, що відповідає за початок гри у вигляді сюжетної кастцени.

Туди нас пересилає скрипт, прикріплений до Menu Controller у ієрархії проекту – MenuManager.cs. На лістингу 3.4 видно, що тут задається не лише сама функція переходу, але й сторінка, що кешується у дані (через те, що змінні

однієї сцени з іншої можна змінити лише шляхом запам’ятовування даних у пам’яті та зчитування кодом на місці).

Лістинг 3.4 – Елемент скрипту, що відповідає за перехід до сюжетної сцени

```
    void OnGameStartClicked()
    {
        hasStartedGame = true;
        GameContinue.interactable = true;

        PlayerPrefs.SetInt("LoreScene_CurrentPage", 1);
        PlayerPrefs.Save();

        SceneManager.LoadScene("LoreMetaScene");
    }
```

кінець лістингу 3.4

На відміну від сцени меню, тут маємо лише один скрипт – LoreMetaSceneController. Саме він зчитує зображення, що розміщені у папці MetaLoreData за адресою Assets\Resources\MetaLoreData у структурі проекту (папка, з якою код може підгрузити елементи прямо – тобто без Інспектору).

Там також відмічені візуальні елементи, через що код скрипту досить великий, проте зазначимо основні моменти у лістингу 3.5.

Лістинг 3.5 – Основні моменти реалізації задачі сюжетної сцени

```
    private void Start()
    {
        currentPage = PlayerPrefs.GetInt("LoreScene_CurrentPage", 1);
        StartCoroutine(FadeArrowPrompt(false));

        Sprite bck = Resources.Load<Sprite>($"{folderPath}/sc3_bckgr");
        if (bck) background.sprite = bck;

        StartCoroutine(PlayPanels());
    }
```

кінець лістингу 3.5

Кодова реалізація функцій створена так, що вже на старті зчитує з пам’яті та приймає змінну сторінки, з якої нам потрібно почати, а також перенаправляє нас на основний скрипт, де реалізована унікальна подача сюжету.

Структура коду для цієї сцени така:

– код зчитує зображення по їх назві;

– код виділяє сторінку;

– код виділяє фрейми у сторінці, розставляючи їх у правильному порядку;

– код виділяє анімовані фрейми сторінки, якщо такі є;

– в кінці сторінки код показує нам стрілку готовності до перемикання на іншу, і дає користувачеві власноруч це зробити за допомогою ЛКМ.

Унікальність даного підходу полягає в тому, що ми не задаємо у Інспекторі ніяких файлів, а зразу кидаємо їх в папку, просто називаючи їх спеціальним іменем. Як-от на рисунку 3.5, де відображено нинішню кількість фреймів, поділених назвою на сторінки та розставлені у порядку.

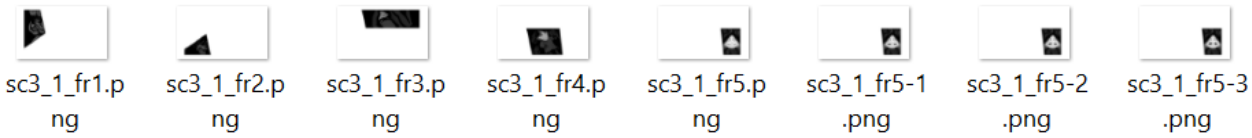

Рисунок 3.5 – Відсортовані за специфічною назвою файли для сторінок сцени, які подаються у форматі коміксу

Для прикладу візьмемо файл sc3_1_fr1.png. Його структура така, що sc3 – це заглушка (у разі використання спільної папки у подальшому), 1 – номер сторінки, fr1 – номер фрейму (таким чином задаємо їх порядок).

Анімовані фрейми – окрема частина коду, що подає картинки, які підписані на рисунку 3.5 у форматі fr5, fr5-1, fr5-2, fr5-3 як частину однієї цілої картинки за специфічно вказаним проміжком часу, що викликає у глядача ефект перегляду анімованого коміксу.

Пояснення специфікації таких фреймів (кадрів) полягає у тому, що активний фрейм збільшується, а попередній зменшується і розмивається. Тому якщо б так звані анімовані фрейми шли один за одним як і інші, такого ефекту анімації не було б.

Розглянемо рисунок 3.6, на якому активний четвертий фрейм першої сторінки. Попередні – розмиті та зменшені. Наступний – ще не відображений.

Рисунок 3.6 – Демонстрація візуальних прийомів подачі сюжету у вигляді коміксу з послідовним завантаженням кадрів

П’ятий фрейм навпаки – спочатку відображається як стандартний. Проте надалі наступні після нього завантажуються вже як єдині об’єкти. Також зазначимо, що кодово типи фреймів ніяк не прив’язані до порядку, хоч і на обох сторінках у проекті анімовані розміщені останніми. Це лише візуальний прийом коміксної подачі задля плавного переходу до динамічного сегменту – самої гри.

Як саме виглядає один єдиний анімований фрейм можна побачити на рисунку 3.7.

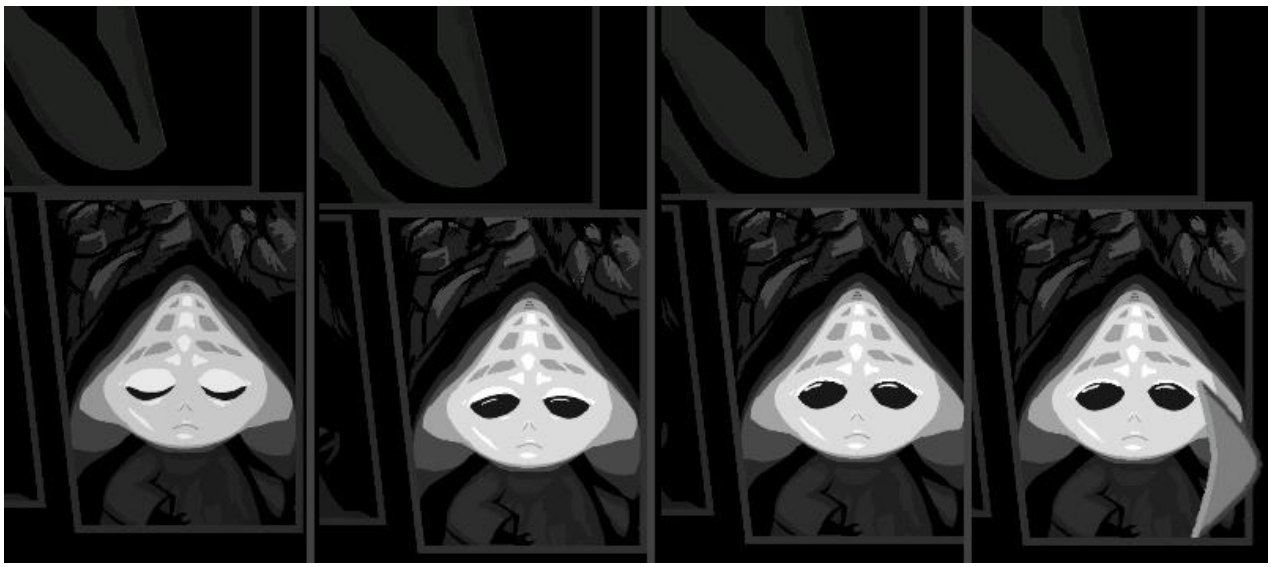

Рисунок 3.7 – Демонстрація подачі сюжету у вигляді одного анімаційного фрейму, а також відображення стрілки готовності перемикання після завантаження останнього

Таку кодову структуру реалізації прокрутки анімованих кадрів, як у лістингу 3.6, потрібно реалізувати лише раз, і потім можна буде додавати правильно названі зображення без кінця. І вони будуть відображатись як потрібно навіть якщо будуть першими у списку прокрутки.

Лістинг 3.6 – Кодова функція прокрутки анімованих кадрів

```
    List<Sprite> LoadAnimationFrames(int page, int frame)
    {
        List<Sprite> frames = new List<Sprite>();
        for (int i = 1; i <= 20; i        {
            string path = $"{folderPath}/{scenePrefix}_{page}_fr{frame}-
{i}";
            Sprite spr = Resources.Load<Sprite>(path);
            if (spr == null) break;
            frames.Add(spr);
        }
        return frames;
    }
```

кінець лістингу 3.6

Фінальна частина цієї сцени описана у тій частині скрипту, що буде автоматично перенаправляти гравця до потрібної сцени.

Раніше було згадано про адаптованість сцени до сюжетного розширення, і це правда. Фінальна частина скрипту реалізує саме цю функцію.

Розробник власноруч задає у Інспекторі цього скрипту у Unity сторінку, а також назву сцени для переходу, і мається на увазі, що як тільки переглядач пройде якусь сюжетну точку, то зразу потрапить до геймплейного сегменту проекту (рис. 3.8).

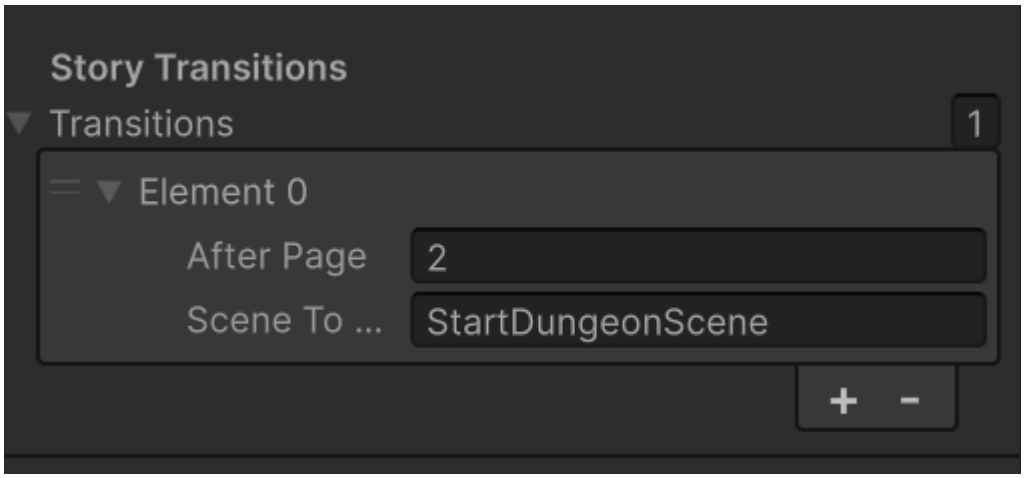

Рисунок 3.8 – Робота скрипту, подана у Інспекторі Unity, в якій реалізовано систему переходу на певну сцену (Scene To Load) після конкретної сторінки (After Page)

Це дозволяє продовжувати список і задавати не лімітовану кількість сюжетних сегментів, якщо це потрібно, після сцен геймплею, або перед ними, і не дублювати сцени, а завжди працювати на одній.

Після – можна буде вказати наступну сторінку на сцені LoreMetaScene, щоб продовжити перегляд сюжету у вигляді коміксу, при бажанні.

Будь-яка нова сцена чи її частина можуть бути легко інтегровані в існуючу структуру без потреби в окремих обробниках, а логіка переходів залишиться прозорою й керованою з єдиного центру управління.

Обґрунтована реалізація задачі увійшла у склад функції PlayPanels(), що і відображає фрейми, адже в ній було досить комфортно задати перевірку збігання змінних (ліст. 3.7).

Лістинг 3.7 – Кодова реалізація списку та переходу на сцену після сторінки

```
    [System.Serializable]
    public class LoreSceneTransition
    {
        public int afterPage;
        public string sceneToLoad;
    }

IEnumerator PlayPanels()
    {
        while (true)
        {
            string framePath =
$"{scenePrefix}_{currentPage}_fr{currentFrame}";
            Sprite mainSprite =
Resources.Load<Sprite>($"{folderPath}/{framePath}");

            if (mainSprite == null)
            {
                StartCoroutine(FadeArrowPrompt(true));
                waitingForInput = true;

                yield return new WaitUntil(() =>
                    Input.GetKeyDown(KeyCode.Return) ||
                    Input.GetKeyDown(KeyCode.Space) ||
                    Input.GetMouseButtonDown(0));

                StartCoroutine(FadeArrowPrompt(false));
                waitingForInput = false;

                ClearAllPanels();

                int previousPage = currentPage;
                currentPage++;
                currentFrame = 1;
                PlayerPrefs.SetInt("LoreScene_CurrentPage", currentPage);
                foreach (var transition in transitions) {
                if (previousPage == transition.afterPage)
                    {
                        PlayerPrefs.Save();
                        SceneManager.LoadScene(transition.sceneToLoad);
                        yield break;
                    } }
                continue;
            } } }
```

кінець лістингу 3.7

Поки що потрібен лише перехід на сцену StartDungeonScene, де і буде проходити основний геймплей.

Перше, що побачить на ній гравець – це інтерфейс (рис. 3.9), що складається з датчика здоров'я об'єкта (смайлик, що повільно заповнюється темним кольором, якщо у гравця потрапляє снаряд), датчика досвіду для розблокування навички (біла змійка, що також заповнюється темним, якщо компаньйон гравця збирає темний вогник – елемент досвіду – з підлоги), та датчика циклу навичок (гілка з трьома кружками, що заповнюються по мірі виконання трьох навичок для того, щоб гравець відслідковував час їх виконання та не пропустив момент, коли зможе знову викликати цикл).

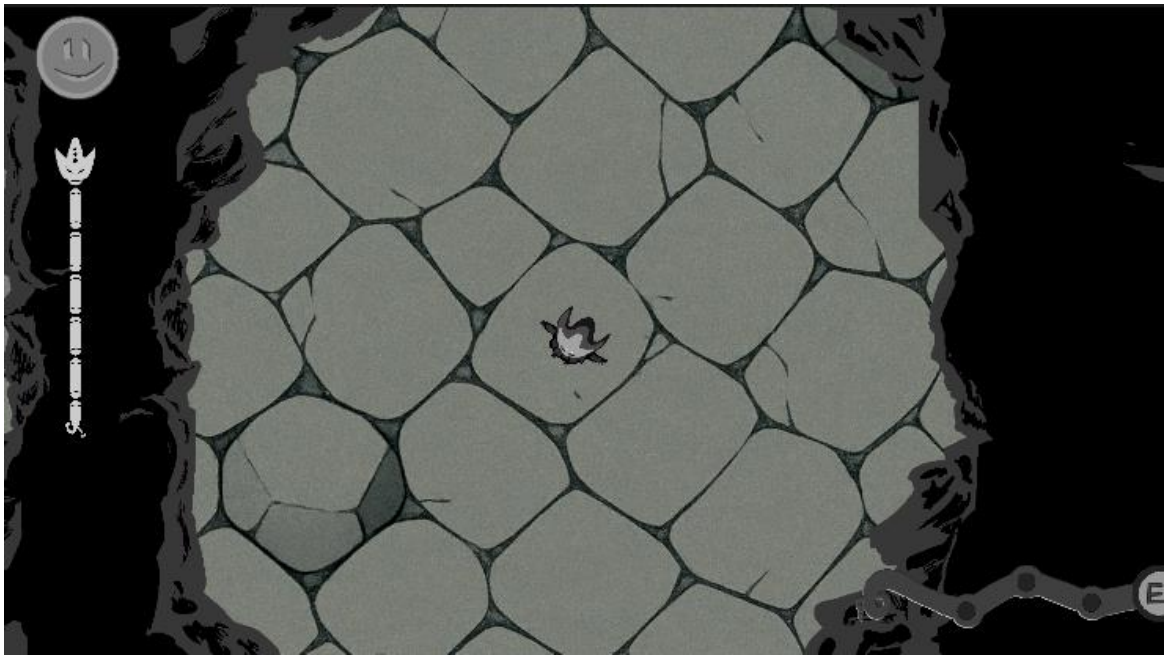

Рисунок 3.9 – Вигляд ігрового інтерфейсу (датчик здоров'я, досвіду гравця, а також порядку виконання циклу навичок), стартової кімнати, в якій немає ворогів, та гравця

На датчику циклу навичок є маленький елемент підказки – буква «E», саме яка нас і відправляє на екран навичок, що зображений на рисунку 3.10.

Така підказка виконує роль інтерактивного індикатора для користувача, спрощуючи навігацію по інтерфейсу гри та роблячи керування більш інтуїтивним.

Рисунок 3.10 – Вигляд екрану навичок, що відображається разом з плавним фейдінгом – саме тут можна вказати цикл навичок з доступних

За сам цикл і його виконання відповідає один скрипт, разом з їх розблокуванням, а за їх роботу зовсім інший. Тут функції викликаються одна за одним, а саме які це будуть навички – вирішує лише гравець, що ставить їх в певному порядку у слоти. Так, на рисунку 3.11, видно, що комбінувати їх можна по-різному.

Рисунок 3.11 – Вигляд екрану навичок, коли певна їх кількість розблокована, та у випадковому порядку вставлена у активні слоти

Вони – це draggable-об’єкти, що означає, що їх можна переміщувати до слотів власноруч. Коли вони активні (знаходяться у слотах), їх можна лише скинути шляхом натискання ПКМ. Після чого змінити обрані.

Саме скрипт з додатку Б відповідає за порядок викликання навичок, реалізацію функції їх переміщення до слотів та зчитування цього.

Опис роботи самих навичок представлений у скрипті з додатку В.

До кого ж прикріплюються ці навички? Так, вони не всі застосовуються за персонажа, оскільки у нього є компаньйон – вогник Арекс, що й буде атакувати ворогів та збирати досвід у вигляді темних вогників. Для прикладу, на рисунку 3.12 можна побачити цього компаньйона, що з’являється з гравця та летить за курсором миші, якщо ЛКМ натиснутий. У інакшому випадку – він летить назад до гравця та зникає в ньому.

Рисунок 3.12 – Вигляд компаньйона (білий вогник), який слідує за гравцем, ворога з анімацією атаки, що запустив темний снаряд, а також досвіду (чорний вогник), який може підібрати компаньйон Арекс

Поки навички не встановлені у слотах, компаньйон просто слідує за курсором, проте саме навички вдосконалюють його роботу. Тепер це не лише компаньйон для атаки поблизу нього, але й для захисту від снарядів (рис. 3.13).

На зображеному рисунку видно, як активувалось дві навички – спочатку щит, а потім вогонь, а також те, як виглядає заповнення гілки циклу навичок.

Тобто, вона говорить нам яка саме навичка активна у даний момент.

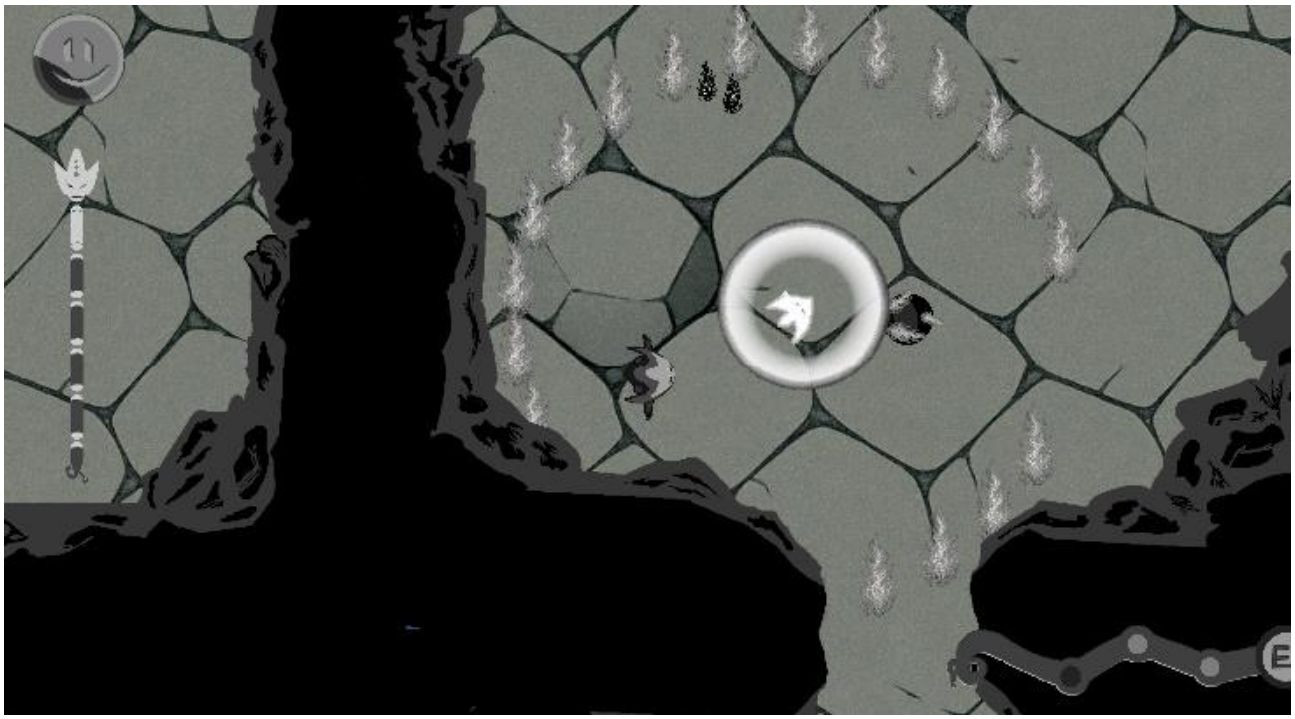

Рисунок 3.13 – Вигляд навички у першому слоті та тої, що зараз у другому, а також анімації згорання ворога

Unity унікальний також у тому, як тут можна викликати якийсь об’єкт за допомогою ЛКМ. Наприклад, у алгоритмічному рушії Construct 2 це робиться у декілька рухів [2].

Проте там не можна настільки детально вказувати умови виклику, як знаючи мову програмування C# [1].

У лістингу 3.8 наглядно представлений процес виклику компаньйона, що не дуже короткий, проте простий по структурі коду. Це базова гра зі змінною активності об’єкту у Unity.

Лістинг 3.8 – Кодова реалізація випуску компаньйона Арекса у світ

```
    void Update()
    {
        if (Input.GetMouseButtonDown(0))
        {
            if (currentCompanion != null && !currentCompanion.activeSelf)
            {
                currentCompanion.transform.position = transform.position;
                currentCompanion.SetActive(true);

currentCompanion.GetComponent<ArexsBehaviour>().SetPlayer(transform);
            }
            if (tempTarget == null)
            {
                tempTarget = new GameObject("TempTarget");
            }
        }
        if (Input.GetMouseButton(0) && currentCompanion != null &&
currentCompanion.activeSelf)
        {
            Vector3 mouseWorldPos =
Camera.main.ScreenToWorldPoint(Input.mousePosition);
            mouseWorldPos.z = 0f;

            tempTarget.transform.position = mouseWorldPos;
currentCompanion.GetComponent<ArexsBehaviour>().SetTarget(tempTarget.tra
nsform);
        }
        if (Input.GetMouseButtonUp(0))
        {
            if (tempTarget != null)
            {
                Destroy(tempTarget);
                tempTarget = null;
            }

currentCompanion.GetComponent<ArexsBehaviour>().SetTarget(transform,
true);
        }
    }
```

кінець лістингу 3.8

У структурі видно, що функцію виклику компаньйона розміщено до Update(). Таким чином вона завжди готова до його виклику.

Також розглянемо як проходить перехід від рівня з полем зору зверху та рівнем з полем зору збоку.

Дана функція є додатковою у проекті і представлена для того, щоб продемонструвати суміжність обох полів зору у 2D.

Фактично на кодовому рівні поле зору збоку відрізнається від поля зору зверху тим, що у першому присутній так званий елемент гравітації, що притягує об’єкти до землі, а також через це гравець обмежується у переміщенні за допомогою клавіатури лише по трьом, а іноді і по двом (якщо стрибок недоступний) полям виміру.

Те, як виглядає додатковий рівень з іншим полем зору, перед тим, як потрапити або у підземелля зовсім іншого типу, або у головне меню в знак виходу з нього, представлено на рисунку 3.14.

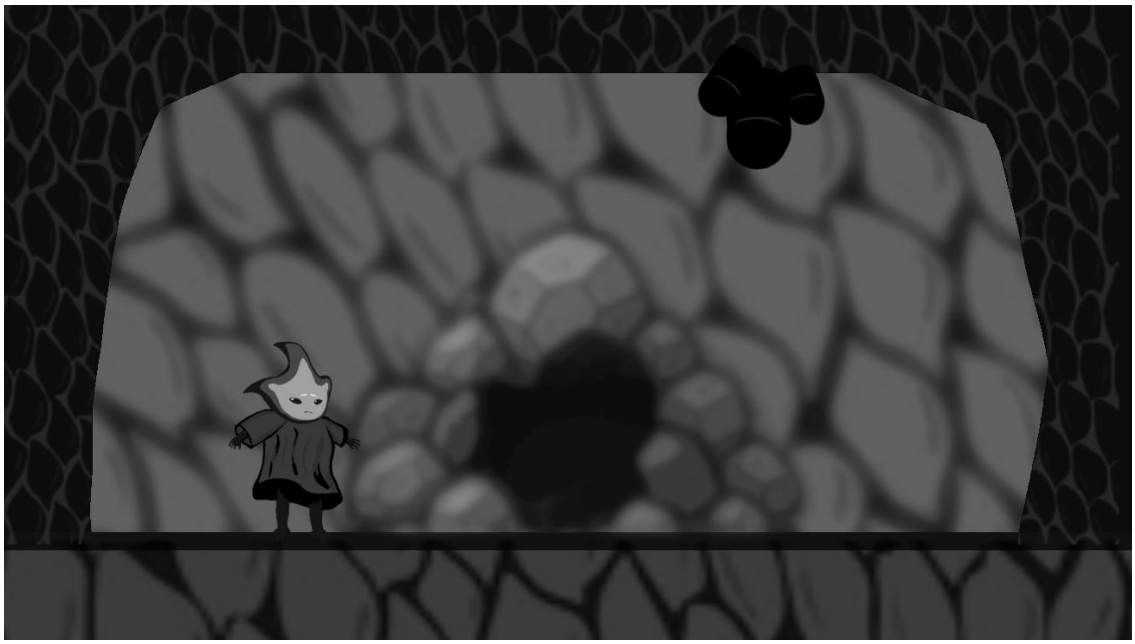

Рисунок 3.14 – Вигляд додаткового рівню з трьома ворогами, розміщеними по іншій осі, та вигляд гравця з бокової перспективи

Перед тим, як нас переміщує далі – або у наступне підземелля, або у меню, проходить п’ять секунд, за які ми маємо можливість знищити додаткових ворогів по осі Y. Саме таким чином наш проект зациклюється і отримує перспективу для подальшого розширення при можливості.

3.2 Тестування та налагодження інформаційно-комп’ютерної системи

Рівень з підземеллям з видом зверху дуже сильно урізноманітнений по кількості скриптів, які потребують іноді детального налагодження. У таблиці 3.1 представлено скрипти цієї сцени та те, за що відповідає кожний.

Таблиця 3.1 – Скрипти та їх призначення на цьому процедурному рівні

Назва скрипту	Функція, яку він виконує
ArexsBehaviour.cs	Відповідає за поведінку компаньйона Арекса
ArexsSummon.cs	Відповідає за появу компаньйона
CameraFollow.cs	Плавне слідування камери за префабом гравця
CursorManager.cs	Кастомізований курсор та пов’язані з ним елементи
DarkBallController.cs	Скрипт-контролер снаряду, що летить в гравця
DarkerAI.cs	Скрипт-контролер інтелекту ворогів
DarkerAnimationEvents.cs	Відповідає за івенти, прив’язані до анімації ворогів
DarkExpController.cs	Поява темних вогників – досвіду для навичок гравця
DLHitHandler.cs	Елемент зчитування колізії снаряду з гравцем для зарахування попадання
DungeonGenerator.cs	Відповідає за процедурну генерацію підземелля, ворогів, гравця, а також кімнати виходу
ExitRoomTrigger.cs	Скрипт-триггер колізії об’єкту виходу з гравцем
LightBallProjectile.cs	Відповідає за поведінку білих снарядів навички
PlayerController.cs	Контролер гравця, що також встановлює колізії
SkillController.cs	Персоналізація самих навичок
SkillTreeUi.cs	Скрипт порядку спрацювання навичок
UISceneController.cs	Скрипт, що відповідає за елементи інтерфейсу

Об’єктом тестування слід встановити DungeonGenerator.cs, оскільки дуже важливо як саме генеруються кімнати і чи немає помилок у логіці – чи кімнати генеруються дійсно випадково, чи прив’язуються до кімнат ключі виходів-входів, чи поєднані ці виходи-входи між собою, чи генерується додатковий шар пустих стін, щоб обмежити поле гравця, тощо.

Так можна виділити основний тип підземелля, який генерується за допомогою скрипту, на рисунку 3.15. Він структурований вірно, кількість кімнат завжди рівна.

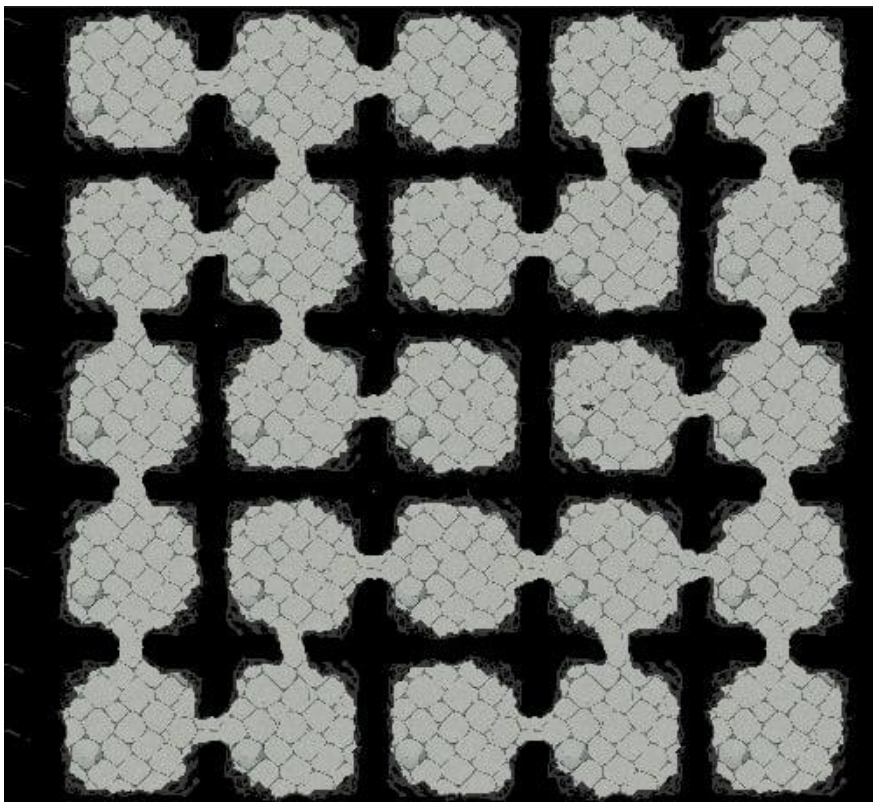

Рисунок 3.15 – Вигляд основного типу процедурно згенерованого рівню

Іноді зустрічається в дечому інший тип генерації. Якщо жоден з входів не йде до кімнати, вона пуста. Це чудово видно на рисунку 3.16.

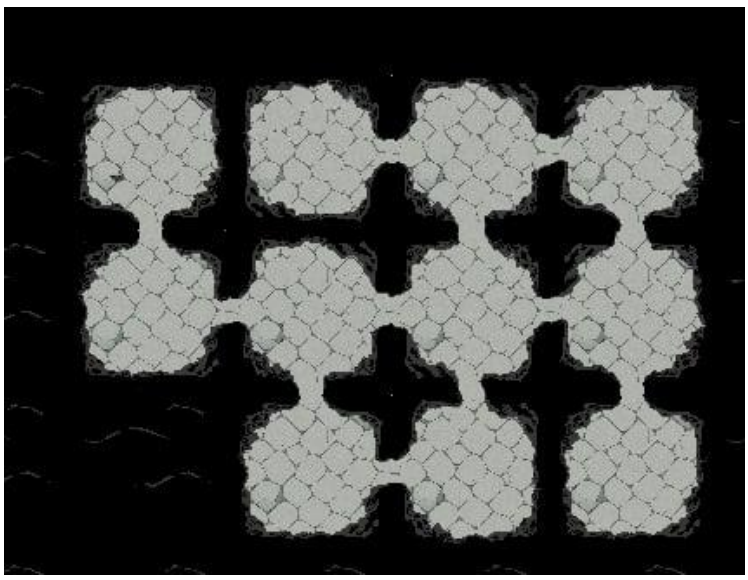

Рисунок 3.16 – Вигляд другого типу процедурно згенерованого рівню, меншого за основний, а також з відсутньою кімнатою

Такий функціонал заздалегідь продуманий скриптом, тому все працює логічно і так, як потрібно.

Проте, іноді пустих кімнат трохи більше (рис. 3.17).

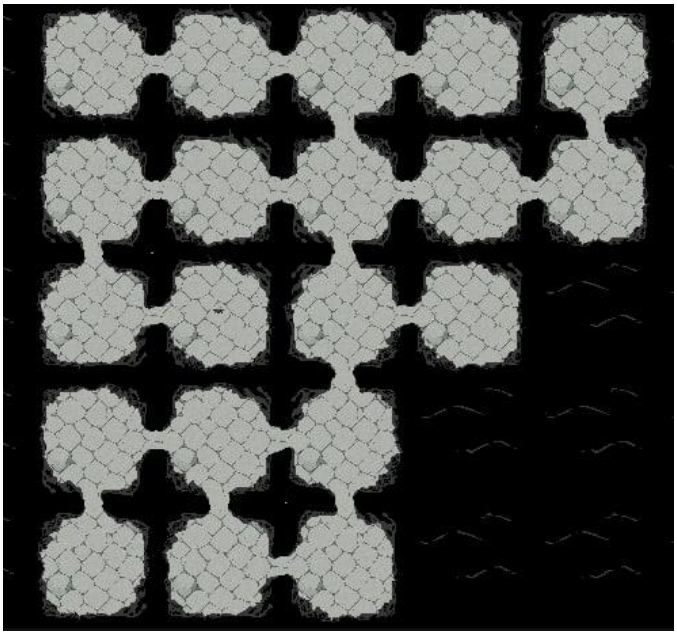

Рисунок 3.17 – Вигляд другого типу процедурно згенерованого рівню, меншого за основний, а також з відсутньою кімнатою

Така робота скрипта також продумана, оскільки проблем в логіці не знайдено. На рисунку також видно, що гравець завжди з’являється у кімнатах з одним виходом.

Отже, протестований скрипт працює як потрібно, і навіть елемент рандомізації не повпливав на появу проблем в його роботі.

3.3 Розробка інсталяційного пакета

Інсталяційний пакет у випадку нашої роботи передбачає єдиний .exe-файл, який підвантажує структуру файлів експортованого з Unity 6 білду нашого проекту. Структура файлів білду зображена на рисунку 3.18.

Це спрощує доступ до продукту та покращує загальний користувацький досвід.

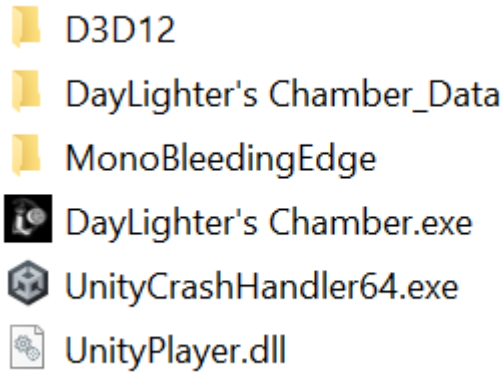

Рисунок 3.18 – Структура файлів білду проекту «DayLighter’s Chamber»

Він завантажується і працює коректно, проте це ще не все. Щоб створити саме інсталятор даного проект, потрібно використати дочірню програму – наприклад, Inno Setup [14].

Це компілятор, що базується на мові програмування Pascal (Delphi). В ньому генерується спеціальний скрипт, що компілює наші файли у єдиний інсталяційний пакет. Сам скрипт – DayLighter_Compiller.iss. Його зображено разом з інсталятором поруч з папкою, в якій знаходиться білд проекту, на рисунку 3.19.

Кожна сцена більше не є замкненим контейнером, а радше вузлом у гнучкій оповідальній сітці, де можливі як лінійні переходи, так і розгалуження.

Рисунок 3.19 – Папка білду проекту «DayLighter’s Chamber», скрипт для створення його інсталятору та сам інсталятор

В скрипту можна вказати назву проекту, його версію, автора, посилання на файли, на іконку (по замовчуванню береться та, що стоїть на .exe білду проекту, проте можна задати й кастомізовану (рис. 3.20).

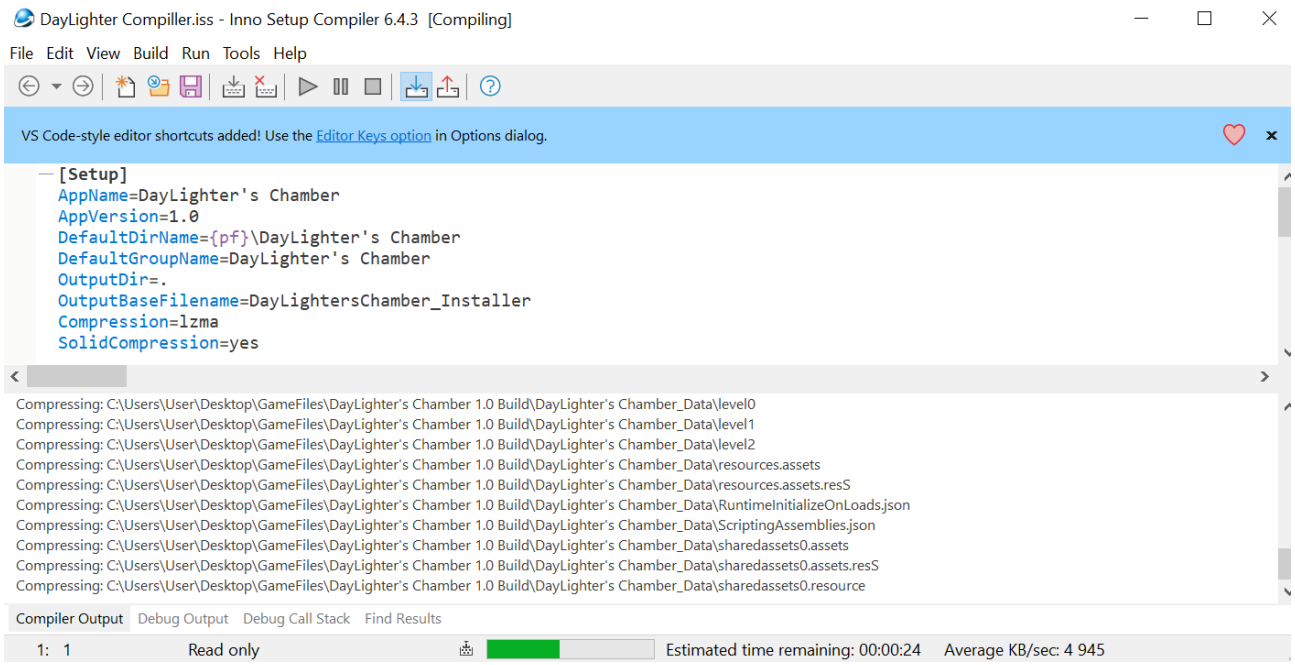

Рисунок 3.20 – Процес компіляції скрипту

У результаті роботи маємо інсталятор, представлений вище, і після запуску нашого інсталятора отримуємо готовий .exe на робочому столі, з посиланням на .exe білду, куди було встановлено сам проект при інсталяторі.

Те, як виглядає сам .exe-файл, можна побачити на рисунку 3.21. Слід зазначити, що для проекту також була створена кастомізована іконка.

Рисунок 3.21 – Готовий .exe-файл встановленого проекту «DayLighter’s Chamber» за допомогою створеного інсталятора

Приблизний код для скрипта не важкий, вся документація є на офіційному сайті Inno Setup і вже за п’ять хвилин можна написати код, що представлений у лістингу 3.9.

Лістинг 3.9 – Структура файлу скрипту

```
[Setup]
AppName=DayLighter's Chamber
AppVersion=1.0
DefaultDirName={pf}\DayLighter's Chamber
DefaultGroupName=DayLighter's Chamber
OutputDir=.
OutputBaseFilename=DayLightersChamber_Installer
Compression=lzma
SolidCompression=yes

[Files]
Source: "DayLighter's Chamber 1.0 Build\DayLighter's Chamber.exe";
DestDir: "{app}"; Flags: ignoreversion
Source: "DayLighter's Chamber 1.0 Build\UnityPlayer.dll"; DestDir:
"{app}"; Flags: ignoreversion
Source: "DayLighter's Chamber 1.0 Build\UnityCrashHandler64.exe";
DestDir: "{app}"; Flags: ignoreversion
Source: "DayLighter's Chamber 1.0 Build\DayLighter's Chamber_Data\*";
DestDir: "{app}\DayLighter's Chamber_Data"; Flags: recursesubdirs
ignoreversion
Source: "DayLighter's Chamber 1.0 Build\MonoBleedingEdge\*"; DestDir:
"{app}\MonoBleedingEdge"; Flags: recursesubdirs ignoreversion
Source: "DayLighter's Chamber 1.0 Build\D3D12\*"; DestDir: "{app}\D3D12";
Flags: recursesubdirs ignoreversion

[Icons]
Name: "{group}\DayLighter's Chamber"; Filename: "{app}\DayLighter's
Chamber.exe"
Name: "{commondesktop}\DayLighter's Chamber"; Filename:
"{app}\DayLighter's Chamber.exe"; Tasks: desktopicon

[Tasks]
Name: "desktopicon"; Description: "Створити ярлик на робочому столі";
GroupDescription: "Додаткові задачі:"    }
```

кінець лістингу 3.9

Подана реалізація цього коду пришвидшує процес компіляції інсталятора, змінюючи лише кілька ключових параметрів у шаблоні.

3.4 Розробка документації для програмного продукту

Документація до проекту «Daylighter’s Chamber» охоплює повний цикл взаємодії користувача з програмним продуктом: від його встановлення до користування та видалення. Особлива увага була приділена саме інсталяції, яка реалізована за допомогою інсталяційного пакета, розробленого у Inno Setup – інструменті, що дозволяє створити єдиний .exe-файл інсталятора на базі скриптів мовою Pascal [12].

Це основний елемент, якому варто приділити увагу у документації до встановлення програмного продукту.

До складу інсталятора входять усі необхідні елементи розробки, зібрані з білду, експортованого з Unity 6. Усі файли зберігаються у визначеній директорії, до якої посилається компілятор Inno Setup через власноруч написаний скрипт DayLighter_Compiller.iss.

Після компіляції скрипту формується інсталятор, який автоматично копіює всі необхідні файли гри у вибрану директорію (це його призначення), створює ярлик на робочому столі та в меню «Пуск», а також дозволяє зручно видалити гру через «Програми та компоненти» Windows.

Саме він розповсюджується на відповідних Інтернет-платформах, та у подальшому потребує встановлення на власному персональному комп’ютері.

Документація користувача, яка йде в комплекті разом з інсталятором у архівному zip-файлі, подана у вигляді .pdf-файлу (рис. 3.22), який містить:

- короткий опис гри;
- інструкції з керування;
- системні вимоги;
- інструкції з встановлення/видалення;
- інформацію про підтримку та контакти;
- деталі авторського права гри.

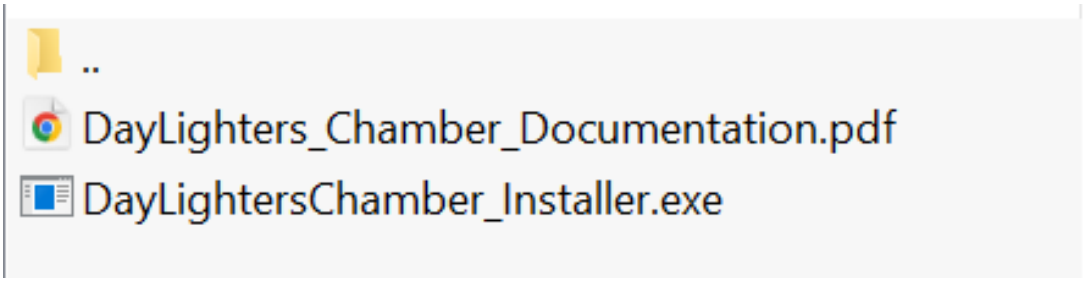

Рисунок 3.22 – Наповнення архіву проекту «DayLighter’s Chamber», що містить інсталятор актуальної версії білду гри, а також документацію до неї

Розроблена документація забезпечує розуміння основної функції інсталятора – надійного розгортання гри без потреби вручну копіювати чи переміщувати файли.

Таким чином, користувач розуміє та бачить, щоо весь процес є автоматизованим та зручним, і з більшою вірогідністю буде активніше слідкувати за розвитком цього проекту у майбутньому.

ВИСНОВКИ

У кваліфікаційній роботі було реалізовано повний цикл створення 2D-гри «DayLighter’s Chamber» у жанрі horror-like із використанням рушія Unity 6, що включає унікальні механіки зміни поля зору, подачі сюжету через коміксні панелі та процедурної генерації рівнів із циклічним викликом навичок.

На початковому етапі проведено огляд та детальний аналіз існуючих розробок у жанрі 2D-horror на платформі Unity, а також наявних програмних рішень і підходів для реалізації ігрової логіки мовою C#, що дало змогу обрати оптимальні інструменти для розробки.

Була спроектована майбутня структура гри у вигляді детальних схем, які відображають взаємозв’язки між основними сценами, системами ігрового процесу та елементами інтерфейсу, що слугують основою для подальшої розробки та тестування.

Розроблено архітектуру гри, що включає механізми процедурної генерації рівнів, систему циклу навичок, а також логіку взаємодії між персонажем, компаньйоном і ворогами, що реалізує динамічну ігрову поведінку.

Особливу увагу приділено реалізації механіки зміни перспективи – поля зору гравця, яке динамічно змінюється під час переходу між рівнями (на одному рівні – поле зору зверху, на іншому – поле зору збоку), що підсилює ігрову атмосферу і тактичний аспект геймплею.

Створено механіку подачі сюжету у вигляді коміксних панелей, які з’являються у визначені моменти ігрового часу або при взаємодії з ключовими об’єктами, що дозволяє розкривати сюжет через візуальні засоби без нав’язливих текстових вставок.

Реалізовано поведінку керованого компаньйона на ім’я Аркес, який реагує на ворогів із врахуванням їх дистанції та динамічно підтримує гравця, що додає глибині ігровому процесу і покращує тактичні можливості.

Проведено комплексне тестування гри, виявлено та виправлено низку помилок, що забезпечило стабільну роботу проекту, а також покращило якість ігрового досвіду.

Розроблено унікальний дизайн та стилістику гри з урахуванням отриманого досвіду під час навчання, що дозволило сформувати атмосферу, відповідну жанру horror-like, та уникнути порушень авторських прав при використанні сторонніх текстур і спрайтів.

Створено інсталяційний пакет за допомогою Inno Setup та підготовлено детальну документацію, що полегшує розповсюдження гри та забезпечує користувачам простий і зручний спосіб інсталяції та запуску проекту.

Для забезпечення подальшого розвитку проекту проведена модернізація коду із врахуванням можливості масштабування, розширення функціоналу та залучення співрозробників, що робить гру перспективною для комерційного або освітнього застосування.

Проект «DayLighter’s Chamber» успішно поєднує інноваційні технічні рішення, оригінальний сюжетний підхід і атмосферний дизайн, демонструючи високий рівень володіння навичками розробки ігор і готовність автора до професійної діяльності у сфері ігрової індустрії.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. C# Guide. Microsoft. URL: https://learn.microsoft.com/en-us/dotnet/csharp/ (дата звернення: 03.04.2025).

2. Construct 2 Manuals and Documentation. Scirra. URL: https://www.construct.net/en/construct-2/manuals (дата звернення: 05.04.2025).

3. Darkwood. Acid Wizard Studio. URL: https://en.wikipedia.org/wiki/Darkwood (дата звернення: 08.03.2025).

4. Documentation for Visual Studio Code. Microsoft. URL: https://code.visualstudio.com/docs (дата звернення: 12.04.2025).

5. Don’t Starve. Klei Entertainment. URL: https://en.wikipedia.org/wiki/Don%27t_Starve (дата звернення: 16.03.2025).

6. Endless War – Roguelike RPG. Game Dev Team. URL: https://play.google.com/store/apps/details?id=com.gamedevteam.endlesswar (дата звернення: 24.03.2025).

7. Lone Survivor. Superflat Games. URL: https://en.wikipedia.org/wiki/Lone_Survivor_(video_game) (дата звернення: 08.03.2025).

8. Mad Father. Sen. URL: https://en.wikipedia.org/wiki/Mad_Father (дата звернення: 16.03.2025).

9. Otherworld Legends. ChillyRoom. URL: https://play.google.com/store/apps/details?id=com.chillyroom.zhmr.gp (дата звернення: 24.03.2025).

10. Steam – Store & Community Platform. Valve Corporation. URL: https://store.steampowered.com (дата звернення: 18.05.2025).

11. The Witch’s House MV. Fummy. URL: https://en.wikipedia.org/wiki/The_Witch%27s_House_MV (дата звернення: 17.03.2025).

12. Unity Asset Store. Unity Technologies. URL: https://assetstore.unity.com (дата звернення: 16.03.2025).

13. Unity Documentation. Unity Technologies. URL: https://docs.unity.com/ (дата звернення: 11.03.2025).

14. InnoHelp – Documentation for Inno Setup. Leserg73. URL: https://leserg73.github.io/InnoHelp/ (дата звернення: 17.05.2025).

15. Itch.io – Indie game hosting & distribution. Leaf Corcoran. URL: https://itch.io (дата звернення: 18.05.2025).

ДОДАТКИ

Додаток А – Лістинг скрипту з кодом процедурної генерації

DungeonGenerator.cs

```
using UnityEngine;
using System.Collections.Generic;
using UnityEngine.Tilemaps;
using System.Linq;

public class DungeonGenerator : MonoBehaviour
{
    public int width = 5;
    public int height = 5;
    public static DungeonGenerator Instance;
    public float roomSpacing = 10f;
    public int minEnemiesPerRoom = 1;
    public int maxEnemiesPerRoom = 3;
    public bool gmselector = false;
    public RoomInstance playerStartRoom;
    public RoomInstance exitRoom;
    public Tilemap FloorTilemap;
    public TileBase FloorTile;

    [System.Serializable]
    public class RoomPrefab
    {
        public string id;
        public GameObject prefab;
        public HashSet<char> exits => new HashSet<char>(id.ToCharArray());
    }
    public List<RoomPrefab> roomPrefabs;
    public GameObject emptyRoom;
    public GameObject exitPrefab;
    public GameObject darkerPrefab;
    private RoomPrefab[,] placedRooms;
    void Start()
    {
        placedRooms = new RoomPrefab[width, height];
        GenerateDungeon();
        EnsureConnectivity();
        InstantiateDungeon();
        SpawnOuterEmptyRooms();
        SpawnPlayer();
        SpawnLevelExit();
        SpawnEnemiesInAllRooms();
    }
    private void Awake()
    {
        Instance = this;
    }
    void GenerateDungeon()
    {
        for (int x = 0; x < width; x++)
        {
            for (int y = 0; y < height; y++)
            {
                var valid = GetValidRoomsForPosition(x, y);
                if (valid.Count == 0)
                {
                    placedRooms[x, y] = null;
```

```
                    continue;
                }
                RoomPrefab chosen = valid[Random.Range(0, valid.Count)];
                placedRooms[x, y] = chosen;
            }
        }
    }
    void EnsureConnectivity()
    {
        bool[,] visited = new bool[width, height];
        Queue<Vector2Int> queue = new Queue<Vector2Int>();
        Vector2Int start = new Vector2Int(width / 2, height / 2);
        if (placedRooms[start.x, start.y] == null)
        {
            return;
        }
        queue.Enqueue(start);
        visited[start.x, start.y] = true;
        while (queue.Count > 0)
        {
            var pos = queue.Dequeue();
            foreach (var dir in placedRooms[pos.x, pos.y].exits)
            {
                Vector2Int neighbor = GetNeighbor(pos, dir);
                if (!IsInside(neighbor)) continue;
                if (visited[neighbor.x, neighbor.y]) continue;
                if (placedRooms[neighbor.x, neighbor.y] == null) continue;
                char opposite = GetOpposite(dir);
                if (!placedRooms[neighbor.x, neighbor.y].exits.Contains(opposite))
continue;
                visited[neighbor.x, neighbor.y] = true;
                queue.Enqueue(neighbor);
            }
        }
        for (int x = 0; x < width; x++)
            for (int y = 0; y < height; y++)
            {
                if (!visited[x, y] && placedRooms[x, y] != null)
                {

                    placedRooms[x, y] = null;
                }
            }
    }
    void SpawnOuterEmptyRooms()
    {
        int outerLayer = 2;
        for (int x = -outerLayer; x < width + outerLayer; x++)
        {
            for (int y = -outerLayer; y < height + outerLayer; y++)
            {
                if (x >= 0 && x < width && y >= 0 && y < height)
                    continue;
                Vector3 spawnPos = new Vector3(x * roomSpacing, y * roomSpacing, 0f);
                if (emptyRoom != null)
                {
                    Instantiate(emptyRoom, spawnPos, Quaternion.identity);

                }
            }
        }
    }
```

```
    void InstantiateDungeon()
    {
        FloorTilemap.ClearAllTiles();

        for (int x = 0; x < width; x++)
            for (int y = 0; y < height; y++)
            {
                Vector3 spawnPos = new Vector3(x * roomSpacing, y * roomSpacing, 0f);
                if (placedRooms[x, y] != null)
                {
                    Instantiate(placedRooms[x, y].prefab, spawnPos,
Quaternion.identity);
                    Debug.Log($"Spawned room {placedRooms[x, y].id} at {x},{y}");
                    placedRoomInstances.Add(new RoomInstance(placedRooms[x, y], new
Vector2Int(x, y)));
                    GenerateFloorForRoom(x, y, spawnPos);
                }
                else if (emptyRoom != null)
                {
                    Instantiate(emptyRoom, spawnPos, Quaternion.identity);
                    Debug.Log($"Spawned empty room at {x},{y}");
                }
            }
    }
    void GenerateFloorForRoom(int roomX, int roomY, Vector3 roomWorldPos)
    {
        float roomSizeUnits = 10f;
        float tileSizeUnits = 10f;  // 1024/102.4 = 10
        int tilesCountX = Mathf.CeilToInt(roomSizeUnits / tileSizeUnits);
        int tilesCountY = Mathf.CeilToInt(roomSizeUnits / tileSizeUnits);
        Vector3Int tilemapOrigin = FloorTilemap.WorldToCell(roomWorldPos);
        for (int tx = 0; tx < tilesCountX; tx++)
            for (int ty = 0; ty < tilesCountY; ty++)
            {
                Vector3Int tilePos = new Vector3Int(tilemapOrigin.x + tx,
tilemapOrigin.y + ty, 0);

                FloorTilemap.SetTile(tilePos, FloorTile);
            }
    }
    List<RoomPrefab> GetValidRoomsForPosition(int x, int y)
    {
        List<RoomPrefab> validRooms = new List<RoomPrefab>();
        foreach (var room in roomPrefabs)
        {

            if ((y == height - 1 && room.exits.Contains('U')) ||
                (y == 0 && room.exits.Contains('D')) ||
                (x == 0 && room.exits.Contains('L')) ||
                (x == width - 1 && room.exits.Contains('R')))
            {
                continue;
            }
            if (IsCompatible(room, x, y))
                validRooms.Add(room);
        }
        return validRooms;
    }
    bool IsCompatible(RoomPrefab room, int x, int y)
    {
        if (y < height - 1 && placedRooms[x, y + 1] != null)
        {
```

```
            var neighbor = placedRooms[x, y + 1];
            if (neighbor.exits.Contains('D') != room.exits.Contains('U')) return false;
        }
        if (y > 0 && placedRooms[x, y - 1] != null)
        {
            var neighbor = placedRooms[x, y - 1];
            if (neighbor.exits.Contains('U') != room.exits.Contains('D')) return false;
        }
        if (x > 0 && placedRooms[x - 1, y] != null)
        {
            var neighbor = placedRooms[x - 1, y];
            if (neighbor.exits.Contains('R') != room.exits.Contains('L')) return false;
        }
        if (x < width - 1 && placedRooms[x + 1, y] != null)
        {
            var neighbor = placedRooms[x + 1, y];
            if (neighbor.exits.Contains('L') != room.exits.Contains('R')) return false;
        }

        return true;
    }
    void SpawnEnemiesInAllRooms()
    {
        foreach (RoomInstance room in placedRoomInstances)
        {

            if (room == playerStartRoom || room == exitRoom)
                continue;
            int enemyCount = Random.Range(minEnemiesPerRoom, maxEnemiesPerRoom + 1);
            HashSet<Vector2Int> usedSectors = new HashSet<Vector2Int>();
            for (int i = 0; i < enemyCount; i++)
            {
                Vector2Int sector;
                int attempts = 0;
                do
                {
                    sector = GetRandomInnerSector();
                    attempts++;
                }
                while (usedSectors.Contains(sector) && attempts < 10);
                usedSectors.Add(sector);
                Vector3 roomWorldPos = new Vector3(room.gridPosition.x * roomSpacing,
room.gridPosition.y * roomSpacing, 0f);
                float sectorUnitSize = roomSpacing / sectorSize;
                Vector3 sectorOffset = new Vector3(
                    (-roomSpacing / 2f) + sector.x * sectorUnitSize + sectorUnitSize /
2f,
                    (-roomSpacing / 2f) + sector.y * sectorUnitSize + sectorUnitSize /
2f,
                    0f
                );
                Vector3 spawnPos = roomWorldPos + sectorOffset;
                GameObject enemyInstance = Instantiate(darkerPrefab, spawnPos,
Quaternion.identity);
                float randomScale = Random.Range(0.6f, 1f);
                enemyInstance.transform.localScale = new Vector3(randomScale,
randomScale, 1f);

            }
        }
    }
    void SpawnLevelExit()
```

```
    {
        var oneExit = placedRoomInstances.Where(r => r.prefab.exits.Count == 1 && r !=
playerStartRoom).ToList();
        var twoExit = placedRoomInstances.Where(r => r.prefab.exits.Count == 2 && r !=
playerStartRoom).ToList();
        List<RoomInstance> possibleRooms = oneExit.Count > 0 ? oneExit : twoExit;
        if (possibleRooms.Count == 0)
        {
            return;
        }
        RoomInstance chosenRoom = possibleRooms[Random.Range(0, possibleRooms.Count)];
        Vector3 roomWorldPos = new Vector3(chosenRoom.gridPosition.x * roomSpacing,
chosenRoom.gridPosition.y * roomSpacing, 0f);
        Vector2Int sector = new Vector2Int(2, 3);
        float sectorUnitSize = roomSpacing / sectorSize;
        Vector3 sectorOffset = new Vector3(
            (-roomSpacing / 2f) + sector.x * sectorUnitSize + sectorUnitSize / 2f,
            (-roomSpacing / 2f) + sector.y * sectorUnitSize + sectorUnitSize / 2f,
            0f
        );
        exitRoom = chosenRoom;
        Vector3 spawnPos = roomWorldPos + sectorOffset;
        Instantiate(exitPrefab, spawnPos, Quaternion.identity);
    }
    void SpawnPlayer()
    {
        var oneExit = placedRoomInstances.Where(r => r.prefab.exits.Count ==
1).ToList();
        var twoExit = placedRoomInstances.Where(r => r.prefab.exits.Count ==
2).ToList();

        List<RoomInstance> possibleRooms = oneExit.Count > 0 ? oneExit : twoExit;
        if (possibleRooms.Count == 0)
        {
            return;
        }
        RoomInstance chosenRoom = possibleRooms[Random.Range(0, possibleRooms.Count)];
        Vector2Int sector = GetRandomInnerSector();
        Vector3 roomWorldPos = new Vector3(chosenRoom.gridPosition.x * roomSpacing,
chosenRoom.gridPosition.y * roomSpacing, 0f);
        float sectorUnitSize = roomSpacing / sectorSize;
        Vector3 sectorOffset = new Vector3(
            (-roomSpacing / 2f) + sector.x * sectorUnitSize + sectorUnitSize / 2f,
            (-roomSpacing / 2f) + sector.y * sectorUnitSize + sectorUnitSize / 2f,
            0f
        );
        Vector3 spawnPos = roomWorldPos + sectorOffset;
        playerStartRoom = chosenRoom;
        GameObject playerInstance = Instantiate(playerPrefab, spawnPos,
Quaternion.identity);
        Camera.main.GetComponent<CameraFollow>().SetTarget(playerInstance.transform);


    }
    Vector2Int GetRandomInnerSector()
    {

        int min = 1;
        int max = sectorSize - 2;

        int sx = Random.Range(min, max + 1);
        int sy = Random.Range(min, max + 1);
```

```
        return new Vector2Int(sx, sy);
    }
    Vector2Int GetNeighbor(Vector2Int pos, char dir)
    {
        switch (dir)
        {
            case 'U': return new Vector2Int(pos.x, pos.y + 1);
            case 'D': return new Vector2Int(pos.x, pos.y - 1);
            case 'L': return new Vector2Int(pos.x - 1, pos.y);
            case 'R': return new Vector2Int(pos.x + 1, pos.y);
            default: return pos;
        }
    }
    char GetOpposite(char dir)
    {
        switch (dir)
        {
            case 'U': return 'D';
            case 'D': return 'U';
            case 'L': return 'R';
            case 'R': return 'L';
            default: return '?';
        }
    }
    bool IsInside(Vector2Int pos)
    {
        return pos.x >= 0 && pos.x < width && pos.y >= 0 && pos.y < height;
    }
    [System.Serializable]
    public class RoomInstance
    {
        public RoomPrefab prefab;
        public Vector2Int gridPosition;
        public GameObject roomGameObject;
        public bool gmselector = false;
        public RoomInstance(RoomPrefab prefab, Vector2Int pos)
        {
            this.prefab = prefab;
            this.gridPosition = pos;
        }
    }
    public int sectorSize = 5; // 5x5
    public List<RoomInstance> placedRoomInstances = new List<RoomInstance>();
    public GameObject playerPrefab;
}
```

Додаток Б – Лістинг скрипту з кодом порядку викликання навичок

SkillTreeUi.cs

```
using UnityEngine;
using UnityEngine.UI;
using System.Linq;
using UnityEngine.EventSystems;
using System.Collections;
using System.Collections.Generic;

public class SkillTreeUI : MonoBehaviour
{
    public CanvasGroup skillTreeGroup;
    public GameObject newSkillPrefab;
    public SkillController skillController;
    public Transform[] skillPlaceholders; // 6
    public Image[] skillSlots; // 3
    public Sprite lockedSprite; // skill_b
    public Sprite[] expProgressSprites;
    public Image levelProgressImage;
    public Sprite[] skillProgressSprites;
    public Image releaseProgressImage;
    public int unlockedSkills = 0;
    public int expPoints = 0;
    public Sprite[] skillSprites;
    public string[] skillNames = { "skill1", "skill2", "skill3", "skill4", "skill5",
"skill6" };
    private bool isTreeVisible = false;
    private bool isSkillChainRunning = false;
    private GameObject[] skillInstances = new GameObject[6];
    private GameObject[] equippedSkillObjects = new GameObject[3];
    void Start()
    {
        skillTreeGroup.alpha = 0;
        skillTreeGroup.interactable = false;
        skillTreeGroup.blocksRaycasts = false;
        Time.timeScale = 1;

        SpawnAllSkills();
        UpdateLevelProgressVisual();
    }
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.E))
        {
            ToggleSkillTree();
        }
        if (Input.GetKeyDown(KeyCode.Alpha1))
        {
            expPoints += 20;
            UpdateLevelProgressVisual();
        }

        if (!isTreeVisible && Input.GetMouseButtonDown(0) && !isSkillChainRunning)
        {
            StartCoroutine(ExecuteSkillChain());
        }
    }
    void ToggleSkillTree()
```

```
    {
        isTreeVisible = !isTreeVisible;
        StopAllCoroutines();
        StartCoroutine(FadeSkillTree(isTreeVisible));

        if (isTreeVisible)
            PauseGameLogic();
        else
            ResumeGameLogic();
    }
    void PauseGameLogic()
    {
        foreach (var pausable in
FindObjectsOfType<MonoBehaviour>().OfType<IPausable>())
        {
            pausable.Pause();
        }
    }
    void ResumeGameLogic()
    {
        foreach (var pausable in
FindObjectsOfType<MonoBehaviour>().OfType<IPausable>())
        {
            pausable.Unpause();
        }
    }
    IEnumerator FadeSkillTree(bool fadeIn)
    {
        float duration = 0.15f;
        float start = skillTreeGroup.alpha;
        float end = fadeIn ? 1f : 0f;
        float t = 0;
        skillTreeGroup.interactable = fadeIn;
        skillTreeGroup.blocksRaycasts = fadeIn;
        while (t < duration)
        {
            t += Time.unscaledDeltaTime;
            skillTreeGroup.alpha = Mathf.Lerp(start, end, t / duration);
            yield return null;
        }
        skillTreeGroup.alpha = end;
    }
    void SpawnAllSkills()
    {
        for (int i = 0; i < skillPlaceholders.Length; i++)
        {
            GameObject newSkill = Instantiate(newSkillPrefab,
skillTreeGroup.transform);
            newSkill.GetComponent<RectTransform>().anchoredPosition =
skillPlaceholders[i].GetComponent<RectTransform>().anchoredPosition;
            Image img = newSkill.GetComponent<Image>();
            img.sprite = lockedSprite;
            NewSkillDrag dragScript = newSkill.AddComponent<NewSkillDrag>();
            dragScript.Init(this, i);
            dragScript.SetDraggable(false);
            skillInstances[i] = newSkill;
        }
    }
    public void GainExp()
    {
        if (unlockedSkills >= skillInstances.Length) return;
        expPoints++;
```

```
        UpdateLevelProgressVisual();
        if (expPoints >= 21)
        {
            expPoints = 0;
            GameObject skillToUnlock = skillInstances[unlockedSkills];
            skillToUnlock.GetComponent<Image>().sprite = skillSprites[unlockedSkills];
            Debug.Log($"Exp: {expPoints} / Unlocked: {unlockedSkills}");
            skillToUnlock.GetComponent<NewSkillDrag>().SetDraggable(true);
            unlockedSkills++;
            UpdateLevelProgressVisual();
        }
    }
    public void UpdateLevelProgressVisual()
    {
        int frameIndex = Mathf.Clamp(expPoints / 3, 0, expProgressSprites.Length - 1);
        if (levelProgressImage != null && expProgressSprites.Length > frameIndex)
        {
            levelProgressImage.sprite = expProgressSprites[frameIndex];
        }
    }
    public void TryBindSkillToSlot(GameObject skillObject, int skillIndex,
PointerEventData pointerData)
    {
        int targetSlotIndex = -1;

        for (int i = 0; i < skillSlots.Length; i++)
        {
            if
(RectTransformUtility.RectangleContainsScreenPoint(skillSlots[i].rectTransform,
pointerData.position))
            {
                targetSlotIndex = i;
                break;
            }
        }
        if (targetSlotIndex != -1)
        {
            for (int i = 0; i < equippedSkillObjects.Length; i++)
            {
                if (equippedSkillObjects[i] == skillObject)
                {
                    equippedSkillObjects[i] = null;
                    break;
                }
            }
            equippedSkillObjects[targetSlotIndex] = skillObject;
            skillObject.transform.SetParent(skillSlots[targetSlotIndex].transform);
            skillObject.GetComponent<RectTransform>().anchoredPosition = Vector2.zero;
        }
        else
        {
            skillObject.GetComponent<RectTransform>().anchoredPosition =
skillPlaceholders[skillIndex].GetComponent<RectTransform>().anchoredPosition;
            skillObject.transform.SetParent(skillTreeGroup.transform);
        }
    }
    public void UnbindSkill(GameObject skill)
    {
        for (int i = 0; i < equippedSkillObjects.Length; i++)
        {
            if (equippedSkillObjects[i] == skill)
            {
```

```
                equippedSkillObjects[i] = null;
                Debug.Log($"Skill unbound from slot {i}");
                break;
            }
        }
        for (int i = 0; i < equippedSkillObjects.Length; i++)
        {
            Debug.Log($"Slot {i} skill: {(equippedSkillObjects[i] != null ?
equippedSkillObjects[i].name : "null")}");
        }
    }
    IEnumerator ExecuteSkillChain()
    {
        isSkillChainRunning = true;
        int currentStep = 1;
        if (equippedSkillObjects.All(x => x == null))
        {
            Debug.LogWarning("No skills equipped, skipping execution.");
            isSkillChainRunning = false;
            yield break;
        }
        foreach (GameObject skillObj in equippedSkillObjects)
        {
            if (skillObj != null)
            {
                int index = skillObj.GetComponent<NewSkillDrag>().skillIndex;
                Debug.Log("Executing: " + skillNames[index]);
                switch (index)
                {
                    case 0:
                        skillController.Skill1Active();
                        break;
                    case 1:
                        skillController.Skill2Active();
                        break;
                    case 2:
                        skillController.Skill3Active();
                        break;
                    case 3:
                        skillController.Skill4Active();
                        break;
                }
                UpdateReleaseProgressVisual(currentStep);
                currentStep++;
                yield return new WaitForSeconds(1f);
            }
        }
        UpdateReleaseProgressVisual(0);
        isSkillChainRunning = false;
    }
    private void UpdateReleaseProgressVisual(int skillIndex)
    {
        int frameIndex = Mathf.Clamp(skillIndex, 0, skillProgressSprites.Length - 1);
        if (releaseProgressImage != null && skillProgressSprites.Length > frameIndex)
        {
            releaseProgressImage.sprite = skillProgressSprites[frameIndex];
        }
    }
}

public class NewSkillDrag : MonoBehaviour, IBeginDragHandler, IDragHandler,
IEndDragHandler
```

```
{
    private SkillTreeUI skillTreeUI;
    private RectTransform rectTransform;
    private Canvas canvas;
    private CanvasGroup canvasGroup;
    private bool isInSkillSlot = false;
    private Vector2 originalPosition;
    private Transform originalParent;
    private bool isDraggingEnabled = true;
    public int skillIndex;
    public void Init(SkillTreeUI ui, int index)
    {
        skillTreeUI = ui;
        skillIndex = index;
        rectTransform = GetComponent<RectTransform>();
        canvas = GetComponentInParent<Canvas>();
        canvasGroup = gameObject.AddComponent<CanvasGroup>();
        originalParent = transform.parent;
        originalPosition = GetComponent<RectTransform>().anchoredPosition;
    }

    public void Update()
    {
        if (isInSkillSlot && Input.GetMouseButtonDown(1)) // ПКМ
        {
            transform.SetParent(originalParent);

            StartCoroutine(SmoothReturn());

            skillTreeUI.UnbindSkill(gameObject);

            isDraggingEnabled = true;
            isInSkillSlot = false;
        }
    }
    private IEnumerator SmoothReturn()
    {
        RectTransform rt = GetComponent<RectTransform>();
        Vector2 startPos = rt.anchoredPosition;
        float duration = 0.3f;
        float elapsed = 0f;

        while (elapsed < duration)
        {
            rt.anchoredPosition = Vector2.Lerp(startPos, originalPosition, elapsed /
duration);
            elapsed += Time.deltaTime;
            yield return null;
        }
        rt.anchoredPosition = originalPosition;
    }
    public void OnBeginDrag(PointerEventData eventData)
    {
        if (!isDraggingEnabled) return;

        canvasGroup.blocksRaycasts = false;
        canvasGroup.alpha = 0.6f;
        transform.SetParent(skillTreeUI.skillTreeGroup.transform);
    }
    public void SetDraggable(bool canDrag)
    {
        isDraggingEnabled = canDrag;
```

```
    }
    public void OnDrag(PointerEventData eventData)
    {
        if (!isDraggingEnabled) return;

        rectTransform.anchoredPosition += eventData.delta / canvas.scaleFactor;
    }
    public void OnEndDrag(PointerEventData eventData)
    {
        if (!isDraggingEnabled) return;

        GetComponent<CanvasGroup>().blocksRaycasts = true;
        skillTreeUI.TryBindSkillToSlot(gameObject, skillIndex, eventData);

        if (transform.parent == skillTreeUI.skillSlots[0].transform ||
            transform.parent == skillTreeUI.skillSlots[1].transform ||
            transform.parent == skillTreeUI.skillSlots[2].transform)
        {
            isDraggingEnabled = false;
            isInSkillSlot = true;
        }
        else
        {
            isDraggingEnabled = true;
            isInSkillSlot = false;
        }
    }
}
```

Додаток В – Лістинг скрипту з кодовою реалізацією самих навичок

SkillController.cs

```
using System.Collections;
using UnityEngine;

public class SkillController : MonoBehaviour
{
    [Header("Prefabs & References")]
    public GameObject arex;
    public GameObject darkBallPrefab;

    [Header("Shield Settings")]
    public float shieldDuration = 1.5f;
    public float shieldKillRadius = 1.5f;
    public GameObject playerShieldPrefab;
    private GameObject playerShieldInstance;

    [Header("LightBall Settings")]
    public GameObject lightBallPrefab;
    public float lightBallSpeed = 2f;
    public int lightBallCount = 12;
    public float lightBallLifetime = 0.1f;

    [Header("Healler Settings")]
    public SkillTreeUI skillTree;
    public void SetArex(GameObject arexInstance)
    {
        arex = arexInstance;
    }
    private void Start()
    {
        if (ArexsSummon.Instance != null)
        {
            GameObject ar = ArexsSummon.Instance.GetCurrentCompanion();
            if (ar != null)
                SetArex(ar);
        }
    }
    private void Update()
    {
        if (arex == null && ArexsSummon.Instance != null)
        {
            GameObject ar = ArexsSummon.Instance.GetCurrentCompanion();
            if (ar != null)
                SetArex(ar);
        }
    }
    public void Skill1Active()
    {
        StartCoroutine(ActivateLightShield());
    }
    public void Skill2Active()
    {
        if (arex == null)
        {
            Debug.LogWarning("Arex not enabled.");
            return;
        }
```

```
        StartCoroutine(SpawnLightBalls());
    }
    public void Skill3Active()
    {
        StartCoroutine(ActivatePlayerShield());
    }
    public void Skill4Active()
    {
        Debug.Log("Skill 4 activated");
        if (skillTree == null)
        {
            Debug.LogWarning("SkillTreeUi not found.");
            return;
        }
        if (skillTree.expPoints >= 7)
        {
            skillTree.expPoints -= 7;
            skillTree.UpdateLevelProgressVisual();
            DLHitHandler dlHandler = DLHitHandler.instance;
            if (dlHandler != null && dlHandler.dlHit >= 2)
            {
                dlHandler.dlHit = Mathf.Max(0, dlHandler.dlHit - 2);
                dlHandler.SendMessage("UpdateDLImage",
SendMessageOptions.DontRequireReceiver);
            }
            else
            {
                Debug.LogWarning("DLHitHandler.instance not found.");
            }
        }
    }
    private IEnumerator ActivateLightShield()
    {
        if (arex == null)
        {
            Debug.LogWarning("Arex not found.");
            yield break;
        }
        ArexsBehaviour arexBehaviour = arex.GetComponent<ArexsBehaviour>();
        if (arexBehaviour == null)
        {
            yield break;
        }
        GameObject shield = arexBehaviour.GetShield();
        shield.SetActive(true);

        float elapsed = 0f;
        while (elapsed < shieldDuration)
        {
            if (arex == null || shield == null) break;
            Vector3 arexPos = arex.transform.position;
            shield.transform.position = arexPos;
            KillDarkBallsNearShield(arexPos);
            elapsed += Time.deltaTime;
            yield return null;
        }
        shield.SetActive(false);
    }
    private IEnumerator SpawnLightBalls()
    {
        Vector3 origin = arex.transform.position;
        for (int i = 0; i < lightBallCount; i++)
```

```
        {
            float angle = i * (360f / lightBallCount);
            Vector3 dir = new Vector3(Mathf.Cos(angle * Mathf.Deg2Rad), Mathf.Sin(angle
* Mathf.Deg2Rad), 0).normalized;
            GameObject ball = Instantiate(lightBallPrefab, origin,
Quaternion.identity);
            LightBallProjectile projectile = ball.GetComponent<LightBallProjectile>();
            if (projectile != null)
                projectile.Init(dir, lightBallSpeed, lightBallLifetime);
        }
        yield return null;
    }
    private void KillDarkBallsNearShield(Vector3 center)
    {
        DarkBallController[] allBalls = FindObjectsOfType<DarkBallController>();

        foreach (DarkBallController dk in allBalls)
        {
            if (dk.gameObject.name.Contains(darkBallPrefab.name))
            {
                if (Vector3.Distance(dk.transform.position, center) <=
shieldKillRadius)
                {
                    Destroy(dk.gameObject);
                }
            }
        }
    }
    private IEnumerator ActivatePlayerShield()
    {
        GameObject player = GameObject.FindGameObjectWithTag("Player");
        if (player == null)
        {
            Debug.LogWarning("Dayligher not found!");
            yield break;
        }

        Transform playerTransform = player.transform;
        Transform existingShield = playerTransform.Find("PlayerLightShield");
        GameObject shield;
        if (existingShield != null)
        {
            shield = existingShield.gameObject;
        }
        else
        {
            shield = Instantiate(arex.GetComponent<ArexsBehaviour>().lightShieldPrefab,
playerTransform.position, Quaternion.identity, playerTransform);
            shield.name = "PlayerLightShield";
            shield.transform.localScale *= 1.2f;
        }
        shield.SetActive(true);
        float elapsed = 0f;
        while (elapsed < shieldDuration)
        {
            if (player == null || shield == null) break;
            shield.transform.position = playerTransform.position;
            KillDarkBallsNearShield(playerTransform.position);
            DarkBallController[] allBalls = FindObjectsOfType<DarkBallController>();
            foreach (DarkBallController ball in allBalls)
            {
```

```
                if (Vector3.Distance(ball.transform.position, playerTransform.position)
<= shieldKillRadius)
                {
                    shield.SetActive(false);
                    yield break;
                }
            }
            elapsed += Time.deltaTime;
            yield return null;
        }
        shield.SetActive(false);
    }
}
```