

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

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка 3D відеогри на Unity з елементами хорору
та RPG»

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

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

_____ Даніл ЯДЛОСЬ
(підпис)

Виконав: здобувач вищої освіти групи ПД-41

_____ Даніл ЯДЛОСЬ

Керівник: _____ Микита ТРЕНЬОВ
асист.

Рецензент: _____

Київ 2026

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ**
Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

Інженерії програмного забезпечення

_____ Ірина ЗАМРІЙ

« ____ » _____ 2026 р.

**ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ**

Ядлось Данілу Олександровичу

1. Тема кваліфікаційної роботи: «Розробка 3D відеогри на Unity з елементами хорору та RPG»

керівник кваліфікаційної роботи асистент кафедри ІІЗ Микита ТРЕНЬОВ,
затверджені наказом Державного університету інформаційно-комунікаційних
технологій від «20» лютого 2026 року № 51.

2. Строк подання кваліфікаційної роботи «29» травня 2026 року.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про розробку 3D відеоігор з елементами хорору та RPG; матеріали щодо оцінювання ігрового досвіду, залучення та занурення гравця; результати аналізу існуючих ігор-аналогів; вимоги до програмного продукту; технічна документація Unity, C#, Unity Scripting API, HDRP, Git/GitHub та засобів, використаних для реалізації гри.

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

1. Огляд та аналіз підходів до розробки 3D відеоігор з елементами хорору та RPG.

2. Аналіз ігор-аналогів та визначення вимог до ігрового застосунку.

3. Проектування архітектури та основних функціональних модулів гри.
4. Програмна реалізація ігрового застосунку з використанням Unity, C# та засобів роботи з 3D-середовищем.
5. Реалізація основних ігрових механік, інтерфейсу користувача та системи збереження прогресу.
6. Тестування функціональності та продуктивності ігрового застосунку.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Концепт гри.
3. Вимоги до програмного забезпечення.
4. Програмні засоби реалізації.
5. Ігровий цикл.
6. Діаграма варіантів використання.
7. Діаграма класів.
8. Екранні форми.
9. Апробація результатів дослідження.

6. Дата видачі завдання «23» лютого 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	23.02-28.02.2026	
2	Аналіз та дослідження існуючих аналогів	01.03-07.03.2026	
3	Огляд підходів до розробки 3D відеоігор, формування ігрового досвіду, залучення та занурення гравця	08.03–20.03.2026	
4	Проектування ігрового застосунку SITE-108	21.03–19.04.2026	
5	Програмна реалізація ігрового застосунку в Unity	20.04–01.05.2026	
6	Тестування застосунку	02.05–10.05.2026	
7	Оформлення роботи: вступ, висновки, реферат	11.05–17.05.2026	
8	Розробка демонстраційних матеріалів	18.05–11.05.2026	
9	Попередній захист роботи	12.05-22.05.2026	

Здобувач вищої освіти

(підпис)

Даніл ЯДЛОСЬ

Керівник
кваліфікаційної роботи

(підпис)

Микита ТРЕНЬОВ

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: _69_ стор., _13_ табл., _20_ рис., _21_ джерело.

Мета роботи – підвищення рівня залучення та занурення гравця в ігровий процес за рахунок розробки програмного продукту в середовищі Unity з використанням механік, характерних для жанрів хорору та RPG.

Об'єкт дослідження – процеси проєктування та розробки 3D відеогри з елементами хорору та RPG.

Предмет дослідження – засоби, методи та технології реалізації ігрових механік, атмосферних рішень і системи прогресії в середовищі Unity.

Короткий зміст роботи: В роботі проаналізовано підходи до розробки 3D відеоігор з елементами хорору та RPG, а також особливості формування ігрового досвіду, залучення та занурення гравця. Розглянуто існуючі ігри-аналоги, визначено їх переваги, недоліки та рішення, корисні для проєктування власного ігрового застосунку. Спроектовано архітектуру гри, визначено функціональні модулі та реалізовано ключові можливості: керування персонажем, взаємодію з об'єктами ігрового середовища, інвентар, інтерфейс користувача та збереження ігрового прогресу. Окрему увагу приділено побудові атмосферного 3D-середовища, що підтримує напругу, дослідження простору та поступову прогресію гравця. Проведено тестування функціональності, стабільності та продуктивності гри. Для реалізації використано Unity, C#, Unity Scripting API, HDRP, Git/GitHub та asset-ресурси.

Сферою використання застосунку є індустрія цифрових розваг, зокрема створення сюжетно-орієнтованих 3D відеоігор для користувачів, зацікавлених в атмосферному проходженні, дослідженні ігрового середовища, хорор-елементах та RPG-прогресії.

КЛЮЧОВІ СЛОВА: 3D ВІДЕОГРА, UNITY, C#, ХОРОР, RPG, ІГРОВИЙ ДОСВІД, ЗАНУРЕННЯ, ІГРОВІ МЕХАНІКИ.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	9
ВСТУП.....	10
1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ РОЗРОБКИ 3D ВІДЕОІГОР З ЕЛЕМЕНТАМИ ХОРОРУ ТА RPG.....	12
1.1 Характеристика предметної галузі розробки 3D відеоігор.....	12
1.2 Особливості формування ігрового досвіду, залучення та занурення гравця.....	14
1.3 Аналіз існуючих ігор-аналогів.....	15
1.4 Обґрунтування актуальності та практичної новизни розробки.	18
1.5 Огляд засобів реалізації 3D відеогри.....	19
1.6 Визначення об'єкта, предмета, мети та задач роботи.....	21
2 ПРОЄКТУВАННЯ ІГРОВОГО ЗАСТОСУНКУ.....	23
2.1 Визначення функціональних та нефункціональних вимог.....	23
2.2 Діаграма варіантів використання.....	27
2.3 Проєктування архітектури програмного продукту.....	28
2.4 Проєктування основних функціональних модулів гри.....	30
2.5 Проєктування ігрового циклу та структури рівня.....	32
2.6 Проєктування інтерфейсу користувача.....	35
3 ПРОГРАМНА РЕАЛІЗАЦІЯ 3D ВІДЕОГРИ.....	40
3.1 Організація Unity-проєкту.....	40
3.2 Діаграма класів та специфікація ключових програмних модулів.....	43
3.3 Реалізація керування персонажем і системи взаємодії.....	47
3.4 Реалізація інвентарю, доступів та ігрової прогресії.....	50
3.5 Реалізація технічних систем, несправностей і моніторингу.....	54
3.6 Реалізація системи збереження ігрового прогресу.....	56
3.7 Реалізація інтерфейсу користувача.....	59
3.8 Формування ігрового середовища.....	60

4 ТЕСТУВАННЯ ІГРОВОГО ЗАСТОСУНКУ.....	63
4.1 Обґрунтування вибору видів тестування.....	63
4.2 Функціональне тестування основних ігрових механік.....	64
4.3 Тестування сценаріїв проходження та взаємодії з інтерфейсом	65
4.4 Тестування системи збереження ігрового прогресу.....	67
4.5 Тестування стабільності та продуктивності.....	68
4.6 Тест-кейси та результати їх виконання.....	70
4.7 Оцінка відповідності програмного продукту поставленим	72
вимогам.....	
ВИСНОВКИ.....	76
ПЕРЕЛІК ПОСИЛАНЬ.....	79
ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ.....	81
ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ.....	88

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

3D – Three-Dimensional

API – Application Programming Interface

FPS – Frames Per Second

GDD – Game Design Document

HDRP – High Definition Render Pipeline

JSON – JavaScript Object Notation

RPG – Role-Playing Game

UI – User Interface

UML – Unified Modeling Language

UX – User Experience

VS Code – Visual Studio Code

ВСТУП

Актуальність: індустрія відеоігор залишається однією з найбільш динамічних сфер цифрових розваг, а попит на ігрові продукти з високим рівнем занурення та емоційної напруги продовжує зростати [1; 2]. Для 3D хорор-ігор особливо важливими є атмосфера, напруга, дослідження простору, якість взаємодії з ігровим середовищем і стабільність роботи програмного продукту. Дослідження хорор-ігор показують, що страх у таких іграх формується не лише через візуальні або сюжетні елементи, а й через взаємодію гравця з правилами, простором, загрозами та станом ігрового світу [3].

Поєднання елементів хорору та RPG є актуальним, оскільки дозволяє не тільки створювати короткочасний ефект страху, а й підтримувати довготривалу залученість гравця через прогресію, відкриття нових можливостей, дослідження середовища та взаємодію з об'єктами. У такому підході хорор-складова відповідає за атмосферу, напругу та відчуття небезпеки, а RPG-елементи підтримують мотивацію до проходження і поступове поглиблення взаємодії з ігровим світом. Саме тому розробка 3D відеогри на Unity з елементами хорору та RPG є актуальною задачею в галузі інженерії програмного забезпечення.

Об'єкт дослідження: процеси проєктування та розробки 3D відеогри з елементами хорору та RPG.

Предмет дослідження: засоби, методи та технології реалізації ігрових механік, атмосферних рішень і системи прогресії в середовищі Unity.

Мета роботи: підвищення рівня залучення та занурення гравця в ігровий процес за рахунок розробки програмного продукту в середовищі Unity з використанням механік, характерних для жанрів хорору та RPG.

Для досягнення поставленої мети в роботі передбачено аналіз предметної галузі розробки 3D відеоігор з елементами хорору та RPG, дослідження підходів до формування ігрового досвіду, залучення та занурення гравця, аналіз існуючих ігор-аналогів, визначення їх переваг і недоліків, обґрунтування вибору засобів

реалізації, формування функціональних і нефункціональних вимог, проєктування архітектури програмного продукту, основних функціональних модулів, інтерфейсу користувача, ігрового циклу та структури рівня, а також реалізацію і тестування основних механік ігрового застосунку.

Методи дослідження: аналіз наукових, технічних та інформаційних джерел з тематики розробки відеоігор; порівняльний аналіз ігор-аналогів; системний аналіз для визначення вимог до програмного продукту; методи об'єктно-орієнтованого проєктування та програмування; UML-моделювання для опису архітектури й основних компонентів системи; прототипування в середовищі Unity; функціональне тестування, тестування інтерфейсу користувача, стабільності та продуктивності ігрового застосунку.

Для реалізації програмного продукту було обрано ігровий рушій Unity, мову програмування C#, Unity Scripting API для реалізації ігрової логіки, HDRP для налаштування візуальної складової 3D-середовища, Visual Studio Code як середовище роботи з кодом, а також Git і GitHub для контролю версій та організації розробки. Для формування ігрового середовища використано asset-ресурси, які адаптовано відповідно до потреб проєкту.

Практична новизна роботи полягає у поєднанні в одному 3D ігровому застосунку RPG-механік, спрямованих на підтримку довготривалої залученості гравця та асоціації з персонажем, із механіками атмосферного хорору, що формують напругу, невизначеність і відчуття небезпеки. Таке поєднання реалізується через дослідження ігрового середовища, поступову прогресію, взаємодію з об'єктами, обмежений доступ до інформації та збереження ігрового прогресу.

Практична значущість результатів полягає у створенні прототипу 3D відеогри, який може бути використаний як основа для подальшої розробки сюжетно-орієнтованого ігрового продукту з елементами хорору та RPG. Реалізовані програмні модулі можуть бути розширені новими рівнями, механіками, подіями, інтерфейсними елементами та засобами взаємодії з ігровим середовищем.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ РОЗРОБКИ 3D ВІДЕОІГОР З ЕЛЕМЕНТАМИ ХОРОРУ ТА RPG

1.1 Характеристика предметної галузі розробки 3D відеоігор

Предметна галузь даної роботи охоплює розробку інтерактивних 3D відеоігор, які поєднують програмну логіку, візуальне середовище, користувацький інтерфейс, систему взаємодії з об'єктами та засоби формування ігрового досвіду. На відміну від прикладних інформаційних систем, відеогра орієнтована не лише на виконання конкретної утилітарної задачі, а й на створення цілісного інтерактивного середовища, у якому користувач керує персонажем, досліджує простір, реагує на події та поступово засвоює правила ігрового світу.

3D відеогра як програмний продукт складається з багатьох взаємопов'язаних компонентів. До них належать ігрові сцени, тривимірні моделі, матеріали, освітлення, камери, звукові ефекти, анімації, фізичні взаємодії, інтерфейс користувача, скрипти ігрової логіки, системи збереження прогресу та механізми керування станом ігрових об'єктів. У процесі розробки важливо забезпечити не тільки наявність цих елементів, а й їх узгоджену роботу в межах єдиного ігрового циклу.

Особливістю 3D відеоігор є активна роль гравця у формуванні перебігу взаємодії з програмним продуктом. Користувач не лише отримує інформацію від системи, а постійно виконує дії: переміщується у просторі, взаємодіє з предметами, приймає рішення, використовує ресурси, реагує на зміну стану середовища та отримує зворотний зв'язок від гри. Тому якість такої системи визначається не тільки коректністю роботи окремих функцій, а й тим, наскільки зрозумілою, стабільною та логічною є взаємодія користувача з ігровим світом.

Важливою складовою предметної галузі є поняття ігрового досвіду. У межах моделей дослідження відеоігор якість ігрового досвіду пов'язується з рівнем концентрації, відчуттям контролю, складністю завдань, зворотним

зв'язком, зануренням та емоційною реакцією гравця [4]. З позиції MDA-підходу гра може розглядатися через взаємозв'язок механік, динаміки та естетики: механіки визначають правила і можливості дії, динаміка формується під час взаємодії гравця із системою, а естетика відображає емоційне враження від проходження [6]. Це означає, що під час розробки 3D відеогри недостатньо реалізувати набір окремих функцій; необхідно забезпечити їхню взаємодію таким чином, щоб вони підтримували очікуваний ігровий досвід.

Розробка 3D відеогри також має низку технічних особливостей. До них належать побудова ігрового простору, налаштування фізики та колізій, реалізація керування персонажем, організація взаємодії з об'єктами, створення інтерфейсу користувача, налаштування освітлення, оптимізація продуктивності та забезпечення стабільності роботи програми. Окрему роль відіграє структура проекту, оскільки ігровий застосунок зазвичай містить велику кількість об'єктів, ресурсів, скриптів і сцен, які мають бути логічно організовані для подальшої підтримки та розширення.

Для 3D відеоігор важливою є продуктивність, оскільки затримки, нестабільна частота кадрів, некоректна робота колізій або помилки в логіці подій безпосередньо впливають на сприйняття гри користувачем. На відміну від багатьох прикладних програм, де окрема затримка може бути менш критичною, у відеогрі технічні помилки швидко порушують занурення, знижують відчуття контролю та можуть зробити проходження незручним або неможливим. Тому розробка ігрового застосунку потребує поєднання проєктування програмної архітектури, реалізації ігрових механік, роботи з візуальними ресурсами та подальшого тестування.

У межах даної кваліфікаційної роботи предметна галузь розглядається в контексті створення 3D відеогри з елементами хорору та RPG. Для такого програмного продукту особливо важливими є атмосферність, занурення, поступове відкриття можливостей, взаємодія з ігровим середовищем, система прогресії та стабільна робота механік. Саме ці особливості визначають подальший аналіз жанрових елементів, ігор-аналогів, засобів реалізації та вимог

до розроблюваного програмного продукту.

1.2 Особливості формування ігрового досвіду, залучення та занурення гравця

Ігровий досвід є однією з ключових характеристик відеогри як інтерактивного програмного продукту. На відміну від звичайного прикладного програмного забезпечення, гра оцінюється не тільки за коректністю виконання функцій, а й за тим, наскільки вона здатна підтримувати увагу користувача, створювати емоційну реакцію та мотивувати гравця продовжувати проходження. У межах даної роботи ігровий досвід розглядається як результат взаємодії гравця з ігровими механіками, середовищем, системою прогресії та подіями, що відбуваються під час проходження.

Одним із важливих аспектів ігрового досвіду є залучення гравця. У моделі GameFlow воно пов'язується з низкою складових, серед яких для одиночної 3D гри з елементами хорору та RPG особливе значення мають чіткість цілей, відчуття контролю, зворотний зв'язок і поступове зростання виклику [4]. Наявність цих елементів забезпечує усвідомлену участь гравця в ігровому процесі, дозволяє йому орієнтуватися в поточній ситуації, оцінювати наслідки власних дій і підтримувати інтерес до подальшого проходження.

Занурення гравця пов'язане з тим, наскільки цілісно сприймається ігровий світ. У дослідженні Е. Brown і Р. Cairns занурення описується як поступовий процес, що може переходити від початкового залучення до глибшого поглинання уваги гравця [5]. Для цього важливо, щоб простір, атмосфера, інтерфейс і поведінка об'єктів не суперечили одне одному. Якщо гра має незрозумілу взаємодію, нестабільну роботу або нелогічні реакції системи, це швидко руйнує відчуття присутності в ігровому світі.

З позиції MDA-підходу ігровий досвід формується через взаємозв'язок механік, динаміки та естетики [6]. Механіки визначають доступні дії гравця, динаміка виникає під час взаємодії з цими правилами, а естетика відображає

емоційний результат проходження. Для 3D відеогри це означає, що окремі функції не повинні існувати ізольовано. Рух персонажа, взаємодія з об'єктами, інвентар, доступ до нових зон і збереження прогресу мають підтримувати єдиний ігровий цикл.

У жанрі хорору залучення та занурення мають особливе значення, оскільки страх виникає не лише через появу небезпеки. Важливу роль відіграють очікування, невизначеність, обмежена інформація та атмосфера середовища. Якщо гравець не відчуває зв'язку з персонажем або не розуміє значення власних дій, хорор-елементи сприймаються поверхнево. Тому перед створенням сильного емоційного напруження гра має спочатку сформувати зацікавленість і дати гравцю причину залишатися в її світі.

RPG-елементи можуть підтримувати таку зацікавленість за рахунок поступової прогресії. Відкриття нових можливостей, доступ до раніше недоступних зон, накопичення предметів або інформації створюють відчуття розвитку та просування. У поєднанні з хорор-складовою це дозволяє не лише лякати гравця окремими подіями, а й поступово посилювати його особисту залученість у проходження.

Отже, формування ігрового досвіду в 3D відеогрі з елементами хорору та RPG потребує узгодженого поєднання трьох ґрунтовних складових: зрозумілої взаємодії, атмосферного занурення та прогресії. Саме ці складові визначають подальші вимоги до проєктування ігрового циклу, структури рівня, механік взаємодії та програмної реалізації ігрового застосунку.

1.3 Аналіз існуючих ігор-аналогів

Для аналізу існуючих ігрових продуктів було обрано три гри, які демонструють різні підходи до формування хорор-досвіду в 3D середовищі: Resident Evil 7: Biohazard [7], Amnesia: The Bunker [8] та Outlast [9]. Вони не є повними аналогами розроблюваної гри, однак є релевантними для дослідження, оскільки дозволяють розглянути різні способи створення напруги, вразливості

гравця, ролі середовища та поступового відкриття нових можливостей.



Рис. 1.1 Ігри-аналоги, обрані для аналізу

Resident Evil 7: Biohazard є прикладом survival horror гри, у якій напруга формується через поєднання замкненого простору, обмежених ресурсів, небезпечних зустрічей і поступового дослідження локацій. Важливою особливістю гри є баланс між вразливістю персонажа та наявністю засобів протидії: гравець має зброю, але її використання обмежується кількістю ресурсів і ситуаційною небезпекою. Для розроблюваної гри корисним є підхід до побудови напруги через простір, керування ресурсами та відкриття нових зон, однак повноцінна бойова модель не є основною для обраної концепції.

Amnesia: The Bunker демонструє більш системний підхід до хорору. У цій грі небезпека пов'язана не лише з присутністю монстра, а й зі станом середовища: освітленням, роботою генератора, шумом, доступними ресурсами та маршрутами переміщення. Такий підхід є особливо важливим для даної роботи, оскільки він показує, як технічні системи всередині ігрового світу можуть безпосередньо впливати на поведінку гравця та рівень напруги. Для розроблюваної гри доцільно врахувати саме цю роль середовища як активного елемента ігрового процесу, але уникати надмірного тиску на гравця, щоб залишити простір для дослідження.

Outlast обрано як приклад хорор-гри, у якій основний акцент зроблено на

беззахисності персонажа. Гравець не має повноцінної можливості протидіяти ворогам і змушений покладатися на втечу, ховання, спостереження та обережне переміщення. Така модель добре демонструє, як обмеження контролю може підсилювати страх і занурення. Для розроблюваної гри цей підхід може бути використаний частково: гравець має отримувати інструменти для взаємодії із середовищем і технічними системами, але не повинен мати повного контролю над джерелом небезпеки.

Для узагальнення результатів аналізу було сформовано порівняльну таблицю, у якій розглянуто ключові характеристики ігор-аналогів та їх значення для розроблюваного програмного продукту.

Таблиця 1.1

Порівняльний аналіз ігор-аналогів

Характеристика	Resident Evil 7: Biohazard	Amnesia: The Bunker	Outlast	Розроблювана гра (SITE-108)
Джерело напруги	Атмосфера, ресурси, вороги	Стан генератора, світло, монстр	Переслідування, темрява, камера	Технічні збої, аномалії, обмежена інформація
Ступінь вразливості	Помірний: є зброя	Високий: захист обмежений	Дуже високий: переважно втеча	Високий: контроль гравця обмежений
Роль середовища	Головоломки, ресурси	Система виживання	Втеча й ховання	Ремонт, доступ, моніторинг
Прогресія	Ключі, нові зони	Інструменти, маршрути	Лінійне проходження	Рівні, системи, інструменти

Аналіз ігор-аналогів показав, що кожен із розглянутих проєктів має власний підхід до формування хорор-досвіду. У Resident Evil 7: Biohazard значну роль відіграють атмосфера замкненого простору, ресурси та небезпечні зіткнення; в Amnesia: The Bunker напруга посилюється через залежність гравця від стану середовища й технічних систем; в Outlast основний акцент зроблено на вразливості персонажа, втечі та спостереженні. Водночас ці підходи переважно зосереджені на окремих джерелах напруги, тоді як для розроблюваної гри доцільним є їх комбінування в межах єдиного ігрового циклу.

Таким чином, результати порівняння підтверджують доцільність

створення 3D відеогри, у якій напруга формується через поєднання атмосферного середовища, технічних несправностей, обмеженого доступу до інформації, дослідження простору та поступового відкриття нових можливостей. Такий підхід дозволяє перейти від аналізу існуючих рішень до формування вимог, архітектури та функціональних модулів ігрового застосунку.

1.4 Обґрунтування актуальності та практичної новизни розробки

Актуальність розробки 3D відеогри з елементами хорору та RPG зумовлена потребою у створенні ігрових продуктів, які поєднують емоційне напруження, дослідження середовища та довготривалу залученість гравця. Для хорор-ігор важливими є атмосфера, відчуття небезпеки, невизначеність і обмеженість контролю. Водночас RPG-елементи дозволяють підтримувати інтерес до проходження через прогресію, відкриття нових можливостей і поступове посилення зв'язку між гравцем та персонажем.

Проведений аналіз ігор-аналогів показав, що існуючі рішення по-різному формують хорор-досвід. Одні ігри роблять акцент на атмосфері, ресурсах і небезпечних зіткненнях, інші — на вразливості персонажа, втечі або залежності від стану ігрового середовища. Такі підходи є ефективними, однак кожен із них переважно зосереджується на окремому джерелі напруги. Тому доцільним є створення гри, у якій напруга формується комплексно: через атмосферу, дослідження простору, обмежений доступ до інформації, технічні події та поступове відкриття нових можливостей.

Практична новизна роботи полягає у поєднанні в одному 3D ігровому застосунку RPG-механік, спрямованих на підтримку довготривалої залученості гравця та асоціації з персонажем, із механіками атмосферного хорору, що формують напругу, невизначеність і відчуття небезпеки. Таке поєднання дозволяє не обмежувати хорор-досвід лише окремими лякаючими подіями, а підтримувати його через саму структуру проходження, взаємодію з середовищем і поступовий розвиток ігрової ситуації.

Особливістю розроблюваної гри є те, що ігрове середовище виступає не лише фоном для подій, а активною частиною ігрового процесу. Напруга підтримується через технічні несправності, необхідність взаємодії з об'єктами, обмеженість інформації та залежність гравця від стану систем усередині ігрового світу. Це дозволяє поєднати дослідження, прогресію та атмосферний хорор у межах єдиного ігрового циклу.

Таким чином, актуальність і практична новизна розробки визначаються поєднанням хорор-атмосфери, RPG-прогресії, дослідження 3D-середовища та технічної взаємодії з об'єктами. Саме така комбінація формує основу для подальшого визначення вимог, вибору засобів реалізації та проєктування ігрового застосунку.

1.5 Огляд засобів реалізації 3D відеогри

Розробка 3D відеогри потребує узгодженого технологічного стеку, який забезпечує повний цикл створення і підтримки ігрового застосунку. У межах даної роботи засоби реалізації добиралися з урахуванням специфіки проєкту: необхідності створити інтерактивне 3D-середовище, підтримати атмосферну візуальну складову та забезпечити стабільну роботу ігрової логіки.

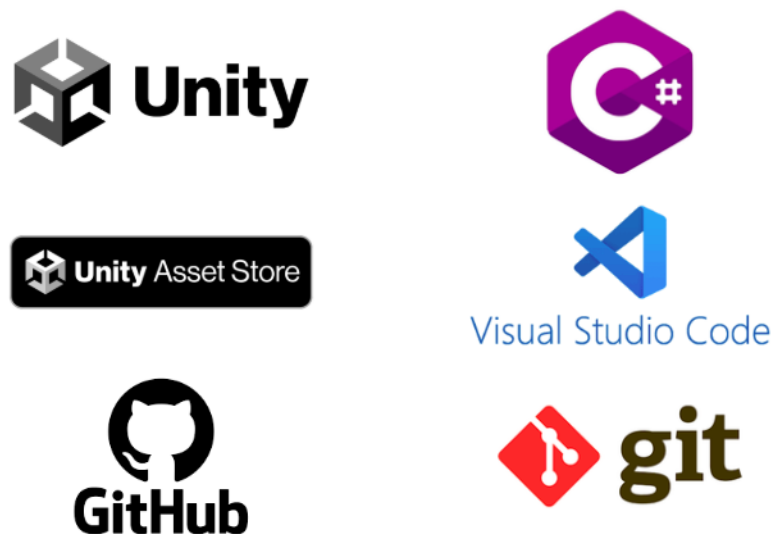


Рис. 1.2 Програмні засоби реалізації 3D відеогри

Основним середовищем розробки обрано Unity. Цей рушій має увесь базовий інструментарій для створення 2D і 3D ігор, а також підтримує компонентний підхід до побудови ігрових об'єктів. У межах такого підходу поведінка об'єкта визначається набором компонентів, прикріплених до GameObject. Це є зручним для розробки інтерактивного 3D-середовища, оскільки окремі об'єкти сцени можуть отримувати власну логіку взаємодії, відображення, обробки подій або зміни стану [10].

Для реалізації ігрової логіки використовується мова програмування C#. Вона застосовується в Unity для створення скриптів, що керують поведінкою персонажа, об'єктів, інтерфейсу, ігрових подій та інших систем. Використання C# є доцільним для даної роботи, оскільки мова підтримує об'єктно-орієнтований підхід, дозволяє структурувати логіку у вигляді класів і методів та забезпечує достатню гнучкість для реалізації взаємодії між окремими компонентами гри [11].

Для налаштування візуальної складової 3D-середовища використовується High Definition Render Pipeline. HDRP є конвеєром рендерингу Unity, орієнтованим на створення високоякісної графіки для продуктивних платформ. Для хорор-гри це має високу значимість, оскільки освітлення, тіні, матеріали, постобробка та загальна візуальна атмосфера безпосередньо впливають на сприйняття ігрового простору. Використання HDRP дозволяє точніше працювати з освітленням і візуальними ефектами, що є важливим для формування напруги та занурення гравця [13].

Для роботи з програмним кодом використовується Visual Studio Code. Це середовище забезпечує редагування C#-скриптів, роботу з розширеннями, навігацію по файлах проєкту та підтримку типових інструментів розробника. У межах даної роботи VS Code використовується як редактор коду, тоді як основне налаштування сцен, об'єктів, компонентів і ресурсів виконується безпосередньо в Unity [14].

Для контролю версій та організації роботи над проєктом використовуються Git і GitHub. Git дозволяє фіксувати зміни у файлах проєкту,

повертатися до попередніх версій і контролювати історію розробки. GitHub використовується для зберігання репозиторію, синхронізації змін і організації задач розробки. Для ігрового проєкту це важливо, оскільки Unity-проєкт містить значну кількість файлів: скрипти, сцени, налаштування, матеріали, префаби та інші ресурси [15; 16].

Окрему роль у розробці відеогри відіграють asset-ресурси. До них належать 3D-моделі, матеріали, текстури, звукові ефекти, елементи інтерфейсу та інші компоненти, що використовуються для формування ігрового середовища. У межах даної роботи asset-ресурси застосовуються як основа для візуального наповнення сцени, однак не замінюють програмну реалізацію гри. Ключові механіки, логіка взаємодії, структура ігрового циклу, робота з інтерфейсом і збереженням прогресу мають бути налаштовані та інтегровані відповідно до задач розроблюваного програмного продукту [17].

1.6 Визначення об'єкта, предмета, мети та задач роботи

На основі проведеного аналізу предметної галузі, особливостей формування ігрового досвіду, ігор-аналогів та засобів реалізації було визначено об'єкт, предмет, мету та задачі кваліфікаційної роботи.

Об'єкт дослідження: процеси проєктування та розробки 3D відеогри з елементами хорору та RPG.

Предмет дослідження: засоби, методи та технології реалізації ігрових механік, атмосферних рішень і системи прогресії в середовищі Unity.

Мета роботи: підвищення рівня залучення та занурення гравця в ігровий процес за рахунок розробки програмного продукту в середовищі Unity з використанням механік, характерних для жанрів хорору та RPG.

Для досягнення поставленої мети необхідно вирішити такі задачі:

1. Проаналізувати предметну галузь розробки 3D відеоігор з елементами хорору та RPG.
2. Дослідити особливості формування ігрового досвіду, залучення та

занурення гравця.

3. Провести аналіз існуючих ігор-аналогів та визначити їх переваги, недоліки й рішення, корисні для розробки власного ігрового застосунку.
4. Обґрунтувати актуальність і практичну новизну розробки.
5. Провести огляд засобів реалізації 3D відеогри та обґрунтувати вибір технологічного стеку.
6. Сформулювати функціональні та нефункціональні вимоги до ігрового застосунку.
7. Спроекувати архітектуру програмного продукту, основні функціональні модулі, інтерфейс користувача, ігровий цикл та структуру рівня.
8. Реалізувати основні ігрові механіки, систему взаємодії з об'єктами, інтерфейс користувача та збереження ігрового прогресу.
9. Провести тестування функціональності, стабільності, продуктивності та відповідності програмного продукту поставленим вимогам.

Визначені мета, об'єкт, предмет і задачі роботи формують основу для подальшого проєктування структури ігрового застосунку, реалізації його механік та перевірки відповідності готового програмного продукту поставленим вимогам.

2 ПРОЄКТУВАННЯ ІГРОВОГО ЗАСТОСУНКУ

У цьому розділі виконується проєктування ігрового застосунку SITE-108 як 3D програмного продукту в середовищі Unity. На основі сформованої в першому розділі концепції та визначених задач роботи уточнюються вимоги до гри, сценарії взаємодії користувача з системою, архітектура програмного продукту, склад функціональних модулів, логіка ігрового циклу та побудова інтерфейсу користувача. Проєктні рішення орієнтовані на подальшу реалізацію в розділі 3 і тому формулюються на такому рівні деталізації, який безпосередньо відповідає майбутнім програмним компонентам.

2.1 Визначення функціональних та нефункціональних вимог

Проєктування програмного продукту доцільно починати з визначення вимог. Відповідно до підходів інженерії вимог, функціональні вимоги визначають, що саме має робити система, а нефункціональні — якими властивостями вона має володіти під час виконання цих функцій, зокрема з погляду продуктивності, зручності використання, надійності та супровідності [18; 19]. Для SITE-108 це особливо важливо, оскільки в хорор-грі технічні затримки, перевантаження інтерфейсу або нестабільне відтворення подій безпосередньо руйнують занурення гравця [4; 5].

Функціональні вимоги для SITE-108 фіксують той необхідний набір можливостей, без якого ігровий цикл не зможе працювати як єдина система.

Таблиця 2.1

Функціональні вимоги до ігрового застосунку SITE-108

Позначення	Вимога	Опис
FR-01	Керування персонажем	Гра повинна забезпечувати переміщення персонажа у 3D-середовищі, огляд простору від першої особи та базові дії користувача.
FR-02	Взаємодія з об'єктами	Гравець повинен мати можливість взаємодіяти з інтерактивними об'єктами: дверима, предметами, технічними елементами, панелями та іншими об'єктами сцени.
FR-03	Система інвентарю	Гра повинна підтримувати збирання, збереження та використання предметів, необхідних для проходження.
FR-04	Система доступів	Гравець повинен мати можливість відкривати нові зони за допомогою карток доступу, відновлення технічних систем або виконання певних умов.
FR-05	Технічні несправності	У грі мають бути передбачені події, пов'язані зі збоями технічних систем, що створюють додаткові задачі для гравця.
FR-06	Ремонт і відновлення систем	Гравець повинен мати можливість відновлювати окремі системи середовища, наприклад камери або електроживлення дверей, використовуючи знайдені ресурси.
FR-07	Прогресія через середовище	Прогресія має реалізовуватися через відкриття нових зон, відновлення систем, отримання інструментів, ресурсів та інформації, а не через пряме посилення характеристик героя.

Продовження таблиці 2.1

Функціональні вимоги до ігрового застосунку SITE-108

FR-08	Інтерфейс користувача	Гра повинна надавати користувачу зрозумілі підказки взаємодії, елементи інвентарю, меню паузи та інші необхідні екранні форми.
FR-09	Збереження прогресу	Гра повинна забезпечувати збереження та завантаження ігрового прогресу, включаючи стан персонажа, інвентарю та важливих об'єктів середовища.
FR-10	Атмосферні події	Гра повинна підтримувати події, що формують напругу: зміни освітлення, звукові сигнали, повідомлення системи, візуальні ефекти та інші елементи хорор-атмосфери.

Нефункціональні вимоги визначають якісні характеристики програмного продукту. Для 3D гри з елементами хорору вони мають особливе значення, оскільки технічна нестабільність, низька продуктивність або незручний інтерфейс безпосередньо впливають на рівень занурення гравця. Якісні характеристики програмного продукту можуть розглядатися через такі властивості, як функціональна придатність, продуктивність, надійність, зручність використання, сумісність, супроводжуваність та інші характеристики [19].

Нефункціональні вимоги до SITE-108 наведено в таблиці 2.2.

Таблиця 2.2

Нефункціональні вимоги до ігрового застосунку

Позначення	Вимога	Опис
NFR-01	Стабільність роботи	Гра не повинна завершуватися аварійно під час базових сценаріїв проходження, взаємодії з об'єктами, відкриття меню та збереження прогресу.
NFR-02	Продуктивність	Ігровий процес має залишатися плавним у межах цільової платформи, без критичних просідань частоти кадрів під час пересування рівнем та взаємодії з об'єктами.
NFR-03	Зручність керування	Керування персонажем і взаємодія з об'єктами мають бути зрозумілими для користувача без потреби у тривалому навчанні.
NFR-04	Читабельність інтерфейсу	UI-елементи мають бути помітними, але не повинні перекривати важливі елементи сцени або руйнувати атмосферу гри.
NFR-05	Атмосферність	Візуальні й звукові рішення повинні підтримувати напругу, ізоляцію та відчуття небезпеки.
NFR-06	Масштабованість	Архітектура гри має дозволяти подальше додавання нових інтерактивних об'єктів, технічних систем, предметів і подій без повної перебудови проєкту.
NFR-07	Супроводжуваність	Структура проєкту, скриптів і сцен має бути організована так, щоб полегшити подальшу підтримку та внесення змін.

Продовження таблиці 2.2

NFR-08	Надійність збережень	Дані прогресу мають зберігатися коректно та відновлюватися без втрати ключових параметрів проходження.
NFR-09	Сумісність з Unity-проектом	Реалізація має відповідати компонентному підходу Unity та бути узгодженою з використаними засобами розробки [10; 12].

2.2 Діаграма варіантів використання

Діаграма варіантів використання застосовується для формалізації основних сценаріїв взаємодії користувача з програмним продуктом. У межах SITE-108 головним актором є гравець, який взаємодіє з ігровою системою через керування персонажем, дослідження середовища, використання предметів, роботу з технічними системами та збереження прогресу.

На рисунку 2.1 наведено діаграму варіантів використання ігрового застосунку SITE-108.

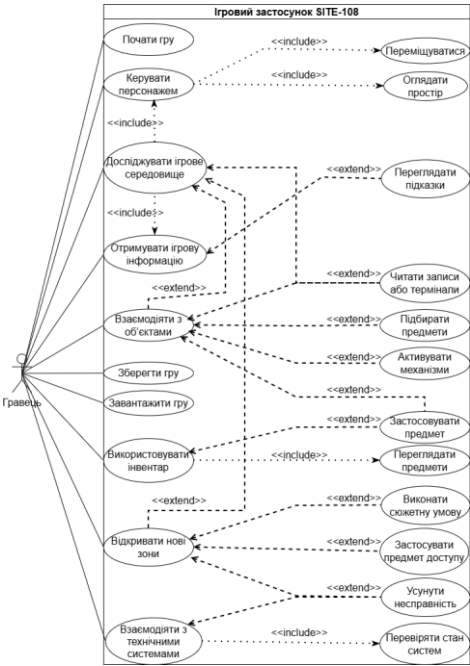


Рис. 2.1 Діаграма варіантів використання ігрового застосунку

Актором на діаграмі є гравець, який взаємодіє з ігровим застосунком через керування персонажем, дослідження середовища, взаємодію з об'єктами, використання інвентарю та роботу з технічними системами. Ці варіанти використання відображають базову логіку проходження SITE-108, де користувач поступово вивчає простір, знаходить предмети, відкриває нові зони та змінює стан ігрового середовища.

Окрему роль у діаграмі мають сценарії, пов'язані з технічними системами об'єкта. Гравець може стикатися з несправними камерами, заблокованими дверима, відсутністю електроживлення або іншими обмеженнями, які потребують пошуку ресурсів чи виконання певних дій. Саме через такі сценарії реалізується прогресія: можливості гравця розширюються не за рахунок посилення персонажа, а через відновлення систем середовища та відкриття нових маршрутів.

Допоміжні варіанти використання, зокрема перегляд інтерфейсних повідомлень, збереження, завантаження гри, зміна налаштувань і завершення сеансу, забезпечують технічну зручність користування застосунком. Вони не формують основний ігровий цикл, але підтримують стабільність, керованість і цілісність проходження.

2.3 Проєктування архітектури програмного продукту

Архітектура ігрового застосунку SITE-108 проєктується відповідно до компонентного підходу Unity. У такому підході ігрові об'єкти створюються як GameObject, а їхня поведінка визначається набором компонентів, прикріплених до цих об'єктів [10]. Програмна логіка реалізується за допомогою C#-скриптів, які взаємодіють з об'єктами сцени, інтерфейсом, подіями та внутрішнім станом гри через засоби Unity Scripting API [12].

Для SITE-108 доцільною є модульна архітектура, у якій окремі частини гри відповідають за власні зони відповідальності: керування гравцем, взаємодію з

об'єктами, інвентар, технічні системи, інтерфейс користувача, збереження прогресу та атмосферні події. Такий підхід дозволяє зменшити зв'язність між частинами системи та спростити подальше розширення проєкту.

Загальну структуру архітектури ігрового застосунку наведено в таблиці 2.3.

Таблиця 2.3

Архітектура ігрового застосунку SITE-108

Рівень архітектури	Призначення	Основні елементи
Рівень користувачького введення	Обробка команд користувача та передача їх до відповідних ігрових систем	Переміщення, огляд, взаємодія, відкриття інвентарю, меню паузи
Рівень керування персонажем	Забезпечення фізичної присутності персонажа в 3D-середовищі	Контролер руху, камера, стани переміщення, обробка базових дій
Рівень взаємодії з об'єктами	Виявлення інтерактивних об'єктів і запуск відповідних дій	Двері, предмети, панелі, перемикачі, технічні елементи сцени
Рівень ігрових систем	Реалізація логіки проходження	Інвентар, доступи, технічні несправності, ремонт, прогресія
Рівень інтерфейсу користувача	Відображення інформації та підтримка взаємодії користувача з грою	Підказки, інвентар, меню паузи, меню налаштувань, повідомлення
Рівень збереження даних	Фіксація та відновлення стану гри	Позиція персонажа, інвентар, відкриті зони, стан об'єктів і систем

Продовження таблиці 2.3

Рівень візуального та звукового представлення	Формування атмосфери, напруги та стилю гри	Освітлення, матеріали, постобробка, звуки, атмосферні ефекти
---	--	--

Наведена архітектура відображає поділ ігрового застосунку на логічні рівні, кожен з яких відповідає за окрему частину функціонування гри. Основою архітектури є компонентний підхід Unity, за якого об'єкти сцени отримують потрібну поведінку через прикріплені до них компоненти та скрипти. Це дозволяє відокремити керування персонажем, взаємодію з об'єктами, інтерфейс, збереження прогресу та системи середовища.

Для SITE-108 така структура є доцільною, оскільки ігровий процес будується навколо взаємодії гравця з технічними системами об'єкта. Дії користувача передаються від рівня введення до систем керування персонажем і взаємодії, після чого можуть змінювати стан інвентарю, доступів, технічних об'єктів або ігрового прогресу. Візуальний і звуковий рівень підтримує атмосферу гри та реагує на зміни стану середовища.

Запропонована архітектура дозволяє розширювати гру без повної перебудови її структури. Нові інтерактивні об'єкти, предмети, технічні системи або атмосферні події можуть додаватися як окремі елементи, пов'язані із загальною логікою взаємодії та прогресії.

2.4 Проєктування основних функціональних модулів гри

Функціональні модулі SITE-108 визначають основні складові програмного продукту, що забезпечують роботу ігрового процесу. Кожен модуль має власну зону відповідальності, але всі вони взаємодіють між собою в межах єдиного ігрового циклу.

Структуру основних функціональних модулів гри наведено в таблиці 2.4.

Таблиця 2.4

Основні функціональні модулі гри

Модуль	Призначення	Пов'язані системи
Модуль керування гравцем	Переміщення персонажа, огляд простору, базова навігація рівнем	Введення, камера, взаємодія з об'єктами
Модуль взаємодії з об'єктами	Обробка дій з інтерактивними елементами сцени	Інвентар, доступи, технічні системи
Модуль інвентарю	Збирання, збереження та використання предметів	Доступи, ремонт, збереження прогресу
Модуль доступів і прогресії	Відкриття нових зон, маршрутів і можливостей	Інвентар, технічні системи, структура рівня
Модуль технічних систем	Робота з несправностями, ремонтом і зміною стану середовища	Доступи, інвентар, атмосферні події
Модуль інтерфейсу користувача	Відображення підказок, меню, інвентарю та повідомлень	Усі важливі ігрові системи
Модуль збереження прогресу	Збереження та відновлення стану гри	Гравець, інвентар, доступи, технічні системи
Модуль атмосферних подій	Формування напруги та хорор-складової	Освітлення, звук, технічні системи, повідомлення
Модуль організації рівня	Просторова структура гри та логіка переходу між зонами	Доступи, технічні системи, ігровий цикл

Наведені функціональні модулі формують основу ігрового процесу SITE-

108. Центральними є модулі взаємодії з об'єктами, інвентарю, доступів і технічних систем, оскільки саме через них реалізується основна прогресія гравця. Гравець не отримує прямого посилення характеристик персонажа, а поступово розширює свої можливості через відновлення систем середовища, відкриття нових зон, отримання ресурсів та доступ до додаткової інформації.

Інші модулі підтримують цей процес. Модуль керування гравцем забезпечує переміщення та огляд простору, інтерфейс користувача надає необхідні підказки й повідомлення, система збереження фіксує стан проходження, а модуль атмосферних подій підтримує напругу та хорор-складову. Завдяки такому поділу кожна система має власну функцію, але всі вони працюють у межах єдиного ігрового циклу.

Такий підхід дозволяє зберегти баланс між прогресією та вразливістю персонажа. Гравець отримує більше інструментів для взаємодії з об'єктом, однак не втрачає відчуття небезпеки, що є важливим для гри з елементами хорору.

2.5 Проєктування ігрового циклу та структури рівня

Ігровий цикл визначає повторювану послідовність дій, через яку гравець проходить основний досвід гри. Для SITE-108 ігровий цикл має поєднати дослідження середовища, напругу хорору, поступове відкриття нових можливостей та взаємодію з технічними системами об'єкта.

На рисунку 2.2 наведено схему основного ігрового циклу SITE-108.

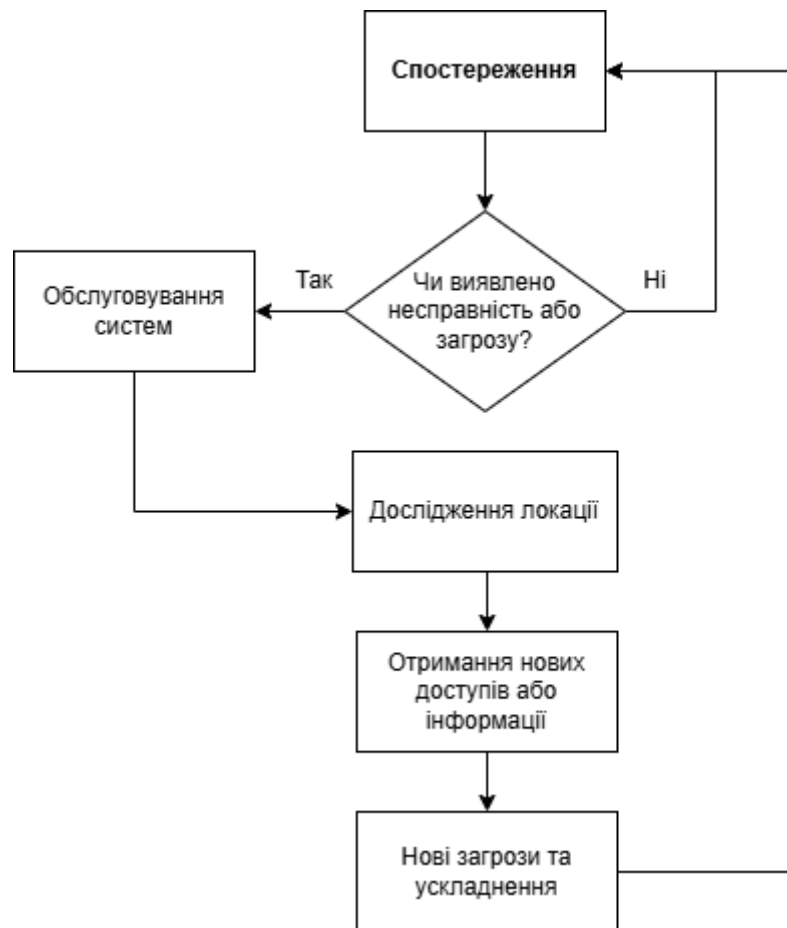


Рис. 2.2 Схема основного ігрового циклу SITE-108

Основний ігровий цикл можна подати як послідовність таких етапів: дослідження середовища, виявлення інформації, предмета, несправності або заблокованої зони, аналіз доступних дій через інвентар або технічні системи, взаємодія з об'єктом, зміна стану середовища, відкриття нового маршруту, ресурсу або інформації, посилення напруги через нові події та перехід до наступного етапу дослідження.

На початковому етапі гравець досліджує доступну частину рівня, знайомиться з простором, шукає підказки та взаємодіє з базовими об'єктами. Далі він стикається з обмеженням: зачиненими дверима, непрацюючою камерою, вимкненим живленням, недоступним маршрутом або нестачею ресурсу. Це обмеження формує локальну задачу, яку потрібно розв'язати через дослідження, пошук предмета або відновлення системи.

Після виконання задачі змінюється стан середовища. Наприклад,

відремонтована камера може відкрити новий спосіб спостереження, відновлене електроживлення — розблокувати двері, а знайдена картка доступу — дозволити перейти до нової частини рівня. У такий спосіб прогресія реалізується через середовище: гравець не отримує прямого бойового посилення, але поступово здобуває більше контролю над об'єктом.

Для хорор-гри важливо, щоб відкриття нових можливостей не знімало напругу повністю. Тому кожен новий доступ або відновлена система може супроводжуватися новими ризиками, зміною атмосфери, появою тривожних сигналів або новою інформацією, яка поглиблює сюжетну невизначеність. Це підтримує баланс між прогресом і вразливістю.

Структура рівня SITE-108 має підтримувати цей ігровий цикл. Ігровий простір доцільно будувати як взаємопов'язану систему приміщень, технічних зон, коридорів, службових кімнат і контрольних точок. Рівень не повинен бути повністю лінійним: частина зон може бути доступною відразу, частина — відкриватися після виконання умов, а частина — використовуватися повторно після зміни стану систем.

На рисунку 2.3 наведено структуру ігрового рівня SITE-108.

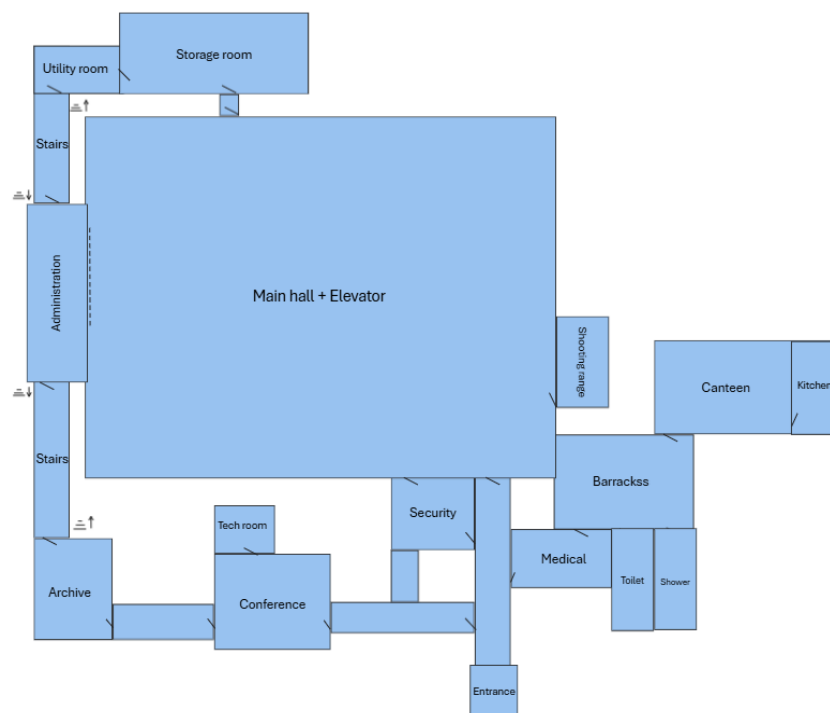


Рис. 2.3 Структура ігрового рівня SITE-108

Структуру рівня доцільно поділити на стартову зону, коридорну навігаційну зону, технічні приміщення, зону моніторингу, службові кімнати, заблоковані або обмежені зони, сюжетно важливі точки та зони повернення після отримання нового доступу.

Стартова зона вводить гравця в ігровий простір і дозволяє безпечно ознайомитися з керуванням, базовою взаємодією та інтерфейсом. Коридорна навігаційна зона виконує роль просторового зв'язку між приміщеннями. Технічні приміщення пов'язані з ремонтом, електроживленням, камерами, вентиляцією або іншими системами. Зона моніторингу може використовуватися для перегляду інформації про стан об'єкта або спостереження за окремими частинами рівня.

Заблоковані зони виконують роль цілей прогресії. Вони створюють у гравця розуміння того, що середовище має приховану структуру, а доступ до неї залежить від виконання певних умов. Сюжетно важливі точки містять інформацію, яка поглиблює розуміння подій у SITE-108 та мотивує продовжувати проходження.

Окрему роль відіграють зони повернення. Якщо гравець після отримання нового ресурсу або відновлення системи повертається до вже знайомого місця, рівень сприймається не як набір окремих кімнат, а як цілісне середовище. Це підсилює ефект занурення та дозволяє використовувати простір більш економно.

Ігровий цикл і структура рівня SITE-108 проєктуються як взаємопов'язані елементи. Ігровий цикл визначає логіку проходження, а структура рівня забезпечує простір для дослідження, обмежень, прогресії та атмосферної напруги.

2.6 Проєктування інтерфейсу користувача

Інтерфейс користувача в ігровому застосунку виконує роль посередника між гравцем і системою. Він має забезпечувати доступ до важливої інформації,

підтримувати взаємодію з об'єктами, допомагати орієнтуватися в доступних діях і водночас не заважати сприйняттю ігрового середовища. Для SITE-108 інтерфейс має бути мінімалістичним, читабельним і стилістично узгодженим з атмосферою хорор-гри.

Проектування інтерфейсу доцільно виконувати з урахуванням принципів людиноорієнтованого дизайну, де важливими є контекст використання, задачі користувача, зрозумілість взаємодії та оцінка зручності рішень [20]. У межах SITE-108 це означає, що інтерфейс не повинен бути просто набором меню, а має підтримувати конкретні ігрові дії: взаємодію з предметами, використання інвентарю, читання системних повідомлень, збереження прогресу та налаштування гри.

До елементів інтерфейсу SITE-108 належать підказка взаємодії, інтерфейс інвентарю, меню паузи, меню налаштувань, меню збереження та завантаження, повідомлення системи, інтерфейс технічних об'єктів або панелей, а також екранні повідомлення про стан гри.

Підказка взаємодії має з'являтися лише тоді, коли гравець дивиться на інтерактивний об'єкт або перебуває достатньо близько до нього. Вона повинна коротко повідомляти, яку дію можна виконати: відкрити, взяти, використати, оглянути, відремонтувати або активувати. Надмірна кількість підказок може знизити напругу, тому їх слід використовувати обмежено й тільки в ситуаціях, де користувачеві потрібне підтвердження доступної дії.

Інтерфейс інвентарю має відображати предмети, які гравець знайшов під час проходження. Для SITE-108 інвентар є важливою частиною прогресії, оскільки предмети можуть відкривати доступ до нових зон або дозволяти відновлювати системи середовища. Тому інтерфейс інвентарю повинен бути достатньо зрозумілим для швидкого вибору предмета, але не має виглядати надмірно яскравим або відірваним від стилістики гри.

Меню паузи призначене для тимчасового припинення ігрового процесу, переходу до налаштувань, збереження або виходу з гри. Його структура має бути простою, щоб користувач міг швидко виконати потрібну дію без зайвого пошуку.

Доцільно передбачити такі пункти: продовжити гру, зберегти гру, завантажити гру, налаштування, вийти до головного меню.

Меню налаштувань має містити параметри, які впливають на комфорт користувача: гучність звуку, чутливість миші, графічні параметри, роздільна здатність та інші базові опції. Такі налаштування не є частиною ігрової прогресії, але впливають на доступність і зручність гри для різних користувачів.

Меню збереження та завантаження повинно надавати можливість керувати ігровими слотами або контрольними точками. Для SITE-108 важливо, щоб користувач розумів, який прогрес буде відновлено після завантаження. Якщо використовуються кілька слотів, доцільно показувати дату, час або коротку інформацію про стан збереження.

Повідомлення системи використовуються для передачі гравцю інформації про події в об'єкті, стан технічних систем або сюжетні сигнали. Вони мають бути стилістично вписані в гру, наприклад через термінали, системні повідомлення або інші внутрішньоігрові засоби. Це дозволяє не руйнувати занурення, оскільки інформація подається як частина світу гри.

Інтерфейс технічних об'єктів або панелей має підтримувати взаємодію з системами середовища. Наприклад, якщо гравець взаємодіє з камерою, електрощитом або дверною панеллю, інтерфейс повинен показати доступну дію, стан системи або причину неможливості виконання дії. Це особливо важливо для прогресії через середовище: користувач має розуміти, що система не просто не працює, а потребує певного ресурсу, доступу або дії.

На рисунку 2.4 наведено приклад екранної форми інвентарю.

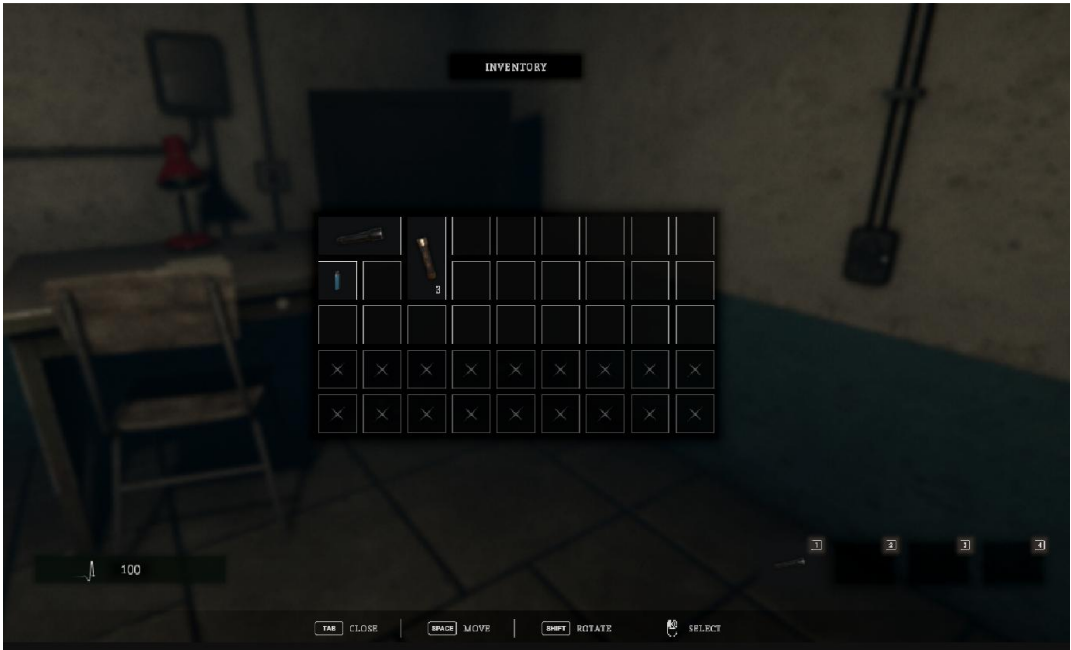


Рис. 2.4 Екранна форма інвентарю персонажа

Вимоги до інтерфейсу користувача наведено в таблиці 2.5.

Таблиця 2.5

Вимоги до інтерфейсу користувача

Позначення	Вимога	Опис
UI-01	Мінімалістичність	Інтерфейс має містити лише ті елементи, які потрібні для поточного сценарію взаємодії.
UI-02	Читабельність	Текстові підказки, меню та повідомлення мають бути зрозумілими й достатньо помітними.
UI-03	Атмосферна відповідність	Візуальний стиль UI має відповідати хорор-тематичі гри та не руйнувати занурення.
UI-04	Контекстність	Підказки взаємодії мають з’являтися лише за наявності доступної дії.
UI-05	Зручність інвентарю	Гравець має швидко знаходити потрібний предмет і розуміти його призначення.

Продовження таблиці 2.5

UI-06	Безпечна робота з меню	Меню паузи, налаштувань і збережень має працювати стабільно та не призводити до втрати прогресу.
UI-07	Ненав'язливість	UI-підказки не повинні перекривати важливі елементи сцени або заважати огляду.

Спроектований інтерфейс має підтримувати основний ігровий цикл: дослідження середовища, взаємодію з об'єктами, використання інвентарю, роботу з технічними системами та збереження прогресу. При цьому він не повинен відволікати увагу гравця, оскільки для хорор-гри важливо зберегти відчуття присутності в ігровому просторі.

Таким чином, у другому розділі було визначено вимоги до ігрового застосунку, описано сценарії взаємодії гравця з системою, спроектовано архітектуру програмного продукту, ключові функціональні модулі, ігровий цикл, структуру рівня та інтерфейс користувача. Отримані проєктні рішення формують основу для подальшої програмної реалізації 3D відеогри SITE-108 у середовищі Unity.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ 3D ВІДЕОГРИ

У цьому розділі наведено опис програмної реалізації гри SITE-108 у середовищі Unity. Основну увагу зосереджено на фактичній структурі Unity-проєкту, організації сцени World, взаємодії між ключовими програмними модулями, реалізації керування персонажем, інвентарю, доступів, технічних систем, збереження прогресу та інтерфейсу користувача.

3.1 Організація Unity-проєкту

Архітектура програмної реалізації SITE-108 побудована відповідно до компонентної моделі Unity. У Unity сцени є окремими ресурсами, які містять цілі рівні, меню або частини гри, а відбір і порядок сцен у збірці задаються явно. Базовими елементами будь-якої сцени є `GameObject`, що виступає контейнером для функціональних компонентів, і `MonoBehaviour`, який надає життєвий цикл сценаріям через методи `Awake`, `Start`, `Update` та інші події [10; 12]. Саме така модель дозволяє поєднати в одному проєкті логіку гравця, взаємодію з об'єктами, UI, атмосферні системи та збереження прогресу.

У поточній реалізації використано Unity 6000.3.10f1. Ключовою ігровою сценою є `World`, у межах якої розміщено головний рівень, гравця, системні менеджери, інтерфейс користувача та інтерактивні об'єкти. У межах кваліфікаційної роботи сцена `World` розглядається як основне середовище реалізації ігрових сценаріїв SITE-108. Допоміжні технічні або тестові сцени, які не формують цілісний ігровий досвід SITE-108, у тексті роботи не розглядаються.

Структура папок Unity-проєкту організована за функціональним принципом. Окремо виділяються каталоги для сцен, скриптів, налаштувань, даних, префабів, матеріалів, моделей, звуків та елементів інтерфейсу. Такий

підхід спрощує навігацію в проєкті та узгоджується із загальною практикою роботи з активами й сценами в Unity, де окремі ресурси зберігаються як незалежні assets і повторно використовуються в межах сцени або кількох сцен [10; 17]. Структуру папок Unity-проєкту наведено на рисунку 3.1.

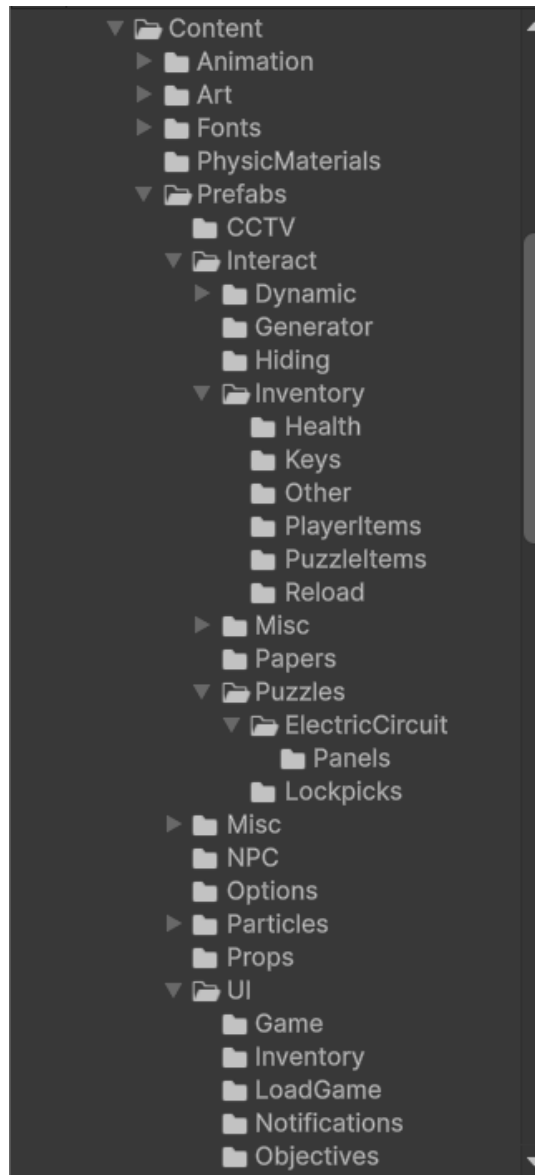


Рис. 3.1 Структура папок Unity-проєкту в Project Window

У структурі сцени World центральну роль відіграють системні об'єкти, згруповані навколо менеджерів гри та героя. На рівні ієрархії доцільно виділити такі вузли: GAMEMANAGER, що об'єднує UI, рендеринг, звук та службові системи, і HEROPLAYER, який містить камеру, віртуальну камеру, точку для

предметів у руках та набір компонентів керування. Така організація відповідає підходу Unity, за якого логіка не концентрується в одному монолітному класі, а розподіляється між компонентами об'єкта сцени.

На об'єкті HEROPLAYER використано CharacterController, PlayerManager, PlayerStateMachine, PlayerHealth, звукові компоненти та допоміжні скрипти персонажа. В інспекторі видно, що PlayerManager зберігає посилання на FPView, основну камеру та віртуальну камеру, а CharacterController задає фізичні параметри капсули персонажа. Офіційна документація Unity визначає CharacterController як компонент, призначений насамперед для керування персонажем від першої або третьої особи без потреби покладатися на Rigidbody; переміщення такого персонажа відбувається через виклики Move, а зіткнення обмежують траєкторію руху [12].

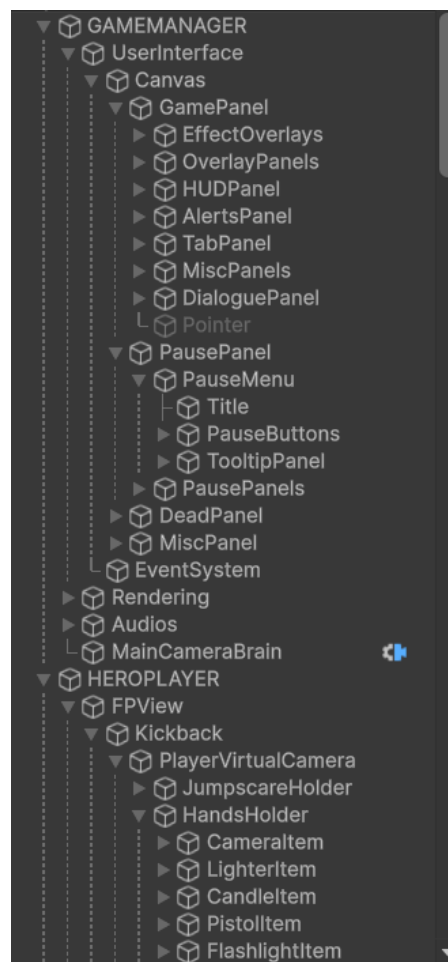


Рис. 3.2 Фрагмент ієрархії сцени World

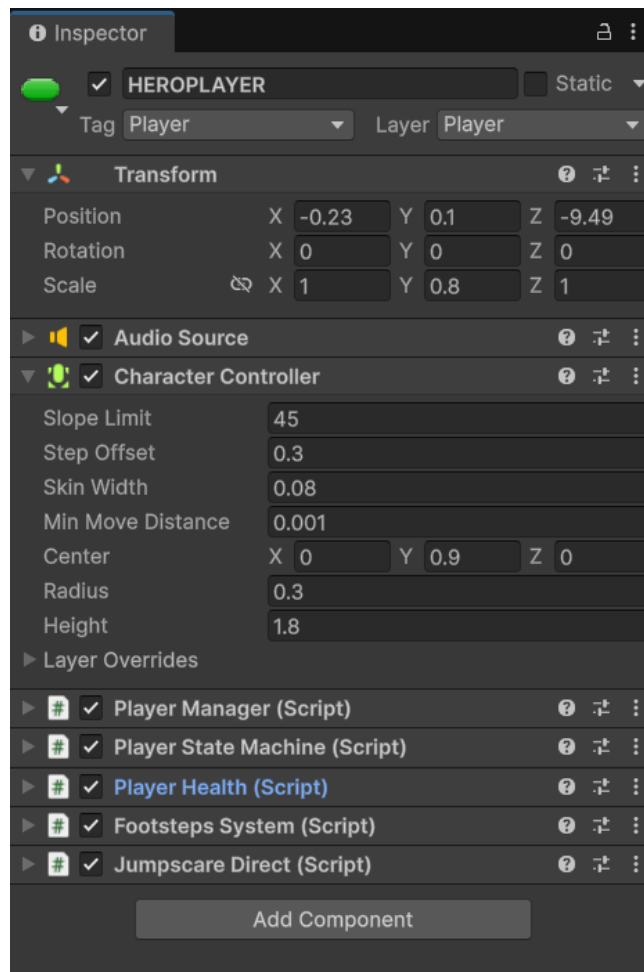


Рис. 3.3 Інспектор об'єкта HIEROPLAYER з компонентами CharacterController, PlayerManager і PlayerStateMachine

Окремою особливістю проєкту є логічне групування об'єктів середовища за кімнатами. У межах кожної кімнати дочірні об'єкти додатково згруповані за функціональною ознакою: меблі, окремі полиці, предмети на полицях, двері, світильники, декоративні елементи. Така організація важлива не лише для зручності редагування, а й для подальшого опису збереження стану, локалізації інтерактивних об'єктів та побудови проходження.

3.2 Діаграма класів та специфікація ключових програмних модулів

На рівні програмних класів проєкт SITE-108 можна розглядати як сукупність взаємопов'язаних модулів: керування персонажем, взаємодія,

інвентар, інтерактивні об'єкти, технічні системи, UI та збереження прогресу. Компонентна природа Unity означає, що класи не існують ізольовано: вони прикріплюються до об'єктів сцени, зберігають посилання один на одного і взаємодіють через інтерфейси, події та менеджери. Базова модель MonoBehaviour якраз і надає для цього точки входу в життєвий цикл сцени й об'єкта.

Для опису структури застосунку доцільно використовувати діаграму класів, у якій центральними елементами виступають PlayerManager, PlayerStateMachine, InteractController, Inventory, DynamicObject. Діаграма відображає логічне ядро проходження: гравець, взаємодія, предмет/двері/технічний об'єкт, зміна стану світу.

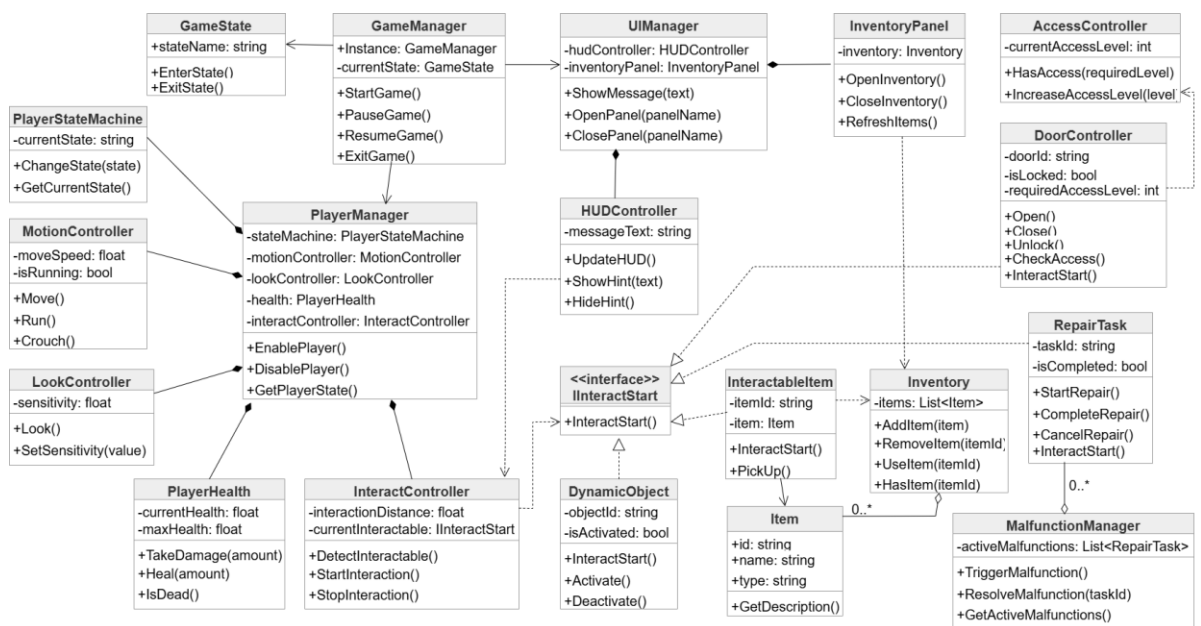


Рис. 3.4 Діаграма класів

Специфікацію ключових програмних модулів наведено в таблиці 3.1.

Таблиця 3.1

Специфікація ключових програмних модулів

Клас	Призначення	Поля	Методи
GameManager	Керує основними станами гри, UI-панелями, підказками взаємодії та ігровими повідомленнями.	GamePanel, PausePanel, InventoryPanel, ReticleImage, InteractInfoPanel	TogglePause(), ShowPanel(), DisplayHintMessage()
PlayerManager, PlayerStateMachine	Забезпечують доступ до компонентів гравця та обробку станів руху персонажа.	MainCamera, PlayerCollider, PlayerHealth, CharacterController, поточний стан	SetState(), OnJumpTriggered()
InteractController	Виявляє інтерактивні об'єкти перед гравцем і запускає взаємодію з ними.	RaycastRange, HoldDistance, InteractLayer, RaycastObject	OnUpdate(), OnInteractStart(), OnHoldStart()
DynamicObject	Реалізує інтерактивні об'єкти середовища: двері, перемикачі, важелі та інші об'єкти зі змінним станом.	dynamicType, dynamicStatus, interactType, target, unlockItem, isLocked	InteractStartPlayer(), InteractHold(), SetLockedStatus(), SetOpenState(), OnSave(), OnLoad()

Продовження таблиці 3.1

Inventory	Керує предметами гравця, контейнерами, слотами інвентарю та збереженням стану інвентарю.	settings, items, carryingItems, currentContainer	AddItem(), RemoveItem(), OpenContainer(), OpenItemSelector(), OnCustomSave()
CCTV_CameraSystem	Забезпечує роботу системи відеоспостереження, перемикання між камерами та керування переглядом.	список камер, активна камера, параметри повороту, режим відображення сигналу	SetMonitorCameraOutput(), SwitchCamera(), ControlCameras()
PowerGenerator	Реалізує логіку роботи генератора, запасу пального та підключених споживачів енергії.	MaxFuelLiters, CurrentFuelLiters, GeneratorEfficiency, MotorFuelDrainPerHour	StartGenerator(), StopGenerator(), AddPowerConsumer(), RemovePowerConsumer()
SaveGameManager	Відповідає за збереження стану гри, підготовку завантаження та роботу зі збережуваними об'єктами.	GameLoadType, LoadSceneName, ObjectReferences, worldSaveables, runtimeSaveables	SaveGame(), SetLoadGameState(), TryDeserializeGameStateAsync(), RemoveAllSaves()

З наведених у таблиці класів найбільш важливими для реалізації ігрового процесу SITE-108 є InteractController, DynamicObject, Inventory і SaveGameManager. InteractController відповідає за виявлення інтерактивного об'єкта перед гравцем, DynamicObject змінює стан об'єктів середовища,

Inventory забезпечує роботу з предметами, а SaveGameManager фіксує стан проходження. Разом ці класи формують основу прогресії через середовище: гравець знаходить об'єкт, взаємодіє з ним, використовує потрібний предмет або змінює його стан, після чого результат може бути збережений. CCTV_CameraSystem і PowerGenerator доповнюють цю структуру технічними системами, що розширюють варіанти взаємодії з ігровим простором і підтримують атмосферу об'єкта SITE-108.

3.3 Реалізація керування персонажем і системи взаємодії

Керування персонажем у SITE-108 реалізовано через поєднання CharacterController, PlayerManager та PlayerStateMachine. Такий підхід є виправданим для гри від першої особи, оскільки CharacterController забезпечує рух із урахуванням колізій без використання фізичної моделі Rigidbody, а окремий керуючий скрипт може відповідати за зміну станів руху, обмеження вводу та синхронізацію з камерою. Офіційна документація Unity прямо рекомендує CharacterController для сценаріїв персонажа від першої або третьої особи, де потрібен контрольований рух і перевірка зіткнень [12].

У поточній структурі PlayerManager виконує роль точки доступу до ключових компонентів персонажа. Саме він об'єднує посилання на камеру, віртуальну камеру, контролер персонажа та інші підсистеми. PlayerStateMachine, у свою чергу, відповідає за вибір і перемикання станів руху: звичайне пересування, прискорення, можливі обмежені режими та спеціальні стани, пов'язані з контекстом сцени. Це дозволяє розвести загальну координацію гравця і конкретну логіку зміни станів.

Система взаємодії побудована на променевій перевірці перед камерою. У термінах Unity така взаємодія природно відображається через Physics.Raycast, який виконує запит до колайдерів у сцені та повертає інформацію про об'єкт і точку перетину [12]. Саме тому описаний у технічних

матеріалах `InteractController` зберігає посилання на поточний об'єкт взаємодії та локальну точку влучання, після чого делегує дію відповідному інтерактивному компоненту.

У реалізації `SITE-108` виявлення об'єкта та виконання дії розділені. Ядро перевірки виконується контролером взаємодії, а сам об'єкт сцени реалізує один або кілька контрактів взаємодії. Це добре видно на прикладі `DynamicObject`, що приймає старт взаємодії від гравця, обробляє тримання дії та її завершення.



Рис. 3.5 Приклад підказки взаємодії при наведенні на об'єкт

Фрагмент обробки взаємодії з динамічним об'єктом:

```
public void InteractStartPlayer(GameObject player)
{
    if (isInteractLocked) return;
```

```

    PlayerManager playerManager = player.GetComponent<PlayerManager>();
    CurrentDynamic?.OnDynamicStart(playerManager);
}

public void InteractHold(Vector3 point)
{
    Vector2 delta =
InputManager.ReadInput<Vector2>(Controls.POINTER_DELTA);
    if (!isLocked) CurrentDynamic?.OnDynamicHold(delta);
}

public void InteractStop()
{
    if (!isLocked) CurrentDynamic?.OnDynamicEnd();
}

```

Наведений фрагмент показує, що власне обробка взаємодії не зашивається в контролер гравця. Натомість DynamicObject приймає початок взаємодії, за потреби передає посилання на PlayerManager, а подальша поведінка визначається поточним типом об'єкта. Це важливо для масштабованості проєкту: двері, важелі, перемикачі та інші схожі об'єкти працюють за спільною моделлю.

Ще одним важливим елементом є контекстні підказки. Скрипт CustomInteractTitle дозволяє формувати назву поточної взаємодії динамічно — залежно від стану об'єкта. У результаті UI не просто показує уніфіковане “Використати”, а може змінювати текст на “Відкрити”, “Закрити”, “Увімкнути”, “Вимкнути”, “Оглянути” тощо, що суттєво підвищує зрозумілість взаємодії для гравця.

Фрагмент формування динамічної назви взаємодії:

```
public TitleParams InteractTitle()
```

```

{
    string title = Title;
    string useTitle = UseTitle;
    string examineTitle = ExamineTitle;

    if (!OverrideTitle) title = null;
    else if (UseTitleDynamic) title = DynamicTitle.Value ? TrueTitle : FalseTitle;

    if (!OverrideUseTitle) useTitle = null;
    else if (UseUseTitleDynamic) useTitle = DynamicUseTitle.Value ?
TrueUseTitle : FalseUseTitle;

    return new TitleParams()
    {
        title = title,
        button1 = useTitle,
        button2 = examineTitle
    };
}

```

Таким чином, система взаємодії у SITE-108 складається з трьох шарів: виявлення об'єкта перед камерою, делегування виклику відповідному компоненту та текстового супроводу дії через UI.

3.4 Реалізація інвентарю, доступів та ігрової прогресії

Інвентар у SITE-108 визначає багато важливих аспектів гри, зокрема спосіб прогресії гравця. Розширення можливостей відбувається через накопичення і використання предметів, ключ-карт, ресурсів та контекстних інструментів доступу. Це відповідає логіці гри з елементами RPG, де розвиток виражається

через нові сценарії взаємодії з простором.

На рівні реалізації Inventory побудовано як окремий керуючий модуль із сіткою слотів, підтримкою стекування предметів, шорткатів та серіалізації вмісту. З технічної точки зору така система добре узгоджується з data-driven підходом Unity, де дані предметів можуть бути винесені в окремі об'єкти даних, а логіка роботи структурується в компонентах і UI-елементах. Unity рекомендує використовувати ScriptableObject для централізації таких даних, які мають жити незалежно від конкретного GameObject у сцені [12].

Інвентар має конфігуровану сітку, підтримує кількість предметів, орієнтацію предмета та його позицію в сітці. Додавання нового предмета виконується із врахуванням стекування та вільного простору, а при неможливості розмістити об'єкт система може повернути повідомлення про помилку та заборонити підбір.

Фрагмент коду додавання предмета до інвентарю:

```
public bool AddItem(string guid, ushort quantity, ItemCustomData customData, out
InventoryItem inventoryItem)
{
    inventoryItem = null;
    int availableQ = quantity;

    if (items.ContainsKey(guid))
    {
        Item item = items[guid];
        ushort maxStack = item.Properties.maxStack;

        if (item.Settings.isStackable && ContainsItemMany(guid, out OccupyData[]
itemDatas))
        {
            // спроба доповнити існуючі стеки
            ...
        }
    }
}
```

```

    }
else
{
    inventoryItem = CreateItem(guid, availableQ, customData);
    OnItemAdded.OnNext(new(guid, availableQ));
}
}
return inventoryItem != null;
}

```

Важливо, що інвентар інтегрований із системою доступів. У `DynamicObject` передбачено поля `unlockItem`, `keepUnlockItem`, `statusChange`, а це означає, що об'єкт може перевіряти наявність потрібного предмета і змінювати стан із заблокованого на доступний. У поточній версії гри фактично реалізовано двері, що відкриваються за ключ-картами або іншими умовами доступу. Це є прямим проявом середовищної прогресії: знайдений предмет змінює не характеристику героя, а стан світу.

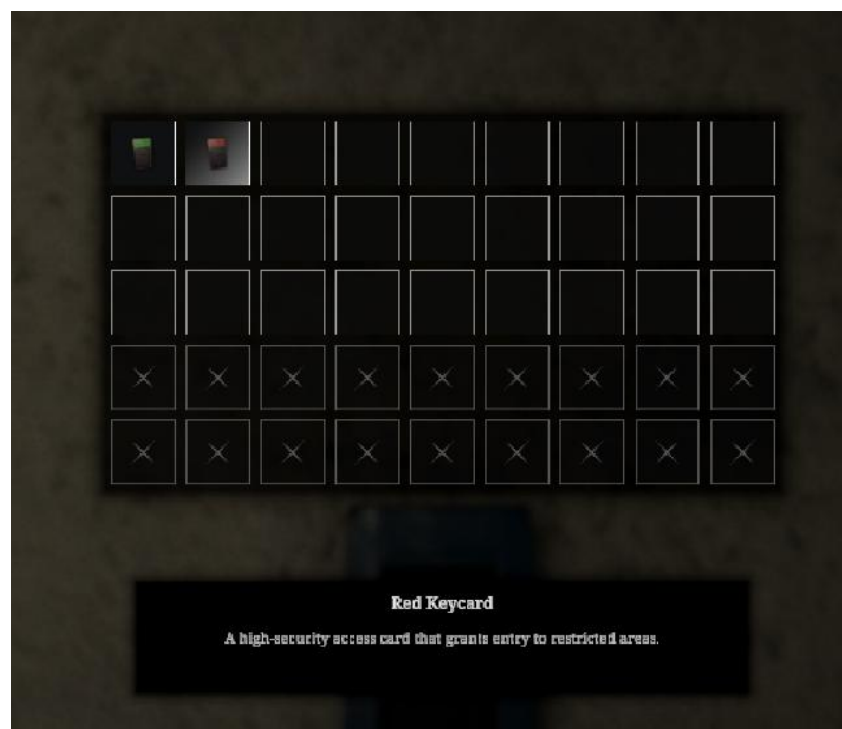


Рис. 3.6 Екран інвентарю з предметом доступу



Рис. 3.7 Приклад об'єкта, що потребує ключ-карти

Окремої уваги заслуговує механізм вибору предмета для цільової взаємодії. Метод `OpenItemSelector()` переводить інвентар у режим селектора, коли гравець відкриває список предметів і надає один з них конкретному сценарію взаємодії. Саме так можуть працювати ключ-карти, ремонтні ресурси, спеціальні предмети для пазлів або інші пропускні об'єкти.

```
public void OpenItemSelector(IInventorySelector inventorySelector)
{
    itemSelector = true;
    this.inventorySelector = inventorySelector;
    gameManager.ShowInventoryPanel(true);
}
```

У проєкті також передбачено збереження структури інвентарю, орієнтації предметів у сітці та шорткатів. Це критично для ігор такого типу: після завантаження гравець має повернутися не лише до тієї самої точки рівня, а й до

того самого стану ресурсів і доступів.

3.5 Реалізація технічних систем, несправностей і моніторингу

Однією з відмінних рис SITE-108 є інтеграція технічних об'єктів у саму тканину ігрового процесу. Йдеться про системи, з якими гравець може вступати у функціональну взаємодію: монітори камер, генератор, технічні вузли середовища, обмеження доступу.

У поточному стані проєкту повноцінно реалізовано систему CCTV-моніторингу. Гравець підходить до телевізора, активує його, після чого переходить у повноекранний режим взаємодії. У цьому режимі на екрані відображається заздалегідь визначений список камер. Далі гравець може перемикатися між ними вперед і назад, повертати камеру мишею та виходити з режиму за клавішею Q. Також телевізор можна вимкнути. З погляду Unity такий підхід є технічно обґрунтованим: RenderTexture призначено саме для ситуацій, коли камера рендерить зображення не на екран напряму, а в текстуру; типовим прикладом використання є якраз камери спостереження [12]. Для цього камера отримує targetTexture, а сама текстура може бути виведена на матеріал або UI-елемент.



Рис. 3.8 Монітор CCTV у звичайному режимі взаємодії



Рис. 3.9 Повноекранний режим перегляду CCTV з перемиканням камер

Перевага такої системи полягає в тому, що вона розширює поле сприйняття гравця, але не знімає напругу повністю. Гравець отримує додаткову інформацію про простір, однак для цього мусить зупинитися, сфокусуватися на екрані та тимчасово переключити контекст взаємодії. Це добре узгоджується з хорор-логікою: доступ до інформації є корисним, але потребує від гравця жертвувати чимось.

Ще одним технічним модулем є генератор, програмна логіка якого реалізована класом `PowerGenerator`. Цей клас містить поля для зберігання поточного обсягу пального, параметрів ефективності, швидкості витрати ресурсу та списку споживачів енергії. Завдяки цьому генератор може використовуватися як основа для системи електроживлення, у якій стан технічних об'єктів залежить від наявності ресурсу та активності джерела живлення.

У межах поточної версії SITE-108 генератор і пов'язані електротехнічні об'єкти присутні в ігровій сцені як частина технічної інфраструктури рівня. Їх реалізація створює програмну основу для подальшого розширення сценаріїв проходження, зокрема для зв'язку між джерелом живлення, технічними системами та обмеженнями доступу. У даному розділі генератор розглядається саме як реалізований програмний компонент енергетичної системи, тоді як конкретні сценарії його використання в ігровому циклі можуть бути розширені на наступних етапах розвитку проєкту.

3.6 Реалізація системи збереження ігрового прогресу

Система збереження є однією з найважливіших частин програмної реалізації SITE-108, оскільки вона фіксує не лише позицію гравця, а й стан світу. Для такого типу гри це принципово важливо: після завантаження необхідно відновити відкриті двері, заблоковані доступи, вміст інвентарю, динамічні об'єкти та локальні зміни середовища.

У технічній реалізації центральним класом виступає `SaveGameManager`. Він координує збереження, відновлення, стежить за списками об'єктів `worldSaveables` і `runtimeSaveables`, а також взаємодіє з `Inventory`, `PlayerManager` і файлами на диску. Для представлення даних використовується JSON-модель на базі `Json.NET: JsonConvert.SerializeObject()` перетворює об'єктну структуру на рядок JSON, а `JsonObject.Parse()` дозволяє зчитати назад дерево даних з текстового вмісту [21]. Такий підхід добре підходить до ієрархічного стану ігрового світу, де в одному контейнері треба зберегти як дані гравця, так і змінені об'єкти сцени.

Схема збереження у проєкті поділена на три рівні. По-перше, формується блок даних гравця: позиція, поворот, локальні параметри та глобальні дані, до яких належать інвентар та цілі. По-друге, формується світовий блок, де серіалізуються всі зареєстровані об'єкти сцени та всі `runtime`-об'єкти, які були створені під час гри. По-третє, створюються метадані збереження: ідентифікатор, активна сцена, час створення, тривалість гри, тип збереження та, за наявності, `thumbnail`.

Фрагмент формування буфера збереження:

```
private async void PrepareAndSaveGameState()
{
    StorableCollection worldStatePlayerData = new();

    var (position, rotation) = playerPresence.GetPlayerTransform();
```

```

worldStatePlayerData.Add("position", position.ToSaveable());
worldStatePlayerData.Add("rotation", rotation.ToSaveable());

StorableCollection globalPlayerData = GetPlayerDataBuffer();
StorableCollection localPlayerData =
playerPresence.PlayerManager.OnCustomSave();

StorableCollection saveablesBuffer = new()
{
    { "id", saveId },
    { "playerData", worldStatePlayerData },
    { "worldState", GetWorldStateBuffer() }
};

await SerializeGameState(saveInfoData, saveablesBuffer, saveFolderName,
saveFolderPath);
}

```

Відновлення стану реалізовано через інтерфейсну модель `ISaveable` та окремі класи на кшталт `SaveableObject` і `SaveableBehaviour`. `SaveableObject` зберігає вибрані атрибути об'єкта за допомогою прапорців: позицію, поворот, масштаб, активність, стан рендера та активність допоміжних компонентів. Це означає, що не кожен об'єкт мусить серіалізувати весь свій стан; зберігається лише те, що потрібно в межах конкретної ролі об'єкта.

Фрагмент збереження та відновлення окремого об'єкта сцени:

```

public StorableCollection OnSave()
{
    StorableCollection storableCollection = new();

```

```

    if (SaveableFlags.HasFlag(SaveableFlagsEnum.Position))
        storableCollection.Add("position", transform.position.ToSaveable());

    if (SaveableFlags.HasFlag(SaveableFlagsEnum.Rotation))
        storableCollection.Add("rotation", transform.eulerAngles.ToSaveable());

    if (SaveableFlags.HasFlag(SaveableFlagsEnum.ObjectActive))
        storableCollection.Add("objectActive", gameObject.activeSelf);

    return storableCollection;
}

public void OnLoad(JToken data)
{
    if (SaveableFlags.HasFlag(SaveableFlagsEnum.Position))
        transform.position = data["position"].ToObject<Vector3>();

    if (SaveableFlags.HasFlag(SaveableFlagsEnum.Rotation))
        transform.eulerAngles = data["rotation"].ToObject<Vector3>();

    if (SaveableFlags.HasFlag(SaveableFlagsEnum.ObjectActive))
        gameObject.SetActive((bool)data["objectActive"]);
}

```

Для runtime-об'єктів використовується `SaveableBehaviour`, який вимагає наявності `UniqueID`. Це важливо, бо під час завантаження гри потрібно не лише відновити значення полів, а й повторно інстанціювати ті об'єкти, які були створені під час проходження.

Підсистема `SaveGameReader` виконує допоміжну, але важливу функцію: читає список доступних збережень, завантажує їхні метадані та, за потреби,

thumbnail. Саме завдяки цьому в UI можна показати назву сцени, час збереження, тривалість проходження та зображення місця, де було зроблено save.

3.7 Реалізація інтерфейсу користувача

Інтерфейс користувача в SITE-108 організовано у вигляді набору окремих панелей, розміщених у межах загального об'єкта Canvas. Така структура дозволяє розділити елементи інтерфейсу за їхнім призначенням і вмикати лише ті блоки, які потрібні в конкретному стані гри. У межах ієрархії інтерфейсу виділено панелі ігрового режиму, інвентарю, паузи, повідомлень, діалогів та стану завершення гри.

Ігровий інтерфейс згруповано в межах GamePanel. До нього належать панелі для відображення HUD-елементів, повідомлень, візуальних накладень, діалогів та інвентарю. Такий підхід дозволяє не змішувати постійні елементи ігрового процесу з тимчасовими повідомленнями або окремими режимами взаємодії. Наприклад, інвентар розміщено як окремий підблок, оскільки його відкриття змінює спосіб взаємодії гравця з предметами.

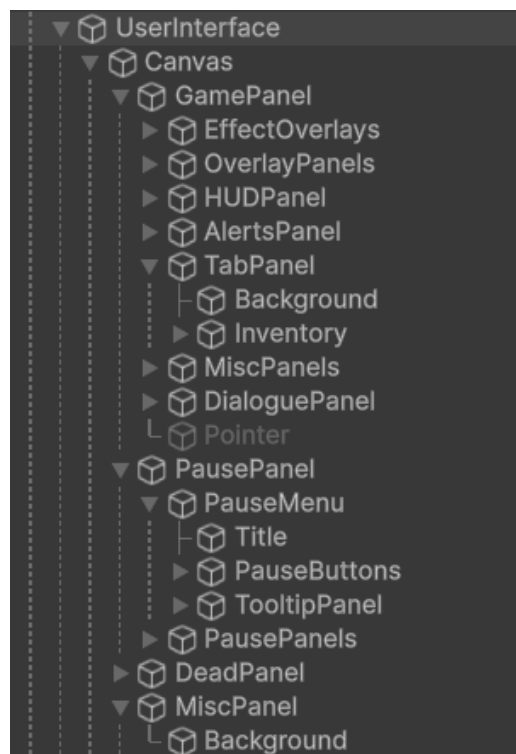


Рис. 3.10 Ієрархія Canvas і UI-панелей

Також реалізовано PausePanel, у межах якої розміщено меню паузи, заголовок, кнопки керування та допоміжну панель підказок. Цей блок використовується тоді, коли активний ігровий процес тимчасово призупиняється, а користувач отримує доступ до службових дій. Для станів, пов'язаних із завершенням проходження або додатковими повідомленнями, передбачено окремі панелі DeadPanel і MiscPanel. Така організація UI спрощує керування інтерфейсом і дозволяє підтримувати різні стани гри без перевантаження одного екрана великою кількістю елементів.

3.8 Формування ігрового середовища

Ігрове середовище SITE-108 формувалося з урахуванням принципів левел-дизайну для сюжетної 3D-гри з елементами хорору. Основна задача полягала у створенні простору, який підтримує дослідження, орієнтацію гравця, поступове розкриття об'єкта та відчуття ізоляції. Тому під час побудови рівня враховувалися логіка переміщення, візуальні орієнтири, ритм проходження, щільність наповнення приміщень і загальна атмосфера технічного об'єкта.

Основна сцена гри має назву World. Її простір побудовано як внутрішню структуру об'єкта SITE-108, що складається з коридорів, технічних кімнат, службових приміщень і локальних зон взаємодії. Коридори виконують роль навігаційного каркаса, а окремі кімнати створюють точки інтересу, у яких гравець може знаходити предмети, інформацію або об'єкти для взаємодії.



Рис. 3.11 Фрагмент ігрової сцени World

Під час побудови рівня застосовано принцип поступового ускладнення простору. Початкові ділянки мають бути зрозумілими для освоєння керування та базової навігації, тоді як подальші зони містять більше розгалужень, затемнених ділянок, технічних приміщень і закритих проходів. Це підтримує інтерес до дослідження та створює відчуття внутрішньої логіки об'єкта.

Керування увагою гравця здійснюється через освітлення, композицію приміщень, розташування об'єктів і візуальні акценти. Важливі елементи розміщуються так, щоб їх можна було помітити під час природного руху рівнем, але без надмірно прямого підказування кожної дії. Завдяки цьому середовище залишається читабельним, але зберігає невизначеність, необхідну для хорор-атмосфери.

Для зручності редагування сцена організована в ієрархії Unity за кімнатами та функціональними зонами. Усередині приміщень об'єкти групуються за змістом і розташуванням: наприклад, меблі можуть об'єднуватися з предметами на полицях, а джерела освітлення та декоративні елементи належать до відповідної кімнати. Така структура спрощує подальше наповнення рівня, коригування композиції та роботу зі сценою.

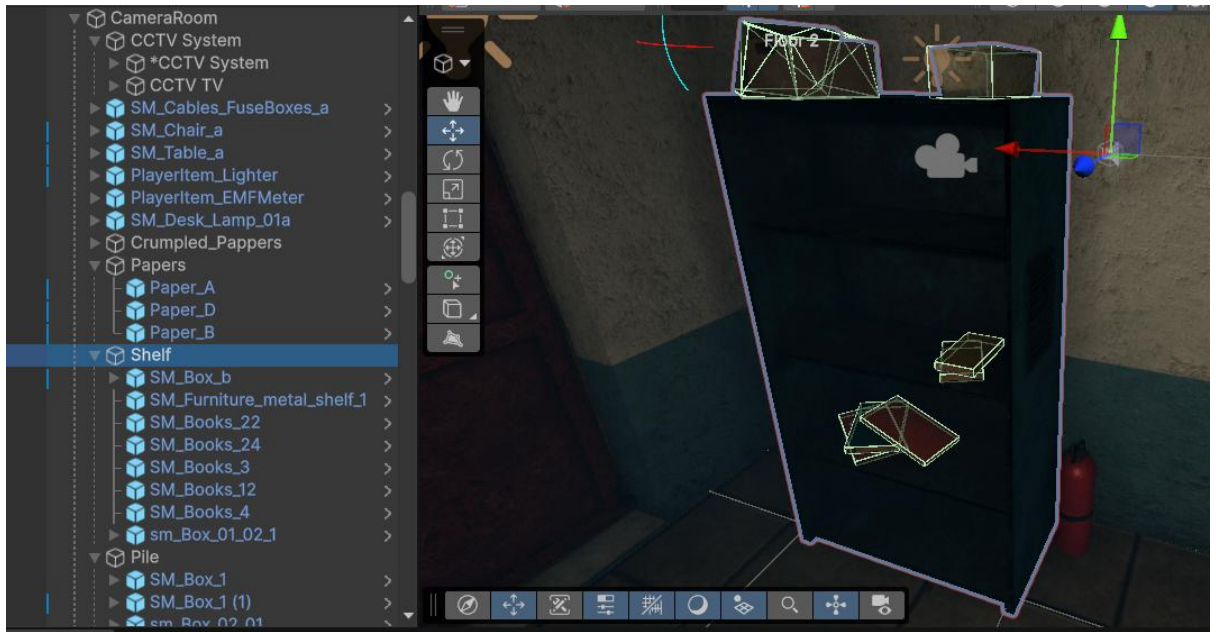


Рис. 3.12 Організація об'єктів рівня в ієрархії Unity

Сформоване середовище визначає не лише зовнішній вигляд гри, а й темп проходження, спосіб орієнтації гравця та загальне відчуття перебування всередині ізолюваного технічного об'єкта. Саме через структуру рівня, освітлення, розміщення об'єктів і просторові обмеження підтримується атмосфера SITE-108.

4 ТЕСТУВАННЯ ІГРОВОГО ЗАСТОСУНКУ

4.1 Обґрунтування вибору видів тестування

Вибір видів тестування для SITE-108 визначався структурою реалізованого застосунку. Оскільки гра поєднує керування персонажем, взаємодію з об'єктами середовища, систему предметів та інвентарю, CCTV-моніторинг, UI-панелі та збереження стану сцени, було необхідно не обмежуватися лише перевіркою окремих функцій. Саме тому до розділу включено: функціональне тестування основних механік, сценарне тестування проходження та інтерфейсу, перевірку системи збереження прогресу, тестування стабільності й продуктивності, а також попереднє внутрішнє оцінювання ігрового досвіду.

Первинна авторська перевірка залишалася доцільною на етапі розробки для швидкого локального виявлення помилок у логіці окремих скриптів, інтерфейсних переходах і збереженні станів. Водночас вона не є достатньо незалежною як контрольна процедура, оскільки автор перевіряє продукт відповідно до власного задуму, а не лише з позиції кінцевого користувача. Тому для контрольної перевірки було залучено внутрішню команду HitekGames, яка проходила однаковий тестовий фрагмент і фіксувала результати в уніфікованій формі.

Для технічної частини тестування використано вбудовані засоби Unity. Згідно з офіційною документацією, Profiler window містить список захоплених сесій, модулі профілювання, елементи керування записом і панель деталізації вибраного кадру; модулі надають покадрові дані щодо CPU, рендерингу, пам'яті, UI та інших аспектів виконання [10]. У свою чергу, вікно Rendering Statistics у Game view відкривається кнопкою Stats та показує реальні показники рендерингу в Play Mode, що робить його зручним інструментом для порівняння

сцени до і після оптимізації.

Таблиця 4.1

Тестове середовище та інструменти перевірки

Операційна система	Windows 11
Середовище розробки	Unity 6000.3.10f1
Тестова сцена	World
Роздільна здатність	1920×1080
Формат перевірки	Авторська перевірка під час розробки; внутрішнє тестування HitekGames
Основні інструменти	Unity Console, Play Mode, Unity Profiler, Game View Stats

Таким чином, обраний набір перевірок охоплює як коректність базової логіки, так і цілісний досвід взаємодії з грою. Це особливо важливо для SITE-108, де відчуття напруги, орієнтація в середовищі та технічна стабільність прямо впливають на якість сприйняття ігрового фрагмента.

4.2 Функціональне тестування основних ігрових механік

Функціональне тестування було спрямоване на перевірку ядра ігрової взаємодії. До нього увійшли: запуск сцени World, переміщення персонажа, огляд простору камерою, виявлення інтерактивних об'єктів у полі зору, підбір предметів, відкриття інвентарю, використання предметів доступу, базова взаємодія з динамічними об'єктами середовища, а також активація та вихід із режиму CCTV.

Перевірка показала, що основна функціональна петля гри працює коректно: персонаж стабільно пересувається сценою, об'єкти взаємодії визначаються в передній півсфері, підбір предметів переводить їх до інвентарю,

а об'єкти з умовами доступу змінюють стан відповідно до наявності необхідного предмета. Режим CCTV активується з інтерактивного монітора, а після виходу керування знову передається персонажу без руйнування загального сценарію.

Окремо слід відзначити, що функціональні перевірки виконувалися не лише як ізольований набір дій, а і як послідовність пов'язаних переходів. Такий підхід дозволив виявити не лише логічні дефекти, а й помилки на межі між підсистемами, коли одна дія формально виконувалася правильно, але її завершення не очищало стан для наступного етапу проходження. Саме ця особливість зумовила потребу в окремому сценарному тестуванні, описаному далі.

У підсумку функціональна перевірка підтвердила працездатність основних механік після циклу виправлень. Формалізовані результати наведено в таблиці тест-кейсів у подальшому підрозділі.

4.3 Тестування сценаріїв проходження та взаємодії з інтерфейсом

Сценарне тестування було спрямоване на перевірку короткого ігрового фрагмента як єдиної послідовності дій, а не набору розрізнених функцій. Контрольний маршрут включав старт у сцені, огляд найближчого простору, підбір предмета, відкриття інвентарю, взаємодію з об'єктом доступу, перехід у технічну зону, активацію CCTV-монітора, перемикання камер і повернення до звичайного режиму керування. Саме в таких сценаріях найчастіше виявляються проблеми, які не видно під час ізольованої перевірки окремих кнопок чи скриптів.

Особливу увагу було приділено інтерфейсу. Для SITE-108 інтерфейс не повинен змагатися з атмосферою сцени: він має лише підказувати доступні дії, не перевантажуючи гравця зайвими службовими ознаками. Тому в сценарному тестуванні окремо перевірялися підказки взаємодії, сповіщення про підбір

предметів, pause-меню, інвентар та поведінка курсора під час спеціальних режимів.

Під час цієї перевірки було зафіксовано сім дефектів різного рівня впливу. Вони наведені в таблиці 4.2.

Таблиця 4.2

Виявлені дефекти та результати виправлення

ID	Опис дефекту	Рівень впливу	Результат
BUG-01	Стан однієї з подій поламки не зберігався між збереженнями гри	Критичний	Виправлено
BUG-02	Сповіщення про підбір предмета з'являлося на екрані та не зникало після завершення дії	Середній	Виправлено
BUG-03	У режимі керування камерою відображався системний курсор, який не мав бути видимим	Середній	Виправлено
BUG-04	У технічній зоні сцени спостерігалось падіння продуктивності через велику кількість активних об'єктів і джерел освітлення	Середній	Оптимізовано
BUG-05	Частина стін або дверних елементів була розміщена без точного snapping, що створювало візуальні зазори за межі мапи	Низький	Виправлено
BUG-06	Виведення зображення камери на текстуру об'єкта працювало некоректно на неплоских поверхнях	Низький	Виправлено
BUG-07	Деякі елементи інтерфейсу могли накладатися один на одного, зокрема інвентар і меню паузи	Низький	Виправлено

Отже, у межах сценарного проходження було зафіксовано 1 критичний, 3 дефекти середнього рівня впливу та 3 дефекти низького рівня впливу. Найсуттєвішим виявився дефект BUG-01, оскільки він напряду загрожував цілісності проходження. Середні за рівнем впливу відхилення не руйнували гру повністю, але погіршували керованість і візуальну ясність. Низькі дефекти переважно стосувалися візуальної цілісності сцени та коректної взаємодії з UI.

Після внесення виправлень було проведено повторне проходження того самого сценарію. За його результатами критичних і середніх відхилень у перевірених сценаріях більше не виявлено, а низькі зауваження було усунуто або локалізовано архітектурним рішенням, як у випадку з виведенням зображення камери на плоский екран.

4.4 Тестування системи збереження ігрового прогресу

Система збереження в SITE-108 є критично важливою з двох причин. По-перше, вона повинна відновлювати не лише позицію гравця, а й фактичний стан сцени: підібрані предмети, змінені стани об'єктів, уже відкриті доступи та ключові події проходження. По-друге, порушення цілісності збереження може створити помилки, які не проявляються одразу, але блокують подальшу логіку сценарію.

Перевірка охоплювала кілька типових послідовностей: збереження після підбору предмета, збереження після зміни стану інтерактивного об'єкта, проміжне збереження під час проходження та завантаження після повторного запуску тестової сцени. Окремо перевірялася подія поламки, яка в попередній версії не відновлювала свій стан між сейвами. Саме цей дефект було класифіковано як критичний, оскільки він міг порушити причинно-наслідковий ланцюг проходження.

Після виправлення логіки серіалізації та повторної перевірки встановлено,

що стан контрольованої події поламки відновлюється коректно. Також стабільно відновлювалися підібрані предмети, зміни стану об'єктів і доступних взаємодій. Це дозволяє розглядати систему збереження як придатну для базового проходження тестового фрагмента гри.

З практичної точки зору саме перевірка збереження стала одним із найважливіших етапів розділу, оскільки вона підтвердила цілісність не лише окремих функцій, а й зв'язків між ними у часі. Для сюжетно-орієнтованого технічного хорору це принципово: будь-яка невидима розбіжність між моментом збереження і моментом завантаження швидко руйнує логіку дослідження простору.

4.5 Тестування стабільності та продуктивності

Тестування стабільності та продуктивності виконувалося у сцені World з роздільною здатністю 1920×1080. Для збирання даних використовувалися два інструменти Unity. Перший — Profiler window, який містить модулі профілювання, елементи запису та панель деталізації вибраного кадру; профілювання в Play Mode є типовим режимом, а запис даних відбувається лише при ввімкненій кнопці Record. Другий — Rendering Statistics у Game view, що відкривається кнопкою Stats і в реальному часі показує показники рендерингу, зокрема FPS, CPU Main, Render, Batches, Tris, Verts і SetPass.

Для базової сесії профілювання було достатньо модулів CPU Usage, Rendering, Memory та UI (Canvas). Офіційна документація Unity окремо попереджає, що модуль GPU Usage має підвищений overhead і тому не є активним за замовчуванням; отже, в цій роботі він використовувався лише точково. Також важливо враховувати, що показник CPU Main у режимі редактора включає не лише час виконання ігрового кадру, а й частину редакторних витрат, тому для цієї КБР ключовим є саме порівняння до/після в однакових умовах, а не абсолютні числа без контексту.

Під час оптимізації сцени вже застосовувалася технологія LOD для частини 3D-об'єктів. Unity визначає Level of Detail як оптимізацію, що зменшує кількість трикутників, які потрібно рендерити для об'єкта зі збільшенням відстані до камери [10]. У межах проєкту це рішення використовувалося для скорочення геометричного навантаження у великих і візуально насичених частинах сцени. Крім того, об'єкти були згруповані за кімнатами та функціональними зонами. Така структура ще не означає, що окремий runtime-механізм room-based culling уже введено в експлуатацію, але вона підготовлена як основа для подальшого відсікання невидимих приміщень. Це узгоджується з документацією Unity, де зазначено, що occlusion culling найефективніше працює у сценах із невеликими чітко відокремленими зонами, типовим прикладом яких є кімнати, з'єднані коридорами.



Рис. 4.1 Показники продуктивності сцени World до оптимізації.

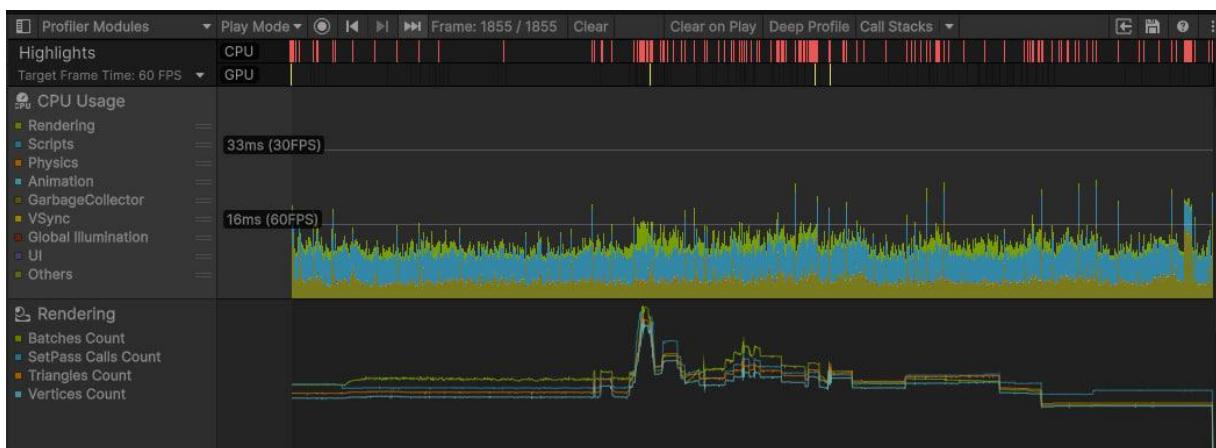


Рис. 4.2 Показники продуктивності сцени World після оптимізації.

Таблиця 4.3

Результати тестування продуктивності сцени World

Показник	До оптимізації	Після оптимізації	Зміна
Середній FPS	69	89	+29,0%
Мінімальний FPS	30–33	45–50	+36–50%
Середній час кадру, мс	14,5	11,2	–22,8%
Стабільність графіка CPU Usage	помітні часті піки	піки менш виражені	покращено
Графік Rendering	навантаження нерівномірне	навантаження стабільніше	покращено

За результатами профілювання сцени World встановлено, що після проведення оптимізації середній час формування кадру зменшився з 14,5 мс до 11,2 мс, що відповідає покращенню приблизно на 22,8%. Орієнтовний середній показник частоти кадрів зріс з 69 FPS до 89 FPS, тобто приблизно на 29,0%. Також зменшилися пікові просідання продуктивності: якщо до оптимізації окремі ділянки графіка наближалися до межі 30–33 мс, то після оптимізації основні просідання залишалися в межах приблизно 20–22 мс. Це свідчить про підвищення стабільності роботи сцени та зменшення навантаження під час проходження контрольного маршруту.

4.6 Тест-кейси та результати їх виконання

Нижче наведено компактний перелік тест-кейсів, які використовувалися для формального підтвердження коректності роботи основних модулів гри. Таблиця укладена у скороченому форматі, щоб уникнути перевантаження розділу другорядними деталями, але водночас відобразити всі контрольні сценарії, які реально впливали на якість продукту.

Таблиця 4.4

Функціональні тест-кейси та результати їх виконання

№	Перевірка	Очікуваний результат	Статус
ТС-01	Запуск сцени World	Сцена завантажується без критичних помилок	Пройдено
ТС-02	Переміщення персонажа та огляд простору	Персонаж рухається, камера реагує коректно	Пройдено
ТС-03	Наведення на інтерактивний об'єкт	Відображається підказка взаємодії	Пройдено
ТС-04	Підбір предмета зі сцени	Предмет додається до інвентарю	Пройдено
ТС-05	Відкриття інвентарю після підбору предмета	Підібраний предмет відображається в інвентарі	Пройдено
ТС-06	Взаємодія з об'єктом доступу	Об'єкт змінює стан відповідно до заданої умови	Пройдено
ТС-07	Увімкнення CCTV-монітора	Відкривається режим перегляду камер відеоспостереження	Пройдено
ТС-08	Перемикання камер і вихід із режиму CCTV через клавішу Q	Камери перемикаються, керування повертається до звичайного режиму	Пройдено
ТС-09	Відкриття меню паузи та інвентарю	UI-панелі не блокують одна одну після закриття	Пройдено
ТС-10	Збереження та завантаження гри	Стан гравця, інвентарю й основних об'єктів відновлюється	Пройдено

Продовження таблиці 4.4

ТС-11	Перевірка стану події поламки після завантаження	Стан події коректно відновлюється	Пройдено
ТС-12	Повторне коротке проходження після виправлень	Критичних помилок не виявлено	Пройдено

Після завершення циклу виправлень усі 12 тест-кейсів були виконані успішно. Отже, підсумковий показник проходження функціонального тестування становить 100%. Важливо підкреслити, що цей результат отримано не на первинній ітерації, а після усунення дефектів, зафіксованих під час сценарних і регресійних перевірок. Саме тому він є показником фактичної готовності поточної версії, а не лише стартового стану прототипу.

4.7 Оцінка відповідності програмного продукту поставленим вимогам

Повноцінна оцінка залучення та занурення гравців можлива лише після релізу, відкритого бета-тестування або ширшого користувацького дослідження. У межах цієї кваліфікаційної роботи було виконано попереднє внутрішнє оцінювання за участю 6 тестувальників HitekGames. Кожен учасник проходив однаковий тестовий фрагмент тривалістю 10–15 хвилин, після чого оцінював гру за шкалою 1–5. Метою етапу було отримати первинний зворотний зв'язок не лише про працездатність системи, а й про те, наскільки зрозумілою, атмосферною та технічно стабільною сприймається поточна версія SITE-108.

Таблиця 4.5

Попереднє внутрішнє оцінювання ігрового досвіду

Параметр оцінювання	Рання версія	Поточна версія	Зміна
Зрозумілість взаємодії	3,2	4,4	+37,5%
Зручність інтерфейсу	3,1	4,2	+35,5%
Атмосферність середовища	3,5	4,5	+28,6%
Відчуття занурення	3,3	4,3	+30,3%
Зацікавленість у подальшому проходженні	3,4	4,4	+29,4%
Технічна стабільність	3,0	4,3	+43,3%
Середнє значення	3,25	4,35	+33,8%

Результати таблиці показують позитивну динаміку за всіма параметрами оцінювання. Найбільший приріст зафіксовано за показником технічної стабільності (+43,3%), що корелює з виправленням сценарних дефектів та покращенням продуктивності сцени. Зростання показників зрозумілості взаємодії й зручності інтерфейсу також підтверджує, що контекстні підказки, коректна робота інвентарю та очищення UI-станів покращили сприйняття гри без спрощення її атмосфери.

Узагальнений зворотний зв'язок тестувальників можна сформулювати так. По-перше, контекстні підказки та логіка доступів роблять взаємодію зі світом зрозумілішою і зменшують кількість ситуацій, коли гравець не розуміє, з чим можна взаємодіяти. По-друге, технічне середовище бункерного типу сприймається цілісно, а CCTV-механіка додає напруги та підсилює відчуття контролю над простором. По-третє, прогресія через предмети, об'єкти середовища та технічні системи оцінюється як логічна. Серед побажань згадувалося подальше розширення сценаріїв поламок і глибше балансування темпу проходження в окремих ділянках сцени.

Підсумкову відповідність реалізованого продукту поставленим вимогам наведено в таблиці 4.6.

Таблиця 4.6

Відповідність програмного продукту поставленим вимогам

Вимога	Спосіб перевірки	Результат
Керування персонажем від першої особи	ТС-02	Виконано
Огляд 3D-простору та переміщення сценою	ТС-02	Виконано
Взаємодія з інтерактивними об'єктами середовища	ТС-03, ТС-06	Виконано
Підбір предметів зі сцени	ТС-04	Виконано
Робота інвентарю	ТС-05	Виконано
Використання предметів доступу	ТС-06	Виконано
Перегляд камер відеоспостереження	ТС-07, ТС-08	Виконано
Робота основних UI-панелей	ТС-09	Виконано
Збереження та завантаження ігрового прогресу	ТС-10, ТС-11	Виконано
Відновлення стану подій після завантаження	ТС-11	Виконано
Стабільність короткого проходження після виправлень	ТС-12	Виконано
Покращення продуктивності сцени	Unity Profiler	Покращено

Отже, результати розділу дозволяють зробити висновок, що реалізована версія SITE-108 відповідає основним технічним і функціональним вимогам, поставленим у роботі. Після циклу виправлень досягнуто 100% проходження функціональних тест-кейсів, середній FPS на контрольному маршруті зріс на +29,0%, а середній показник внутрішнього оцінювання ігрового досвіду збільшився на 33,8%. У сукупності це свідчить про те, що програмний продукт не лише працює коректно в межах протестованого фрагмента, а й рухається в

напрямі поставленої мети — створення більш стабільного, зрозумілого та атмосферного ігрового досвіду.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було досягнуто поставлену мету — підвищення рівня залучення та занурення гравця в ігровий процес за рахунок розробки 3D відеогри в середовищі Unity з використанням механік, характерних для жанрів хорору та RPG. Результатом роботи став прототип ігрового застосунку SITE-108, у якому поєднано атмосферне 3D-середовище, дослідження простору, взаємодію з об'єктами, інвентар, систему доступів, технічні несправності, моніторинг і збереження прогресу.

У першому розділі було проаналізовано предметну галузь розробки 3D відеоігор з елементами хорору та RPG. Розглянуто особливості формування ігрового досвіду, залучення та занурення гравця, а також досліджено ігри-аналоги: Resident Evil 7: Biohazard, Amnesia: The Bunker та Outlast. На основі аналізу визначено, що для власного проєкту доцільно використати атмосферність середовища, обмежений контроль гравця, поступове відкриття простору, технічні системи як джерело напруги та RPG-прогресію через розширення можливостей взаємодії з ігровим середовищем.

У другому розділі було сформовано функціональні та нефункціональні вимоги до ігрового застосунку, спроектовано архітектуру програмного продукту, основні функціональні модулі, ігровий цикл, структуру рівня та інтерфейс користувача. Проєктування дозволило визначити логіку взаємодії між ключовими системами гри: керуванням персонажем, системою взаємодії, інвентарем, доступами, технічними об'єктами, моніторингом, UI та збереженням прогресу.

У третьому розділі було реалізовано основні програмні модулі SITE-108 у середовищі Unity з використанням мови C#. Реалізовано керування персонажем від першої особи, взаємодію з інтерактивними об'єктами, підбір і використання предметів, інвентар, систему доступу до об'єктів, елементи технічних несправностей, камери відеоспостереження, UI-панелі та систему збереження і

завантаження прогресу. Також сформовано ігрове 3D-середовище, яке підтримує атмосферу ізоляції, напруги та поступового дослідження простору.

У четвертому розділі було проведено тестування функціональності, сценаріїв проходження, інтерфейсу, системи збереження, стабільності та продуктивності ігрового застосунку. Для перевірки основних механік було використано 12 функціональних тест-кейсів, усі з яких після циклу виправлень були виконані успішно. Підсумковий показник проходження функціонального тестування становить 100%, що підтверджує коректність роботи основних модулів поточної версії прототипу.

Окремо було проведено профілювання продуктивності сцени World. За результатами оптимізації середній показник частоти кадрів зріс з 69 FPS до 89 FPS, тобто приблизно на 29,0%. Середній час формування кадру зменшився з 14,5 мс до 11,2 мс, що відповідає покращенню приблизно на 22,8%. Мінімальний FPS під час проходження контрольного маршруту зріс з діапазону 30–33 FPS до 45–50 FPS, що свідчить про зменшення просідань продуктивності та підвищення стабільності роботи сцени.

Для попередньої оцінки ігрового досвіду було проведено внутрішнє тестування за участю 6 тестувальників HitekGames. Кожен учасник проходив однаковий тестовий фрагмент тривалістю 10–15 хвилин та оцінював гру за шкалою від 1 до 5. Середній показник оцінювання зріс з 3,25 у ранній версії до 4,35 у поточній версії, що відповідає покращенню на 33,8%. Найбільше зростання зафіксовано за параметром технічної стабільності — з 3,0 до 4,3, тобто на 43,3%. Оцінка відчуття занурення зросла з 3,3 до 4,3, що становить покращення на 30,3%, а атмосферність середовища — з 3,5 до 4,5, тобто на 28,6%.

Таким чином, результати кваліфікаційної роботи підтверджують доцільність обраного підходу до розробки 3D відеогри з елементами хорору та RPG. Розроблений прототип SITE-108 відповідає основним поставленим

вимогам, демонструє стабільну роботу ключових ігрових механік, має підтверджені результати функціонального тестування та позитивну динаміку внутрішнього оцінювання ігрового досвіду. Практична значущість роботи полягає в тому, що реалізований програмний продукт може бути використаний як основа для подальшого розвитку повноцінної сюжетно-орієнтованої 3D гри з розширенням рівнів, подій, механік взаємодії, системи прогресії та засобів формування атмосфери.

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

1. Ядлось Д.О., Треньов М.Г. Розробка 3D відеогри на Unity з елементами хорору та RPG. Матеріали II Всеукраїнської науково-технічної конференції «Виклики та рішення в програмній інженерії», Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. Київ: ДУІКТ, 2026. С. 421-422.

2. Ядлось Д.О., Треньов М.Г. Захист ігрових даних у 3D-відеогрі «SITE-108». Матеріали Всеукраїнської науково-практичної конференції «Цифрова трансформація кібербезпеки», 29 квітня 2026 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. Київ : ДУІКТ, 2026. С. 98-99.

ПЕРЕЛІК ПОСИЛАНЬ

1. Newzoo's Global Games Market Report 2025. Newzoo. 2025. URL: <https://newzoo.com/resources/trend-reports/newzoo-global-games-market-report-2025> (дата звернення: 15.05.2026).
2. Immersive Horror Games Market – Global Forecast 2026–2032. Research and Markets. URL: <https://www.researchandmarkets.com/reports/6055423/immersive-horror-games-market-global-forecast> (дата звернення: 15.05.2026).
3. Thon J.-N. Playing with Fear: The Aesthetics of Horror in Recent Indie Games. *Eludamos: Journal for Computer Game Culture*. 2019. Vol. 10, No. 1. P. 197–231. DOI: <https://doi.org/10.7557/23.6179>.
4. Sweetser P., Wyeth P. GameFlow: A Model for Evaluating Player Enjoyment in Games. *Computers in Entertainment*. 2005. Vol. 3, No. 3. P. 1–24. DOI: <https://doi.org/10.1145/1077246.1077253>.
5. Brown E., Cairns P. A Grounded Investigation of Game Immersion. *CHI 2004 Extended Abstracts on Human Factors in Computing Systems*. New York: ACM Press, 2004. P. 1297–1300. DOI: <https://doi.org/10.1145/985921.986048>.
6. Hunicke R., LeBlanc M., Zubek R. MDA: A Formal Approach to Game Design and Game Research. *Proceedings of the AAAI Workshop on Challenges in Game AI*. 2004. Vol. 4, No. 1. URL: <https://www.cs.northwestern.edu/~hunicke/MDA.pdf> (дата звернення: 15.05.2026).
7. Resident Evil 7 biohazard. CAPCOM. URL: <https://www.residentevil.com/7/> (дата звернення: 15.05.2026).
8. Amnesia: The Bunker. Frictional Games. URL: <https://amnesiathebunker.com/> (дата звернення: 15.05.2026).
9. Outlast. Red Barrels. URL: <https://redbarrelsgames.com/games/outlast/> (дата звернення: 15.05.2026).
10. Unity User Manual. Unity Technologies. URL: <https://docs.unity3d.com/Manual/> (дата звернення: 15.05.2026).
11. C# Guide. Microsoft Learn. URL: <https://learn.microsoft.com/en-us/dotnet/csharp/>

(дата звернення: 15.05.2026).

- 12.Unity Scripting API. Unity Technologies. URL: <https://docs.unity3d.com/ScriptReference/> (дата звернення: 15.05.2026).
- 13.High Definition Render Pipeline overview. Unity Documentation. Unity Technologies. URL: <https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@latest/> (дата звернення: 15.05.2026).
- 14.Documentation for Visual Studio Code. Microsoft. URL: <https://code.visualstudio.com/docs> (дата звернення: 15.05.2026).
- 15.About Version Control. Git Documentation. URL: <https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control> (дата звернення: 15.05.2026).
- 16.About repositories. GitHub Docs. URL: <https://docs.github.com/en/repositories/creating-and-managing-repositories/about-repositories> (дата звернення: 15.05.2026).
- 17.Asset Store packages. Unity User Manual. Unity Technologies. URL: <https://docs.unity3d.com/Manual/AssetStorePackages.html> (дата звернення: 15.05.2026).
- 18.ISO/IEC/IEEE 29148:2018. Systems and software engineering — Life cycle processes — Requirements engineering. Geneva: International Organization for Standardization, 2018.
- 19.ISO/IEC 25010:2023. Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Product quality model. Geneva: International Organization for Standardization, 2023.
- 20.ISO 9241-210:2019. Ergonomics of human-system interaction — Part 210: Human-centred design for interactive systems. Geneva: International Organization for Standardization, 2019.
- 21.Json.NET Documentation. Newtonsoft. URL: <https://www.newtonsoft.com/json/help/html/Introduction.htm> (дата звернення: 15.05.2026).

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ



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



Розробка 3D відеогри на Unity з елементами хорору та RPG

Виконав студент 4 курсу
групи ПД-41
Ядлось Даніл Олександрович
Керівник роботи
асистент кафедри ІПЗ Треньов Микита Георгійович

Київ – 2026

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: підвищення рівня залучення та занурення гравця в ігровий процес за рахунок розробки програмного продукту в середовищі Unity з використанням механік, характерних для жанрів хорору та RPG.

Об'єкт дослідження: процеси проєктування та розробки 3D відеогри з елементами хорору та RPG.

Предмет дослідження: засоби, методи та технології реалізації ігрових механік, атмосферних рішень і системи прогресії в середовищі Unity.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Проаналізувати предметну галузь розробки 3D відеоігор з елементами хорору та RPG.
2. Дослідити підходи до формування ігрового досвіду, залучення та занурення гравця.
3. Провести аналіз ігор-аналогів та визначити вимоги до розроблюваного ігрового застосунку.
4. Проаналізувати засоби розробки 3D відеоігри та обґрунтувати вибір технологічного стеку.
5. Сформулювати функціональні та нефункціональні вимоги до програмного продукту.
6. Підготувати основні проєктні артефакти гри: GDD, структуру рівня, опис ігрового циклу та специфікацію основних систем.
7. Спроектувати архітектуру програмного продукту, діаграму варіантів використання та діаграму класів.
8. Реалізувати основні ігрові механіки, інтерфейс користувача, систему взаємодії з об'єктами та збереження ігрового прогресу.
9. Підготувати тест-план і провести тестування функціональності, стабільності та продуктивності ігрового застосунку.

3

АНАЛІЗ АНАЛОГІВ

	Resident Evil 7	Amnesia: The Bunker	Outlast	SITE-108
Джерело напруги	Атмосфера, ресурси, вороги	Генератор, світло, монстр	Втеча, темрява, камера	Збої систем, аномалії
Вразливість	Помірна: є зброя	Висока: захист обмежений	Дуже висока: втеча	Висока: контроль обмежений
Роль середовища	Головоломки, ресурси	Система виживання	Втеча й ховання	Ремонт, доступ, моніторинг
Прогресія	Ключі, нові зони	Інструменти, маршрути	Лінійне проходження	Рівні, системи, інструменти

4

КОНЦЕПТ ГРИ

- Сетінг гри побудовано навколо ізольованого урядового бункера SITE-108, у якому після катастрофи на третьому рівні головний герой залишається підтримувати роботу критичних систем об'єкта.
- Сюжет розкриває протистояння між автономною системою керування бункером та сутністю з третього рівня, які по-різному маніпулюють гравцем і подають суперечливу інформацію про події на об'єкті.
- Ключові механіки гри включають дослідження середовища, взаємодію з об'єктами, використання інвентарю, усунення технічних несправностей, відкриття нових зон та роботу з інформацією.
- Хорор-складова формується через ізоляцію, невизначеність, технічні події, обмежений контроль гравця та поступове розкриття наслідків катастрофи.
- RPG-елементи проявляються через прогресію доступу, накопичення інформації, розвиток можливостей гравця та вплив прийнятих рішень на фінал гри.
- USP проекту полягає в поєднанні атмосферного хорору, технічного виживання в бункері та сюжетної моделі “двох оман”, де гравець має самостійно визначити, кому довіряти.
- Концепт гри спрямований на те, щоб поєднати страх, дослідження середовища та поступову залученість гравця в єдину систему проходження.

5

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги	Нефункціональні вимоги
1.Керування персонажем у 3D-середовищі з камерою від першої особи.	1.Продуктивність — не менше 30 FPS.
2.Взаємодія з дверима, технічними панелями, предметами та тригерами подій.	2.Реакція інтерактивних об'єктів — із затримкою не більше 0,2 с.
3.Інвентар для підбору, зберігання та використання предметів.	3.Відсутність фрізів і лаг-спайків понад 1 с.
4.Прогресія через відкриття нових зон, інструментів або можливостей.	4.Основні дії інтерфейсу мають виконуватися не більше ніж за 2–3 дії користувача.
5.Ігрові події, що змінюють стан локації або створюють загрозу.	5.Екранні форми (за винятком інвентарю) мають містити до 7–9 інтерактивних елементів, без перевантаження інтерфейсу.
6.Збереження та завантаження ігрового прогресу.	6.UI-підказки не перекривають важливі елементи сцени.

Мінімальні системні вимоги: Windows 10/11 64-bit, Intel Core i5 / Ryzen 5, 8 GB RAM, GPU з підтримкою DirectX 12, 5 GB вільного місця.

Рекомендовані системні вимоги: Windows 10/11 64-bit, Intel Core i5 10-го покоління / Ryzen 5 3600 або вище, 16 GB RAM, GTX 1660 Super/RTX 2060 або вище, SSD, 5 GB вільного місця.

6

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

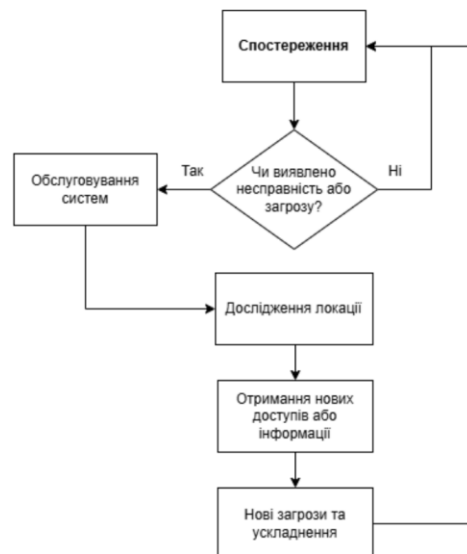


Visual Studio Code



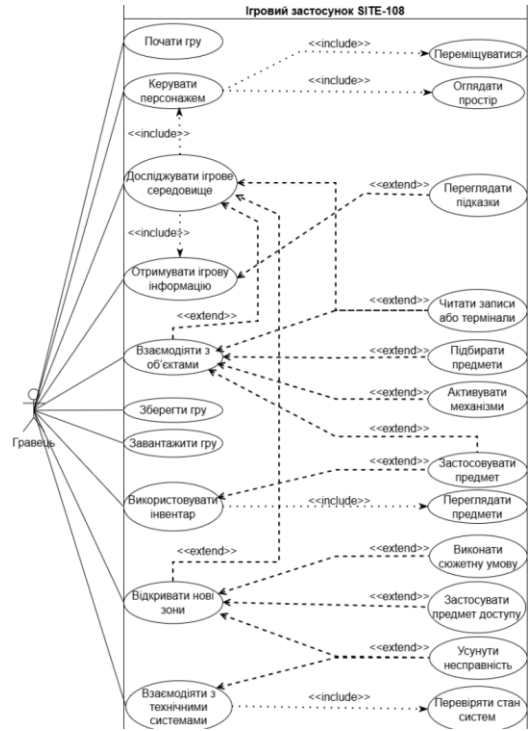
7

ІГРОВИЙ ЦИКЛ



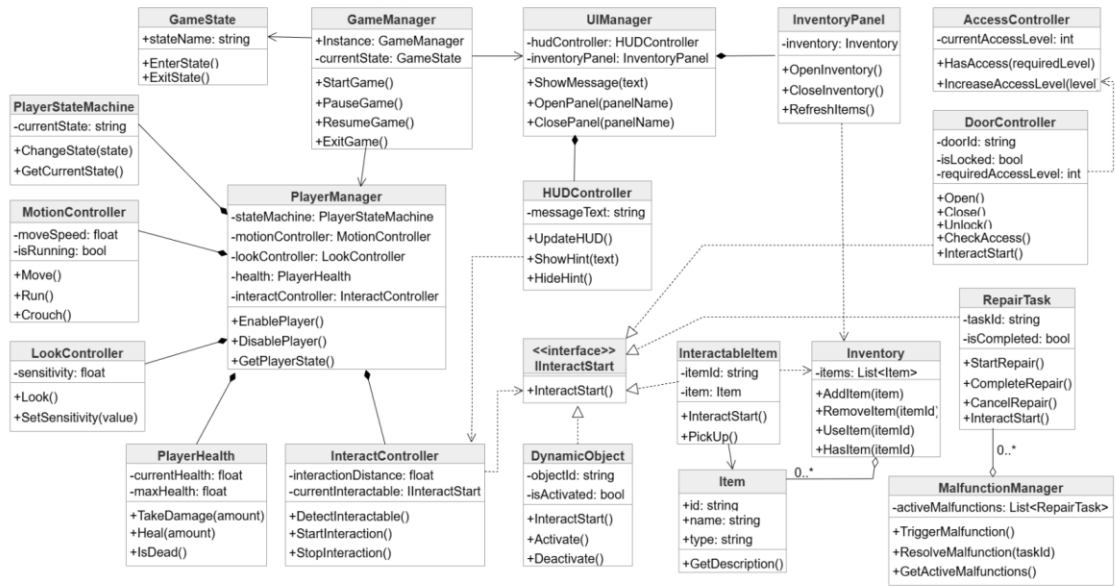
8

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ



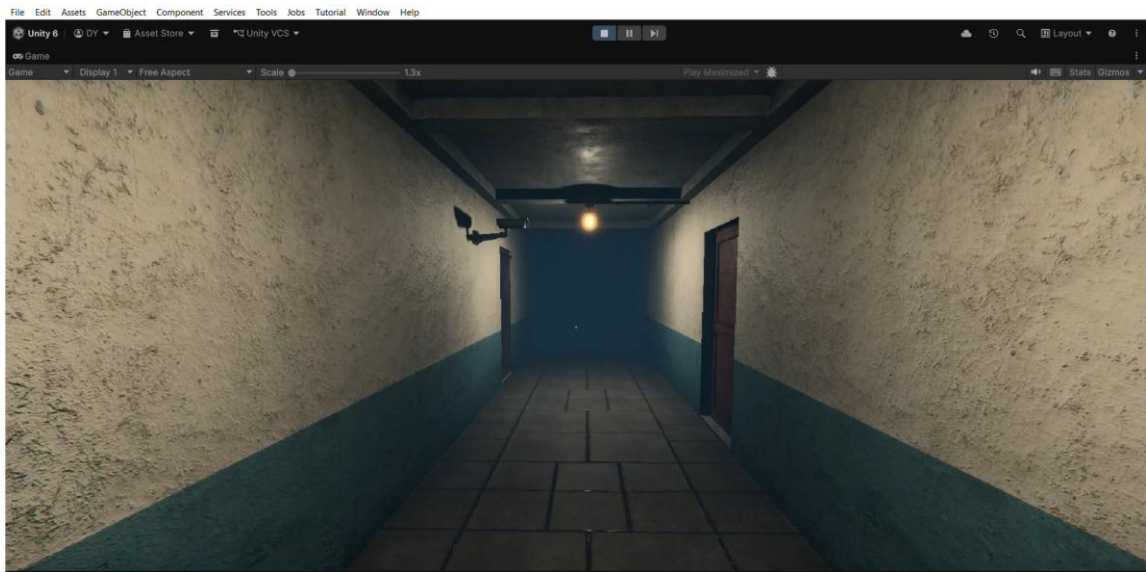
8

ДІАГРАМА КЛАСІВ



9

ЕКРАННІ ФОРМИ



Частина геймплею

10

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

Тези доповідей:

1. Ядлось Д.О., Треньов М.Г. Розробка 3D відеогри на Unity з елементами хорору та RPG. Матеріали II Всеукраїнської науково-технічної конференції «Виклики та рішення в програмній інженерії», Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. Київ: ДУІКТ, 2026. С. 421-422.
2. Ядлось Д.О., Треньов М.Г. Захист ігрових даних у 3D-відеогрі «SITE-108». Матеріали Всеукраїнської науково-практичної конференції «Цифрова трансформація кібербезпеки», 29 квітня 2026 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. Київ : ДУІКТ, 2026. С. 98-99.

11

ВИСНОВКИ

1. Проаналізовано предметну галузь розробки 3D відеоігор з елементами хорору та RPG, що дозволило визначити основні особливості створення ігрового продукту, орієнтованого на атмосферність, інтерактивність та поступове залучення гравця.
2. Досліджено підходи до формування ігрового досвіду, залучення та занурення гравця, у результаті чого обґрунтовано доцільність поєднання хорор-складової з RPG-підходом у межах одного ігрового застосунку.
3. Проведено аналіз ігор-аналогів, що дозволило визначити сильні сторони існуючих рішень, їх слабкості та сформувати напрям розробки SITE-108 як гри з активною роллю середовища та поступовим розкриттям ігрового простору.
4. Проаналізовано засоби розробки 3D відеогри та обґрунтовано вибір технологічного стеку, до якого увійшли Unity, C#, Unity Scripting API, HDRP, Visual Studio Code, Git, GitHub та візуальні asset-ресурси.
5. Сформовано функціональні та нефункціональні вимоги до програмного продукту, які стали основою для подальшого проєктування і реалізації ключових систем гри.
6. Розроблено основні проєктні артефакти гри, що дозволило узгодити геймдизайнерську частину розробки з технічною структурою програмного продукту.
7. Спроектовано архітектуру програмного продукту, діаграму варіантів використання та діаграму класів, що дозволило формалізувати сценарії взаємодії гравця з системою та визначити зв'язки між основними програмними компонентами.
8. Реалізовано функціональну версію 3D відеогри за допомогою обраного стеку технологій, у якій основні механіки об'єднано в єдиний ігровий цикл, спрямований на дослідження середовища, взаємодію з об'єктами та поступове просування гравця.
9. Протестовано ігровий застосунок за тест-планом, що дозволило системно перевірити функціональність, стабільність, продуктивність, коректність основних екранних форм та взаємодію гравця з ключовими механіками гри.

ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ ПРОГРАМНИХ МОДУЛІВ

```

using UnityEngine;
using UnityEngine.Events;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;
using System;
using System.IO;
using System.Linq;
using System.Collections;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace SITE108.Runtime
{
    public enum DynamicSoundType { Open, Close, Locked, Unlock
    }

    public class DynamicObject : MonoBehaviour,
    IInteractStartPlayer, IInteractHold, IInteractStop, ISaveable
    {
        public enum DynamicType { Openable, Pullable, Switchable,
        Rotable }
        public enum TransformType { Local, Global }
        public enum InteractType { Dynamic, Mouse, Animation }
        public enum DynamicStatus { Normal, Locked }
        public enum StatusChange { InventoryItem, CustomScript,
        None }

        public DynamicType dynamicType = DynamicType.Openable;
        public TransformType transformType = TransformType.Local;
        public DynamicStatus dynamicStatus = DynamicStatus.Normal;
        public InteractType interactType = InteractType.Dynamic;
        public StatusChange statusChange =
        StatusChange.InventoryItem;

        public Transform target;
        public AudioSource audioSource;
        public Animator animator;
        public HingeJoint joint;
        public new Rigidbody rigidbody;
        public Inventory inventory;
        public GameManager gameManager;

        public MonoBehaviour unlockScript;
        public bool keepUnlockItem;
        public ItemGuid unlockItem;
        public bool showLockedText;
        public GString lockedText;

        public string useTrigger1 = "Open";
        public string useTrigger2 = "Close";
        public string useTrigger3 = "OpenSide";

        public DynamicOpenable openable = new DynamicOpenable();
        public DynamicPullable pullable = new DynamicPullable();
        public DynamicSwitchable switchable = new
        DynamicSwitchable();
        public DynamicRotable rotatable = new DynamicRotable();

        public AudioClip useSound1;
        public AudioClip useSound2;
        public AudioClip lockedSound;
        public AudioClip unlockSound;

        public UnityEvent useEvent1;
        public UnityEvent useEvent2;
        public UnityEvent<float> onValueChange;
        public UnityEvent lockedEvent;
        public UnityEvent unlockedEvent;

        public bool lockPlayer;
        public bool isLocked;
        public bool isInteractLocked;

        public DynamicObjectType CurrentDynamic
        {
            get => dynamicType switch
            {
                DynamicType.Openable => openable,
                DynamicType.Pullable => pullable,
                DynamicType.Switchable => switchable,
                DynamicType.Rotable => rotatable,
                _ => null,
            };
        }

        public bool IsOpened => CurrentDynamic.IsOpened;
        public bool IsHolding => CurrentDynamic.IsHolding;

        private void OnValidate()
        {
            openable.DynamicObject = this;
            pullable.DynamicObject = this;
            switchable.DynamicObject = this;
            rotatable.DynamicObject = this;
        }

        private void Awake()
        {
            inventory = Inventory.Instance;
            gameManager = GameManager.Instance;

            if (dynamicStatus == DynamicStatus.Locked)
                isLocked = true;

            CurrentDynamic?.OnDynamicInit();
        }

        private void Update()
        {
            if (!isLocked)
                CurrentDynamic?.OnDynamicUpdate();
        }

        public void InteractStartPlayer(GameObject player)
        {
            if (isInteractLocked) return;
            PlayerManager playerManager =
            player.GetComponent<PlayerManager>();
            CurrentDynamic?.OnDynamicStart(playerManager);
        }

        public void InteractHold(Vector3 point)
        {
            Vector2 delta =
            InputManager.ReadInput<Vector2>(Controls.POINTER_DELTA);
            if (!isLocked)
                CurrentDynamic?.OnDynamicHold(delta);
        }

        public void InteractStop()
        {
            if (!isLocked)
                CurrentDynamic?.OnDynamicEnd();
        }

        public void SetLockedStatus(bool locked)
        {
            isLocked = locked;
            if (!locked) isInteractLocked = false;
        }

        public void SetOpenState()
        {
            if (interactType == InteractType.Mouse || isLocked) return;
            CurrentDynamic?.OnDynamicOpen();
        }

        public void SetCloseState()
        {
            if (interactType == InteractType.Mouse || isLocked) return;

```

```

        CurrentDynamic?.OnDynamicClose();
    }

    public void TryUnlockResult(bool unlocked)
    {
        if (unlocked)
        {
            unlockedEvent?.Invoke();
            PlaySound(DynamicSoundType.Unlock);
        }
        else
        {
            lockedEvent?.Invoke();
            PlaySound(DynamicSoundType.Locked);
        }
    }

    public void PlaySound(DynamicSoundType soundType)
    {
        switch (soundType)
        {
            case DynamicSoundType.Open:
                GameTools.PlayOneShot3D(target.position, useSound1,
"Open Sound");
                break;
            case DynamicSoundType.Close:
                GameTools.PlayOneShot3D(target.position, useSound2,
"Close Sound");
                break;
            case DynamicSoundType.Locked:
                GameTools.PlayOneShot3D(target.position,
lockedSound, "Locked Sound");
                break;
            case DynamicSoundType.Unlock:
                GameTools.PlayOneShot3D(target.position,
unlockSound, "Unlock Sound");
                break;
        }
    }

    public StorableCollection OnSave()
    {
        StorableCollection saveableBuffer = new
StorableCollection();

        switch (dynamicType)
        {
            case DynamicType.Openable:
                openable.OnSave(ref saveableBuffer);
                break;
            case DynamicType.Pullable:
                pullable.OnSave(ref saveableBuffer);
                break;
            case DynamicType.Switchable:
                switchable.OnSave(ref saveableBuffer);
                break;
            case DynamicType.Rotable:
                rotatable.OnSave(ref saveableBuffer);
                break;
        }

        saveableBuffer.Add("isLocked", isLocked);
        return saveableBuffer;
    }

    public void OnLoad(JToken data)
    {
        switch (dynamicType)
        {
            case DynamicType.Openable:
                openable.OnLoad(data);
                break;
            case DynamicType.Pullable:
                pullable.OnLoad(data);
                break;
            case DynamicType.Switchable:
                switchable.OnLoad(data);
                break;
        }
    }

```

```

        case DynamicType.Rotable:
            rotatable.OnLoad(data);
            break;
    }

    isLocked = (bool)data["isLocked"];
}

public partial class Inventory : Singleton<Inventory>,
ISaveableCustom
{
    public enum SlotType { Locked, Inventory, Container }

    public Dictionary<string, Item> items = new();
    public Dictionary<InventoryItem, InventorySlot[]>
carryingItems = new();
    public Dictionary<InventoryItem, InventorySlot[]>
containerItems = new();
    public ItemContainer currentContainer;
    public InventorySlot[] slots;
    public InventorySlot[] containerSlots;
    public ShortcutSlot[] shortcuts;
    public Transform itemsTransform;
    public GameManager gameManager;
    public bool itemSelector;
    public IInventorySelector inventorySelector;
    public int expandedSlots;

    public Subject<ItemUpdate> OnItemAdded = new();
    public Subject<ItemUpdate> OnItemRemoved = new();

    public bool AddItem(string guid, ushort quantity,
ItemCustomData customData)
    {
        return AddItem(guid, quantity, customData, out _);
    }

    public bool AddItem(string guid, ushort quantity,
ItemCustomData customData, out InventoryItem inventoryItem)
    {
        inventoryItem = null;
        int availableQ = quantity;

        if (!items.ContainsKey(guid))
            return false;

        Item item = items[guid];
        ushort maxStack = item.Properties.maxStack;

        if (item.Settings.isStackable && ContainsItemMany(guid,
out OccupyData[] itemDatas))
        {
            itemDatas = itemDatas.OrderByDescending(x =>
            {
                int itemQuantity = x.inventoryItem.Quantity;
                return maxStack - itemQuantity;
            }).ToArray();

            foreach (var itemData in itemDatas)
            {
                int currQuantity = itemData.inventoryItem.Quantity;
                inventoryItem = itemData.inventoryItem;

                if (maxStack == 0)
                {
                    currQuantity += availableQ;
                    inventoryItem.SetQuantity(currQuantity);
                    OnItemAdded.OnNext(new(guid, quantity));
                    return true;
                }

                int nextQuantity = currQuantity + availableQ;
                if (nextQuantity <= maxStack)
                {
                    inventoryItem.SetQuantity(nextQuantity);
                    OnItemAdded.OnNext(new(guid, quantity));
                    return true;
                }
            }
        }
    }

```

```

    }

    availableQ = nextQuantity - maxStack;
    inventoryItem.SetQuantity(maxStack);
}

FreeSpace freeSpace = FindFreeSpace(item, slots);
if (!freeSpace.HasFreeSpace)
    return false;

inventoryItem = CreateItem(new ItemCreationData()
{
    itemGuid = guid,
    quantity = (ushort)availableQ,
    orientation = freeSpace.Orientation,
    coords = freeSpace.Coords,
    customData = customData,
    parent = itemsTransform,
    slotsSpace = slots
});

AddItemToFreeSpace(freeSpace, inventoryItem);
OnItemAdded.OnNext(new(guid, quantity));
return true;
}

public bool RemoveItem(string guid)
{
    if (ContainsItem(guid, out OccupyData itemData))
    {
        int quantity = itemData.inventoryItem.Quantity;
        carryingItems.Remove(itemData.inventoryItem);
        Destroy(itemData.inventoryItem.gameObject);

        foreach (var slot in itemData.occupiedSlots)
            slot.itemInSlot = null;

        OnItemRemoved.OnNext(new(guid, quantity));
        return true;
    }

    return false;
}

public int RemoveItem(string guid, ushort quantity)
{
    if (!ContainsItem(guid, out OccupyData itemData))
        return 0;

    if ((itemData.inventoryItem.Quantity - quantity) >= 1)
    {
        int q = itemData.inventoryItem.Quantity - quantity;
        itemData.inventoryItem.SetQuantity(q);
        OnItemRemoved.OnNext(new(guid, quantity));
        return q;
    }

    carryingItems.Remove(itemData.inventoryItem);
    Destroy(itemData.inventoryItem.gameObject);

    foreach (var slot in itemData.occupiedSlots)
        slot.itemInSlot = null;

    OnItemRemoved.OnNext(new(guid, quantity));
    return 0;
}

public void OpenItemSelector(IInventorySelector
inventorySelector)
{
    itemSelector = true;
    this.inventorySelector = inventorySelector;
    gameManager.ShowInventoryPanel(true);
}

public StorableCollection OnCustomSave()
{

```

```

    StorableCollection saveableBuffer = new
StorableCollection();
    StorableCollection itemsToSave = new StorableCollection();
    StorableCollection shortcutsSave = new StorableCollection();

    int index = 0;
    foreach (var item in carryingItems)
    {
        itemsToSave.Add("item_" + index++, new
StorableCollection()
        {
            { "item", item.Key.ItemGuid },
            { "quantity", item.Key.Quantity },
            { "orientation", item.Key.orientation },
            { "position", item.Key.Position.ToSaveable() },
            { "customData", item.Key.CustomData?.ToJson() },
        });
    }

    for (int i = 0; i < shortcuts.Length; i++)
    {
        shortcutsSave.Add("shortcut_" + i, shortcuts[i].item != null
? shortcuts[i].item.ItemGuid : "{}");
    }

    saveableBuffer.Add("expanded", expandedSlots);
    saveableBuffer.Add("items", itemsToSave);
    saveableBuffer.Add("shortcuts", shortcutsSave);
    return saveableBuffer;
}

public void OnCustomLoad(JToken data)
{
    int expandedRowCount = (int)data["expanded"];
    if (expandedRowCount > 0)
        ExpandInventory(expandedRowCount, false);

    JObject items = (JObject)data["items"];
    foreach (var itemProp in items.Properties())
    {
        JToken token = itemProp.Value;
        string itemGuid = token["item"].ToString();
        int quantity = (int)token["quantity"];
        Orientation orientation =
(Orientation)(int)token["orientation"];
        Vector2Int position =
token["position"].ToObject<Vector2Int>();

        InventoryItem inventoryItem = CreateItem(new
ItemCreationData()
        {
            itemGuid = itemGuid,
            quantity = (ushort)quantity,
            orientation = orientation,
            coords = position,
            customData = new ItemCustomData() { JsonData =
token["customData"].ToString() },
            parent = itemsTransform,
            slotsSpace = slots
        });

        AddItemToFreeSpace(new FreeSpace()
        {
            Coords = position,
            Orientation = orientation,
            Space = GetSlotsSpace(position, inventoryItem.Item)
        }, inventoryItem);
    }
}

public class SaveGameManager : Singleton<SaveGameManager>
{
    public enum LoadType { Normal, LoadGameState,
LoadWorldState, LoadWorldLast, LoadPlayer }
    public enum SaveType { Normal, Autosave, NextScene }

    public static LoadType GameLoadType = LoadType.Normal;

```

```

public static string LoadSceneName;
public static string LoadFolderName;

public event Action<string> OnGameSaved;
public event Action OnGameLoaded;

public List<SaveablePair> worldSaveables = new();
public List<RuntimeSaveable> runtimeSaveables = new();

private SaveType GameSaveType = SaveType.Normal;
private PlayerPresenceManager playerPresence;
private ObjectiveManager objectiveManager;
private Inventory inventory;
private bool timerActive;
private static float TimePlayed;

private void Awake()
{
    playerPresence =
GetComponent<PlayerPresenceManager>();
    objectiveManager = GetComponent<ObjectiveManager>();
    inventory = GetComponent<Inventory>();
}

private void Update()
{
    if (timerActive)
        TimePlayed += Time.deltaTime;
}

public static void ClearLoadType()
{
    GameLoadType = LoadType.Normal;
    PlayerData = null;
    GameState = null;
}

public static void SetLoadGameState(string sceneName, string
folderName)
{
    GameLoadType = LoadType.LoadGameState;
    LoadSceneName = sceneName;
    LoadFolderName = folderName;
}

public static void SavePlayer()
{
    Instance.StartCoroutine(Instance.SavePlayerData());
}

public static void SaveGame(bool autosave)
{
    Instance.GameSaveType = autosave ? SaveType.Autosave :
SaveType.Normal;
    Instance.StartCoroutine(Instance.SaveGameData(null));
}

public static void SaveGame(Action onSaved)
{
    Instance.GameSaveType = SaveType.Normal;
    Instance.StartCoroutine(Instance.SaveGameData(onSaved));
}

private IEnumerator SaveGameData(Action onSaved)
{
    StorableCollection saveablesBuffer = new
StorableCollection();
    StorableCollection saveInfoData = new StorableCollection();

    foreach (var pair in worldSaveables)
    {
        if (pair.Instance is ISaveable saveable)
            saveablesBuffer.Add(pair.Token, saveable.OnSave());
    }

    foreach (var runtime in runtimeSaveables)
    {
        StorableCollection runtimeBuffer = new
StorableCollection();
        foreach (var pair in runtime.SaveablePairs)
        {
            if (pair.Instance is ISaveable saveable)
                runtimeBuffer.Add(pair.Token, saveable.OnSave());
        }
        saveablesBuffer.Add(runtime.TokenGUID,
runtimeBuffer);
    }

    saveablesBuffer.Add("inventory", (inventory as
ISaveableCustom).OnCustomSave());
    saveablesBuffer.Add("objectives", (objectiveManager as
ISaveableCustom).OnCustomSave());

    string saveFolderName =
DateTime.Now.ToString("yyyyMMddHHmmss");
    string saveFolderPath = Path.Combine(SavedGamePath,
saveFolderName);
    Task serializeTask = SerializeGameState(saveInfoData,
saveablesBuffer, saveFolderName, saveFolderPath);

    yield return new WaitUntil(() => serializeTask.IsCompleted);

    OnGameSaved?.Invoke(saveFolderName);
    onSaved?.Invoke();
}

private void LoadGameState(JObject gameState)
{
    foreach (var pair in worldSaveables)
    {
        if (pair.Instance is ISaveable saveable &&
gameState.TryGetValue(pair.Token, out JToken data))
            saveable.OnLoad(data);
    }

    if (gameState.TryGetValue("inventory", out JToken
inventoryData))
        (inventory as
ISaveableCustom).OnCustomLoad(inventoryData);

    if (gameState.TryGetValue("objectives", out JToken
objectivesData))
        (objectiveManager as
ISaveableCustom).OnCustomLoad(objectivesData);
}

private async Task SerializeGameState(StorableCollection
saveInfo, StorableCollection worldState, string folderName, string
folderPath)
{
    if (!Directory.Exists(folderPath))
        Directory.CreateDirectory(folderPath);

    string saveInfoPath = Path.Combine(folderPath, "info.sav");
    string saveDataPath = Path.Combine(folderPath, "data.sav");

    await SerializeData(saveInfo, saveInfoPath);
    await SerializeData(worldState, saveDataPath);
}

private async Task SerializeData(StorableCollection buffer,
string path)
{
    string json = JsonConvert.SerializeObject(buffer,
Formatting.Indented, new JsonSerializerSettings()
{
    ReferenceLoopHandling = ReferenceLoopHandling.Ignore
});

    await SerializableEncryptor.Encrypt(SerializationAsset, path,
json);
}

private static async Task<string> DeserializeData(string path)
{

```

```

        return await
        SerializableEncryptor.Decrypt(SerializationAsset, path);
    }
}

public class SaveableObject : MonoBehaviour, ISaveable
{
    public enum SaveableFlagsEnum { Position, Rotation, Scale,
    IsActive }

    public SaveableFlagsEnum[] saveableFlags;
    public string saveableToken;

    public StorableCollection OnSave()
    {
        StorableCollection buffer = new StorableCollection();

        foreach (var flag in saveableFlags)
        {
            switch (flag)
            {
                case SaveableFlagsEnum.Position:
                    buffer.Add("position",
transform.position.ToSaveable());
                    break;
                case SaveableFlagsEnum.Rotation:
                    buffer.Add("rotation",
transform.eulerAngles.ToSaveable());
                    break;
                case SaveableFlagsEnum.Scale:
                    buffer.Add("scale",
transform.localScale.ToSaveable());
                    break;
                case SaveableFlagsEnum.IsActive:
                    buffer.Add("isActive", gameObject.activeSelf);
                    break;
            }
        }

        return buffer;
    }

    public void OnLoad(JToken data)
    {
        if (data.TryGetValue("position", out JToken position))
            transform.position = position.ToObject<Vector3>();

        if (data.TryGetValue("rotation", out JToken rotation))
            transform.eulerAngles = rotation.ToObject<Vector3>();

        if (data.TryGetValue("scale", out JToken scale))
            transform.localScale = scale.ToObject<Vector3>();

        if (data.TryGetValue("isActive", out JToken isActive))
            gameObject.SetActive((bool)isActive);
    }
}

```