Міністерство освіти і науки України
НТУ «Дніпровська політехніка»
Інститут електроенергетики
Факультет інформаційних технологій
Кафедра інформаційних технологій та комп'ютерної інженерії

Пояснювальна записка
Кваліфікаційної роботи бакалавра

Студента Вдовиченко Руслана Сергійовича
Академічної группи 126-21-2
Спеціальності 126 Інформаційні системи та технології
За освітньо-професійною програмою "Інформаційні системи та технології"
На тему: Розробка 2Д гри з видом зверху в жанрі RPG з використанням мови C# та середовища Unity

Крівники	Прізвище, ініціали	оцінка за шкалою		Підпис
		рейтинговою	інституційною	
кваліфікаційної роботи	Нікулін С.Л.			
розділів				
рецензент				
нормоконтролер				

Дніпро
2025

Затверджено:
Професcор кафедри
інформаційних технологій
та комп'ютерної інженерії

_____ Нікулін С.Л._____
(підпис) (прізвище, ініціали)

«_____»_____2025_року

Завдання
на кваліфікаційну роботу
ступеня бакалавра

Студенту Вдовиченко Р.С. академічної группи 126-21-2_____
спеціальності 126 "Інформаційні системи та технології"_____
за освітньою-професійною програмою «Інформаційні системи_____
та технології»_____
на тему:Розробка 2Д гри з видом зверху в жанрі RPG з використанням__
мови C# та середовища Unity_____
Затверджену наказом ректора НТУ «Дніпровська політехніка» від 23.05.2024 р. №469-с

Розділ	Зміст	Термін виконання
Розділ 1	Аналіз предметної області	
Розділ 2	Постановка задач Проектування програми Реалізація додаткових функцій	
Розділ 3	тестування та валідація програми	

Завдання видано _____ _Нікулін С.Л._____
(Підпис керівника) (Прізвище, ініціали)
Дата видачі _____
Дата подання до екзаменаційної комісії _____
Прийнято до виконання _____ _Вдовиченко Р.С.__
(Підпис студента) (Прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 102 с., 15 рис., 1 дод., 8 джерела.

Об'єкт розробки: 2Д гра з видом зверху в жанрі RPG

Мета кваліфікаційної роботи: створити програму, яка допомагає користувачам весело провести дозвілля, в простій грі, не потребуючій потужного ПК та досвіду, досліджуючи ігровий світ, знайомлячись з мешканцями ігрового світу, та беручи участь в ігрових змаганнях проти мешканців .

У вступі здійснено огляд сучасного стану досліджуваної проблематики, визначено мету кваліфікаційної роботи та сферу її використання, обґрунтовано актуальність обраної теми і сформульовано основні завдання дослідження.

У першому розділі проведено аналіз предметної області, розглянуто наявні аналоги з описом їхніх функціональних характеристик, визначено їхні недоліки та обґрунтовано переваги запропонованого підходу.

Другий розділ присвячено постановці задачі, визначенню вимог до програмного забезпечення, вибору методів і алгоритмів, що забезпечують ефективну реалізацію. Описано процес проєктування системи, її структуру, основні алгоритми, вхідні та вихідні дані, а також технічні характеристики й принципи взаємодії з користувачем.

У третьому розділі наведено результати тестування створеної програми, проаналізовано її продуктивність та ефективність функціонування на основі отриманих результатів.

Ключові слова: NPC, C#, Геймплей, UI, canvas, Рушій, Скрипт, ScriptableObject, Проект, Діалог, Міні-гра .

ABSTRACT

Explanatory note:102 P.,15 pic., 1 add., 8 sources.

Object of development: a 2D game with a top view in the RPG genre

The goal of the qualification work is: create a program that helps users have fun in a simple game that does not require a powerful PC and experience, by exploring the game world, getting to know the inhabitants of the game world, and participating in gaming competitions against the inhabitants .

In the introduction, an overview of the current state of the problem under study is carried out, the purpose of the qualification work and the scope of its use are determined, the relevance of the chosen topic is justified, and the main research tasks are formulated.

In the first chapter, the subject area is analyzed, the available analogues are considered with a description of their functional characteristics, their disadvantages are identified and the advantages of the proposed approach are justified.

The second section is devoted to setting the problem, defining software requirements, and selecting methods and algorithms that ensure effective implementation. The system design process, its structure, basic algorithms, input and output data, as well as technical characteristics and principles of user interaction are described.

The third chapter presents the results of testing the created program, analyzes its performance and efficiency of functioning based on the results obtained.

Keywords: NPC, C#, Gameplay, UI, canvas, engine, script, ScriptableObject, Project, dialog, mini-game .

Зміст

Список умовних позначень

ПК - Персональний комп’ютер

UI- User Interface (користувальницький інтерфейс). Компонент що відповідає за візуальну частину.

NPC - None Playable Character (Не Іграбельний Персонаж). Персонаж яким керує не Гравець, а комп'ютер.

Engine- Рушій. Спеціальна програма, яка допомагає розробникам створювати та запускати ігри, надаючи їм необхідні інструменти та функції.

Скрипт - Файл з кодом.

ScriptableObject - Скрипт для храніння інформації.

Canvas - Холст. Об’єкт на якому розміщують інші UI об’єкти.

Вступ

Індустрія розробки відеоігор (Game Development, або геймдев) стрімко розвивається та з кожним роком стає дедалі масштабнішою. Завдяки технічному прогресу, зростанню обчислювальних потужностей та вдосконаленню інструментів розробки, створення ігор стало доступним як для великих студій, так і для незалежних розробників. Ігри вже давно вийшли за межі простої розваги - вони стали засобом самовираження, формою мистецтва та способом розповіді історій. Проте, попри цей стрімкий розвиток, дедалі більше гравців звертають увагу на кризу змісту та унікальності в сучасних проєктах.

Багато нових ігор демонструють однотипний геймплей, передбачуваний сюжет, а головний акцент часто робиться на надмірно реалістичній графіці. Це призводить до того, що багато проєктів, хоч і виглядають візуально вражаюче, залишають по собі відчуття пустоти. Яскравим прикладом цієї трансформації є серія ігор **Assassin's Creed**. Починаючи як інноваційний проєкт із нестандартним ігровим процесом, цікавими історичними подіями та харизматичними персонажами, з часом серія перетворилася на черговий шаблонний бойовик. Нові частини захоплюють картинкою, але часто втрачають глибину сюжету та індивідуальність.

Втім, не все втрачено. Попри домінування великих студій і комерційного підходу, в індустрії все ще з'являються ігри, які створюються з душею - часто навіть однією людиною або невеликою командою. Такі проєкти вражають своєю щирістю, атмосферою та оригінальністю. Яскравими прикладами є 2D-ігри ***Undertale*****, The Binding of Isaac** та **Stardew Valley,** які змогли завоювати серця мільйонів гравців без мільйонних бюджетів, пропонуючи натомість щось набагато цінніше - справжні емоції та унікальний ігровий досвід. І які довели, що навіть у 2D-форматі можна створити унікальні, зворушливі та вражаючі світи.

Розділ 1

Аналіз предметної області

1.1 Масштаби індустрії геймдеву

Індустрія відеоігор є однією з найдинамічніших та найприбутковіших галузей сучасної розвагової економіки. За даними Newzoo, Statista та SuperData, у 2024 році глобальний ринок відеоігор досяг понад $200 мільярдів, що вперше в історії перевищило об’єми кіноіндустрії ($100 млрд) та музичного ринку ($60 млрд) разом узятих.

Ключові фактори зростання:

1. Мобільний геймінг – найбільший сегмент (понад 50% ринку), завдяки доступності смартфонів і F2P-моделям (Free-to-Play).
    - Приклади: Genshin Impact, Honor of Kings (доходи понад $1 млрд на рік).
2. Хмарний геймінг (GeForce Now, Xbox Cloud Gaming) – зростання на 35% щорічно через розвиток 5G та стримінгових технологій.
3. Інді-ігри – завдяки платформам Steam, itch.io та Epic Games Store малі студії отримують доступ до мільйонної аудиторії (Hades 2, Palworld – успіхи 2024 року).
4. Екосистема кіберспорту – ринок перевищив $2 млрд, з майбутнім інтеграції в Олімпійські ігри.

Регіональні особливості:

- Азійсько-Тихоокеанський регіон (Китай, Японія, Південна Корея) – 45% світових доходів.
- Північна Америка – лідер за преміум-іграми (Call of Duty, EA Sports FC).
- Європа – активний розвиток інді-сцени (Україна (Stalker 2), Польща (The Witcher 4)).

Перспективи до 2026 року:

- Прогнозується зростання до $300 млрд через:
    - VR/AR-ігри (Apple Vision Pro, Meta Quest 4).
    - Штучний інтелект у розробці (процедурна генерація контенту, NPC з адаптивною поведінкою).

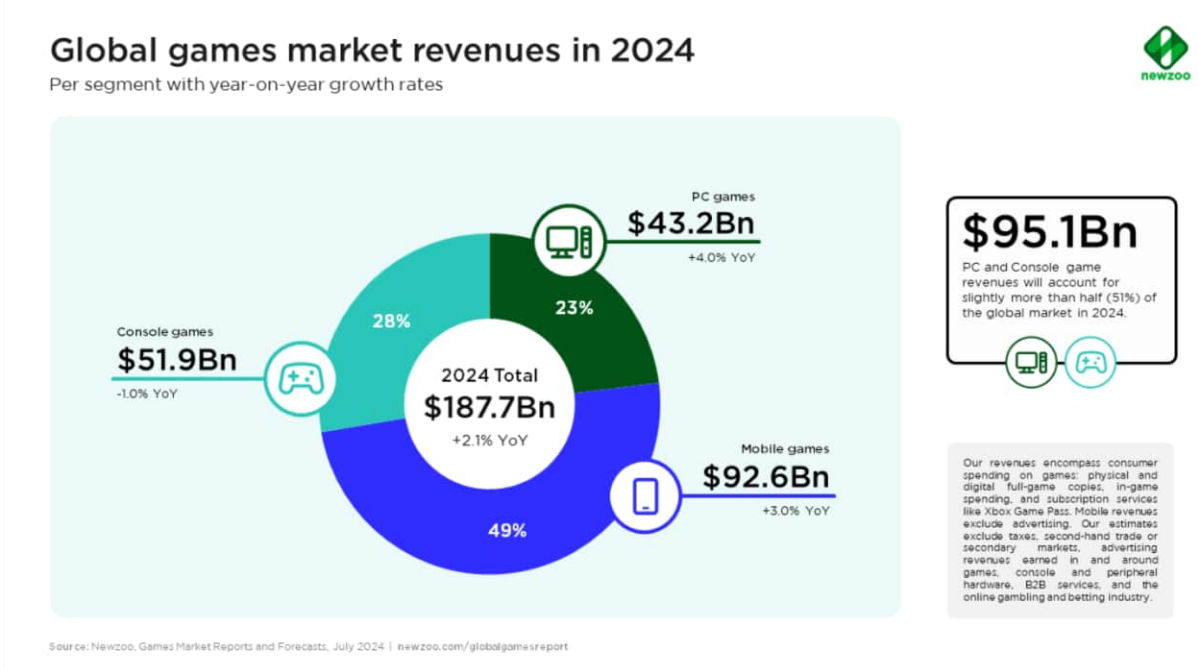


Рис. 1.1 - Діаграмма розповсюдженості ігор на певних платформах.

1.2 Великі компанії та інді-розробники

Сучасний ринок відеоігор поділяється між глобальними корпораціями (Ubisoft, Electronic Arts) та незалежними студіями, які, незважаючи на обмежені ресурси, часто створюють культові проекти.

1.2.1 Корпоративні гіганти

Компанії Ubisoft (понад 20 000 працівників) та Electronic Arts (EA) залишаються лідерами за обсягами продажів, проте їхні ігри часто піддаються критиці:

- Ubisoft (серії Assassin’s Creed, Far Cry) звинувачують у:
    - Шаблонному геймплеї (відкриті світи з повторюваними квестами).
    - Надмірній монетизації (мікротранзакції в однокористувацьких іграх, як у Assassin’s Creed Shadows, 2024).
    - Слабкому сюжеті (гравці відзначають відсутність глибини в сюжеті та персонажах).
- EA зосереджена на ліцензійних франшизах (FIFA, Battlefield), але:
    - FIFA Ultimate Team (FUT) критикують за лутбокси (ігрові азартні механіки).
    - Battlefield 2042 (2021) провалився через технічні проблеми та порожні сервери.

1.2.2 Інді-розробники

На противагу великим студіям, незалежні розробники доводять, що успіх визначається не бюджетами, а оригінальністю:

- Тобі Фокс створив Undertale (2015) самостійно у GameMaker Studio. Гра вразила:
    - Моральними виборами (можна пройти без вбивств).
    - Інтерактивними битвами (гібрид RPG та bullet-hell).
    - Продала 5+ млн копій без реклами.
- Скотт Коутон розробив Five Nights at Freddy’s (2014) як хоррор з мінімалістичним дизайном. Завдяки атмосфері страху та загадковому лору гра:
    - Створила франшизу з 10+ іграми, книгами та фільмом ($130 млн у прокаті).
    - Довела силу «віральної» популярності (Let’s Play на YouTube).

Висновок:

- Корпорації забезпечують технічний прогрес, але часто жертвують креативністю заради прибутку.
- Інді-розробники експериментують, але стикаються з проблемами фінансування та дистрибуції.
- Синергія між ними (наприклад, Ubisoft, що видає інді-ігри через Ubisoft Indie Series) – можливий шлях до балансу.

Рис. 1.2 - Приклад останьої та першої частини ігри Assasins Creed

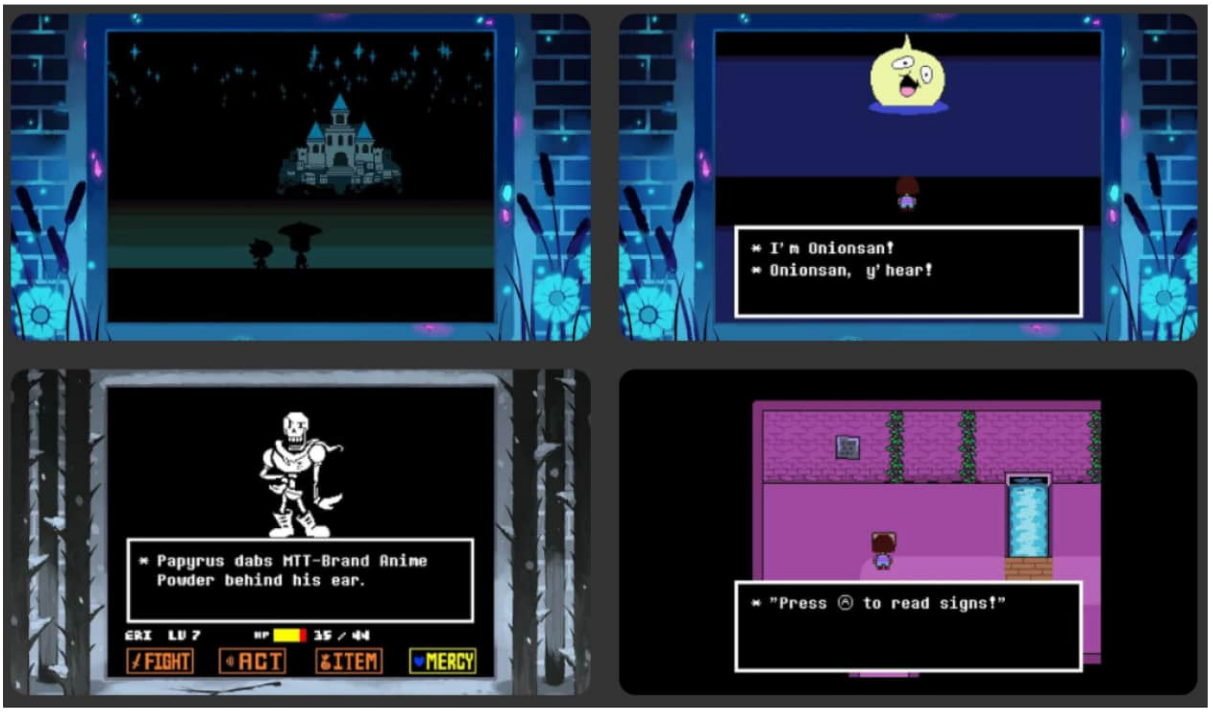


Рис.1.3 - Приклад гри з простою графікою та зробленої одною людиною

1.3 Аналіз переваг та недоліків сучасних відеоігор

Сучасний ігровий ринок демонструє чіткий поділ між унікальними, емоційно насиченими проектами (часто створеними невеликими студіями) та масовими, але критикованими за шаблонність ААА-іграми.

1.3.1 Сильні сторони інді-ігор (на прикладах успішних проектів)

1. Оригінальність та новаторство
    - Undertale (Тобі Фокс) – поєднання RPG, bullet-hell та соціального симулятора з унікальною системою моральних виборів.
    - Hades (Supergiant Games) – процедурно генеровані підземні світи + глибоке впровадження сюжету в roguelike-механіки.
    - Outer Wilds (Mobius Digital) – дослідження всесвіту з фізичною системою орбіт та нелінійним оповіданням.
2. Емоційна глибина та художня цінність
    - Disco Elysium (ZA/UM) – філософський текстовий RPG з політичним підтекстом і психологічною глибиною.
    - Celeste (Maddy Thorson) – платформер, що метафорично розкриває тему тривожності.
    - Gris (Nomada Studio) – візуальна притча про подолання горя через абстрактну анімацію.
3. Персоналізація та авторський стиль
    - Stardew Valley (Ерік Бароне) – створена однією людиною протягом 4 років, замінила для багатьох гравців серію Harvest Moon.
    - Cuphead (Studio MDHR) – ручна анімація у стилі 1930-х років, що вимагала років праці.

1.3.2 Недоліки великих комерційних проектів

1. Шаблонність і втома від франшиз
    - Assassin’s Creed (Ubisoft) – квестова “наповнювачка” (колекціювання 100+ предметів у відкритому світі).
    - Call of Duty (Activision) – мінімальні зміни в геймплеї між різними частинами.
    - FIFA/EA Sports FC (EA) – скіннер-бокси (FUT) замість реального оновлення ігрового процесу.
2. Агресивна монетизація
    - Pay-to-Win механіки (Diablo Immortal, Star Wars Battlefront II до змін після скандалу).
    - Бойовий пас (Battle Pass) – штучне подовження гри через “гринд”.
    - Косметичні DLC за $20+ (Street Fighter 6, Modern Warfare III).
3. Технічні проблеми та «сирі» релізи
    - Cyberpunk 2077 (CD Projekt Red) – провальний запуск на консолях (2020).
    - Battlefield 2042 (EA) – порожні сервери через відсутність контенту.
    - Redfall (Arkane/Bethesda) – нудьговий кооператив із занепадом якості.

1.4 Аналіз технологій та засобів розробки

У сучасній індустрії відеоігор існує велика кількість інструментів, які спрощують та прискорюють процес розробки. Найбільш популярними серед ігрових движків: Unity, Unreal Engine та GameMaker. Порівняємо їх можливості, переваги та недоліки.

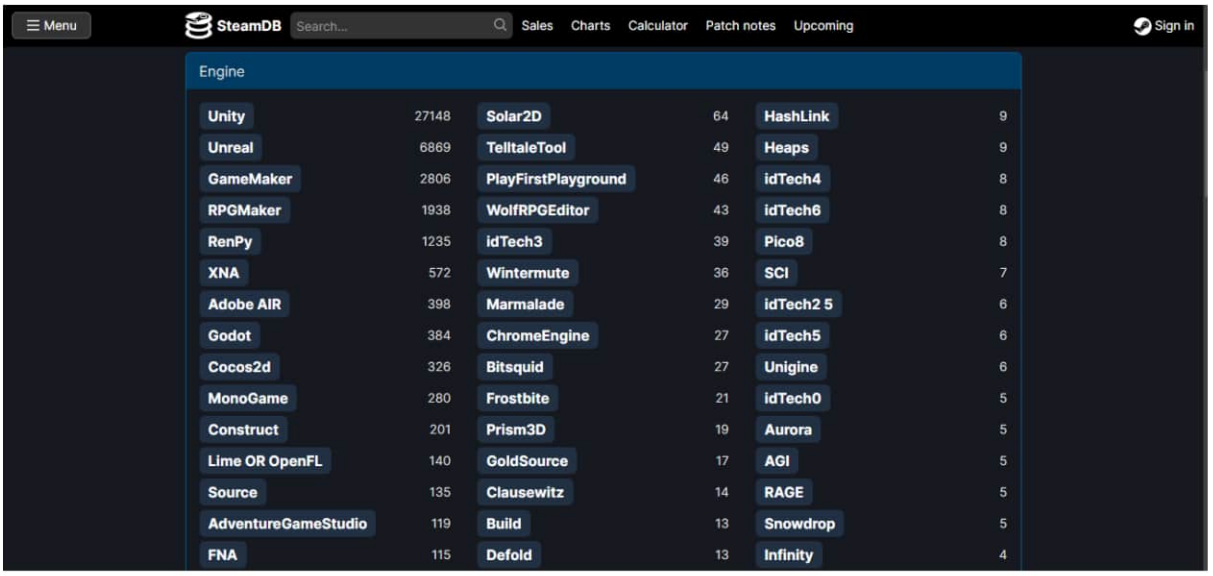


Рис. 1.4 - список найпопулярніших ігрових движків

1.4.1 Unity

Unity - один із найпопулярніших движків у світі, розроблений компанією Unity Technologies. Підтримує 2D і 3D графіку, мобільні платформи, ПК, консолі та навіть VR/AR.

Переваги:

1. Кросплатформеність — підтримка понад 25 платформ, включно з Android, iOS, Windows, macOS, WebGL, PlayStation, Xbox та ін.
2. Підтримка 2D та 3D — однаково ефективний як для 3D проєктів, так і для 2D ігор, що робить його універсальним рішенням.
3. Велика спільнота — численні форуми, документація, уроки та пакети з Unity Asset Store.
4. Зручний для інді-розробників — безкоштовна версія для невеликих студій, простий у вивченні C# як основна мова програмування.
5. Швидкий прототипінг — можливість швидко створити і протестувати ідею.

Недоліки:

- Обмежена графічна якість у порівнянні з Unreal Engine — хоч Unity і може виглядати добре, для фотореалізму потрібні додаткові зусилля.
- Проблеми з продуктивністю у великих проєктах — особливо якщо проєкт погано оптимізований.
- Періодичні зміни політики ліцензування, які викликають невдоволення серед розробників (наприклад, скандал з оплатою за інсталяції у 2023 році).

1.4.2 Unreal Engine

Unreal Engine — потужний рушій, розроблений Epic Games. Найбільше відомий завдяки вражаючій 3D графіці та використанню в AAA-проєктах.

Переваги:

- Фотореалістична графіка — рушій використовує сучасні технології рендерингу, такі як Lumen, Nanite (UE5), що дають надзвичайну якість візуалізації.
- BluePrints — система візуального програмування, що дозволяє створювати логіку без знання коду.
- Безкоштовний доступ — рушій безкоштовний до досягнення $1 млн прибутку (після цього — роялті 5%).
- Використовується у великих проєктах — таких як Fortnite, Gears of War, Final Fantasy VII Remake.

Недоліки:

- Вимоги до «заліза» — для комфортної роботи потрібен потужний комп’ютер.
- Крива навчання — інтерфейс та архітектура складніші порівняно з Unity.
- Надмірний для простих 2D ігор — часто використовується для 3D/VR/AAA-ігор, тоді як для 2D він є «важким».

1.4.3 GameMaker

GameMaker — рушій, орієнтований переважно на створення 2D-ігор. Розроблений компанією YoYo Games. Підходить для інді-розробників, новачків та шкіл геймдеву.

Переваги:

- Простота у використанні — має власну мову програмування (GML) і підтримує візуальне створення логіки без коду.
- Чудовий для 2D — дозволяє швидко створювати платформери, RPG, шутери тощо.
- Оптимізований рендеринг — гарна продуктивність навіть на слабких пристроях.
- Відомі ігри: Undertale, Hyper Light Drifter, Katana Zero були створені на GameMaker.

Недоліки:

- Обмежені можливості 3D — майже повна відсутність повноцінної підтримки 3D.
- Ліцензія не безкоштовна — базова версія безкоштовна, але для експорту на інші платформи потрібні платні пакети.
- Менша спільнота порівняно з Unity чи Unreal.

1.5 Висновок аналізу

Проведений аналіз сучасної ігрової індустрії показав, що попри її стрімкий розвиток та зростання прибутків, значна частина комерційних ігор страждає на втрату унікальності, глибини сюжету та художньої цінності. Великі компанії, як-от Ubisoft чи EA, дедалі частіше випускають однотипні проєкти з акцентом на графіку та монетизацію, ігноруючи емоційну складову та інноваційний підхід. У той же час приклади незалежних розробників — таких як Тобі Фокс (**Undertale**), Скотт Коутон **(Five Nights at Freddy's**) чи студія Future Cat (**OneShot**) — демонструють, що навіть одна людина може створити гру, здатну глибоко зачепити гравця.

Також аналіз показав, що кожен рушій має свої унікальні сильні сторони і підходить для різних задач. Unreal Engine — найкращий вибір для великих 3D-проєктів з високим бюджетом. Unity ідеально підходить для середніх та інді-ігор з можливістю роботи як у 2D, так і в 3D. GameMaker — чудовий варіант для створення 2D-проєктів, особливо для новачків або невеликих команд. У межах цієї роботи обрано Unity, оскільки він поєднує зручність, гнучкість, широке ком'юніті та підтримку 2D-графіки, що ідеально підходить для створення невеликої RPG-гри з емоційним фокусом.

Розділ 2

Постановка задач

Метою є створення 2D рольової гри з унікальним сюжетом, емоційною атмосферою та інтерактивними механіками, орієнтованої на індивідуальний ігровий досвід. Гра має надихатися найкращими прикладами інді-проєктів, таких як *Undertale*, *OneShot* та *Omori*, і демонструвати, що невеликі за масштабом, але щиро зроблені проєкти можуть мати не менший вплив на гравця, ніж великобюджетні продукти.

Основне призначення розробки:

- продемонструвати можливості розробки повноцінної гри однією людиною;
- створити ігровий продукт, що поєднує простоту візуального оформлення з глибиною змісту;
- розкрити потенціал ігрового рушія Unity для реалізації 2D RPG-проєктів;
- закласти фундамент для подальшого розширення гри або її перетворення у повноцінний комерційний продукт.

Область застосування:

- Навчальна: як приклад дипломного проєкту з розробки ігор для студентів IT-спеціальностей, що вивчають C#, Unity та геймдизайн.
- Демонстраційна: як портфоліо для розробника, що показує його вміння в дизайні, програмуванні, написанні сценарію та роботі з графікою.
- Комерційна: у разі доопрацювання проєкт може бути випущений на платформах для інді-ігор (Steam, itch.io) як повноцінна гра.

2.2 Постановка завдання

Основними завданнями, які вирішує система, є:

- Створення ігрового світу з унікальною атмосферою

  Реалізація локацій, персонажів та стилізованого візуального оформлення, що передає настрій гри та підтримує її наратив.
- Розробка діалогової системи

  Побудова інтерактивної системи діалогів, яка дозволяє гравцеві взаємодіяти з персонажами, робити вибір і дізнаватись тонкощі сюжету.
- Інтеграція простих міні-ігор

  Впровадження кількох міні-ігор, які урізноманітнюють геймплей і розкривають ігровий світ через додаткові механіки.
- Реалізація управління персонажем у 2D-просторі

  Створення зручного управління рухом головного героя з використанням Rigidbody2D, фізики та взаємодії з оточенням.
- Забезпечення емоційного впливу через наратив і візуальні ефекти

  Активне використання музики, графіки, сценарних рішень та анімацій для створення емоційного зв’язку між гравцем і грою.
- Модульна архітектура проєкту

  Розробка структури гри таким чином, щоб її було легко підтримувати, масштабувати та розширювати у майбутньому.

2.3. Вимоги до програми

Для забезпечення повноцінної роботи гри та досягнення поставленої мети, до функціональних характеристик розроблюваної системи висуваються такі вимоги:

- **Рух персонажа**

  Гравець повинен мати можливість керувати головним героєм у 2D-просторі, переміщаючись по ігровому світу з використанням клавіатури.

- **Система взаємодії**

  Реалізація взаємодії гравця з об'єктами середовища (NPC, предметами, дверима, тригерами) за допомогою кнопки дії.

- **Система діалогів**

  Персонажі повинні мати власні діалоги. Система повинна підтримувати змінний текст, варіанти відповідей, гілкування діалогу та збереження контексту.

- **Міні-ігри**

  Гравець повинен мати змогу запускати міні-ігри через взаємодію з певними персонажами чи об'єктами. Міні-ігри мають мати чітко визначені правила, переможні умови та результат.

- **Система збереження прогресу**

  Повинна бути реалізована можливість зберігати та завантажувати ігровий прогрес, з урахуванням позиції персонажа, стану діалогів, результатів міні-ігор тощо.

- **Меню гри**

  Наявність стартового меню з можливістю почати нову гру, продовжити збережену, налаштувати гучність тощо.

- **Звуковий супровід та анімації**

  У грі мають бути реалізовані фонові мелодії, звукові ефекти, а також базова анімація персонажів та елементів інтерфейсу.

- **Модульна структура**

  Компоненти гри повинні бути організовані таким чином, щоб нові функції (нові локації, діалоги, міні-ігри тощо) можна було легко додавати або редагувати без порушення загальної структури.

Табл. 2.1 - Основні функції гри

Функція	Опис
Рух персонажа	Переміщення гравця по ігровому світу за допомогою клавіш управління
Взаємодія з об'єктами	Можливість активувати NPC, предмети, двері, зони переходу
Відображення діалогів	Виведення тексту діалогу з анімацією
Міні-ігри	Запуск міні-ігор у рамках сюжету
Головне меню	Стартове меню з налаштуваннями та опціями продовження/нової гри
Звуковий супровід	Фонова музика, звуки дій, озвучення натискань, атмосферні ефекти
Анімація персонажів та UI	Анімації рухів, взаємодій, діалогів та елементів інтерфейсу

Для кращого розуміння логіки взаємодії ігрока з грою, нижче створено таблицю 2.2 в якій перераховано можливі взаємодії ігрока з системою, і яка відображає типові шляхи використання.

Табл. 2.2 – Приклади взаємодії користувача з грою

Функція	Дія користувача	Відповідь гри
Рух персонажа	Гравець натискає клавішу W, A, S або D	Персонаж починає рух у відповідному напрямку
Взаємодія з об'єктами	Гравець підходить до об'єкта та натискає E	Активується відповідна дія (діалог, відкриття дверей, запуск міні-гри)
Відображення діалогів	Гравець читає текст і натискає E	Відображається наступна репліка
Міні-ігри	Гравець натискає на одну з можливих функцій	Запускається логіка, вікно міні-гри оновлюється
Головне меню	Гравець натискає “почати”, “вихід” або “Звук”	Запускає гру, виходить с гри, вимикає звук в грі
Звуковий супровід	Гравець проходить через різні локації чи взаємодіє з об'єктами	Змінюється фонова музика, програються відповідні звукові ефекти
Анімація персонажів та UI	Гравець взаємодіє з меню, предметами, персонажами	Відтворюються відповідні анімації (рухи, відкриття вікон, дії в діалозі)

2.4 Вимоги до складу та параметрів технічних засобів

Дана гра є 2D RPG-проєктом, створеним на ігровому рушії Unity, та не вимагає постійного підключення до інтернету. Вона розрахована на роботу на більшості сучасних та застарілих комп'ютерів. Завдяки оптимізації й помірній графіці, проєкт здатен запускатися навіть на малопотужних пристроях.

Мінімальні технічні вимоги:

- **Операційна система:** Windows 7 / 8 / 10 (64-bit)
- **Процесор:** Intel Core i3 з частотою 2.4 GHz або аналогічний
- **Оперативна пам'ять:** 4 GB
  Відеокарта: Intel HD Graphics 4000 або аналогічна з підтримкою DirectX 10
- **DirectX:** Версія 10 або вище
- **Вільне місце на диску:** не менше 500 MB

Гра не потребує високої продуктивності чи спеціалізованого графічного обладнання, що робить її доступною для широкого кола користувачів, включно з власниками офісних чи бюджетних ноутбуків

2.5. Вимоги інформаційної та програмної сумісності

Програма повинна залишатися максимально простою у структурі та відкритою до подальшого розширення. Основні вимоги до сумісності такі:

- Модульність: архітектура гри має бути побудована за принципами модульності, де кожен компонент (рух, діалоги, міні-ігри, збереження тощо) реалізований окремо, що спрощує тестування,

налагодження та оновлення.

- Читабельність і супровідність коду: код проєкту має бути добре структурованим, задокументованим і логічно поділеним на класи та методи, що дозволяє легко вносити зміни або додавати нові функції.
- Сумісність з популярними версіями ОС Windows: проєкт повинен стабільно працювати на системах Windows 7/8/10/11 (64-bit), без потреби в додатковому ПЗ або драйверах, окрім базових компонентів (на зразок DirectX).
- Використання стандартних форматів: для зберігання даних (наприклад, налаштувань або збережень) використовуються відкриті та зрозумілі формати — такі як JSON чи XML, які легко зчитуються і розширюються.

  Гнучкість до локалізації: структура діалогів та UI повинна передбачати можливість подальшого перекладу гри на інші мови без зміни логіки коду.

Ці вимоги дозволять забезпечити не лише стабільну роботу гри на різних пристроях, а й полегшать її підтримку, оновлення та потенційне масштабування.

2.6. Обґрунтування вибору технологій

Під час розробки гри були проаналізовані популярні інструменти та рушії, що використовуються в ігровій індустрії, зокрема Unity, Unreal Engine та GameMaker. Після порівняння функціональності, складності у використанні та вимог до апаратного забезпечення було прийнято рішення обрати Unity як основну платформу розробки.

Обґрунтування вибору:

- Unity - один з найпопулярніших рушіїв для створення 2D та 3D ігор. Його потужна спільнота, доступ до Asset Store, офіційна документація та кросплатформеність роблять його ідеальним інструментом для розробника-одинака або невеликої команди.

- Мова програмування C#, що використовується в Unity, забезпечує чітку об’єктно-орієнтовану структуру, дозволяє створювати масштабовану архітектуру та легко підтримувати код.

- Збереження у форматі JSON: для збереження даних гри (прогресу, виборів гравця, налаштувань) обрано формат JSON через його простоту, зручність читання, гнучкість та широку підтримку в Unity. Це дозволяє легко реалізувати систему збережень, яку зручно оновлювати та аналізувати.

- UI-вікна замість сцен для міні-ігор: міні-ігри реалізовані у вигляді вікон інтерфейсу користувача (Canvas), які відкриваються поверх основної сцени, без її перезавантаження. Це зменшує час очікування, підвищує плавність ігрового процесу, дозволяє зберігати контекст поточної ситуації та уникати складної логіки переходів між сценами.

- Smart camera (розумна камера): для забезпечення зручного перегляду ігрового простору реалізовано розумну камеру, яка слідує за гравцем з м’яким згладжуванням та обмеженням в межах мапи. Це покращує користувацький досвід та зменшує візуальне навантаження.

2.7. Переваги обраного стеку

У процесі розробки було обрано стек технологій, який забезпечує баланс між простотою реалізації, продуктивністю та можливістю розширення проєкту в майбутньому. Нижче наведено основні переваги кожного з компонентів обраного стеку:

Unity (ігровий рушій)

- Підтримка 2D та 3D ігор
- Велика спільнота та безліч безкоштовних ресурсів
- Зручна система компонентів та об’єктно-орієнтоване проєктування
- Кросплатформеність — підтримка Windows, Android, WebGL та інших платформ
- Потужна система UI, яка дозволяє створювати складні інтерфейси без перемикання сцен

C# (мова програмування)

- Сучасна об'єктно-орієнтована мова
- Зрозумілий синтаксис і широка підтримка в середовищі Unity
- Можливість побудови складної логіки, зручне управління подіями, структурами даних тощо

JSON (формат збереження даних)

- Легкий у використанні і читанні
- Підтримується нативно в Unity через JsonUtility та сторонні бібліотеки
- Гнучка структура, що дозволяє легко додавати нові поля без порушення сумісності

UI (Canvas) для мініігор

- Зменшує навантаження на систему — не потрібно перемикати сцени
- Покращує інтеграцію мініігор у загальну логіку гри
- Дозволяє створювати адаптивні інтерфейси, які реагують на стан гри

Smart Camera (кінематична розумна камера)

- Плавне слідування за гравцем покращує UX
- Не виходить за межі карти
- Забезпечує кінематографічний ефект та атмосферу гри

Цей стек технологій дозволяє створити **якісний, естетичний та функціонально продуманий** 2D RPG-проєкт з діалогами, мінііграми, інтерактивними сценами та елементами персоналізації.

2.8. Функціональне призначення системи

Дана система створена для того, щоб продемонструвати можливості створення атмосферної, сюжетно-орієнтованої 2D RPG-гри невеликою командою або навіть однією особою. Вона акцентує увагу на глибоких діалогах, цікавих мінііграх, естетиці та емоційній складовій, а не на гіперреалізмі чи складних технічних ефектах.

Система має бути:

- доступною для запуску на слабких комп’ютерах без потреби в Інтернеті;
- масштабованою — щоб до гри легко додавались нові сцени, персонажі, міні-ігри;
- зручною у розробці — з чіткою архітектурою та модульністю;
- сфокусованою на гравцеві — з інтуїтивним керуванням і емоційною залученістю.

Табл.2.1

Функція	Опис
Переміщення гравця	Гравець керує персонажем на карті за допомогою клавіш
Взаємодія з NPC	Діалоги з персонажами, отримання інформації або доступ до мініігор
Система діалогів	Виведення реплік з анімацією
Міні-ігри	Вбудовані ігри, реалізовані через Canvas UI, без зміни сцени
Звуковий супровід	Музика, звукові ефекти та фонові шуми залежно від сцени/поверхні
Смарт-камера	Плавне слідування за гравцем у межах поточної сцени
Гнучка система діалогів через ScriptableObject	Зберігання всієї логіки реплік в одному об'єкті для легкого редагування

2.9 Опис використаних технологій та мов програмування

У процесі розробки гри було використано сучасні інструменти та мови програмування, що дозволяють ефективно створювати 2D-проєкти з упором на гнучкість, зручність модифікації та креативність.

Unity разом із мовою програмування C# утворюють потужний та зручний дует для розробки ігор, що пояснює їхню популярність серед інди-розробників та великих студій. Розглянемо їхні переваги та синергію.

2.9.1 Unity: кросплатформовий рушій для сучасних ігор

Unity є одним із найпоширеніших ігрових рушіїв завдяки таким характеристикам:

1. Зручний інтерфейс та інструменти:
    - Візуальний редактор сцен дозволяє швидко створювати рівні та інтерфейси.
    - Drag-and-drop функціонал спрощує роботу з об'єктами та анімаціями.
    - Canvas-система для гнучкої розробки UI та мініігор.
2. Потужна технічна база:
    - Вбудовані системи фізики, аудіо, частинок та шейдерів.
    - Підтримка 2D і 3D-графіки в одному середовищі.
    - Оптимізація під різні платформи (PC, мобільні пристрої, консолі, VR)
3. Екосистема для розробників:
    - Asset Store з тисячами готових моделей, текстур, скриптів та плагінів.
    - Безкоштовна версія (Personal) для невеликих проектів.
    - Активне комьюніті та велика кількість навчальних матеріалів.

2.9.2 C#: мова для Unity

C# є основним мовним інструментом для роботи з Unity через такі переваги:

1. Доступність та продуктивність:
    - Простий синтаксис, схожий на Java/C++, що знижує бар'єр входу для новачків.
    - Автоматичне управління пам'яттю зменшує кількість технічних помилок.
2. Можливості для професійної розробки:
    - Повноцінна підтримка ООП (класи, наслідування, інтерфейси) для чистої архітектури.
    - Події та делегати для гнучкої реалізації ігрової логіки.
3. Інтеграція з Unity:
    - Прямий доступ до API Unity (MonoBehaviour, компонентна система).
    - Гаряче перезавантаження коду (без необхідності перезапускати гру).
    - Можливість оптимізації критичних ділянок коду через Burst Compiler .

Синергія Unity та C#:

1. Швидкий старт проектів: поєднання візуальних інструментів Unity і простоти C# дозволяє швидко прототипувати ідеї.
2. Масштабованість: від простих 2D-ігор до складних 3D-проектів (наприклад, Hollow Knight або Cities: Skylines).
3. Кросплатформеність: код на C# легко переноситься між різними платформами без значних змін.

2.9.3 Unity UI (Canvas): система інтерфейсів

Система Unity UI на базі Canvas є потужним інструментом для створення всіх видів інтерфейсів у грі – від простих меню до складних інтерактивних систем.

Основні можливості Canvas

1. Адаптивний дизайн під будь-які екрани
    - Використання Anchor System для прив'язки елементів до країв екрана чи об'єктів
    - Різні режими рендерингу (Screen Space, World Space, Camera Space)
    - Canvas Scaler для автоматичного масштабування під різні роздільні здатності
2. Розширена робота з графікою
    - Підтримка спрайтів, векторної графіки та SVG (через додаткові плагіни)
    - Маскування для створення складних візуальних ефектів
    - Shader Graph для кастомних візуальних ефектів UI
3. Анімація та інтерактивність
    - Вбудована система UI Animation з Timeline
    - Transition System для станів кнопок (Normal, Highlighted, Pressed, Selected)
    - Event System для обробки введення (миша, тачскрін, геймпад)

Практичне застосування

1. Створення складних меню
    - Ієрархічні системи вкладених меню
    - Прокручувані списки з Scroll Rect
    - Динамічне оновлення контенту через Content Size Fitter

2. Ігрові діалогові системи
    - Поступове відображення тексту (typewriter effect)
    - Дерева діалогів з гілками
    - Інтерактивні QTE-елементи в діалогах
3. Міні-ігри та UI-геймплей
    - Вбудовані пазли та головоломки
    - Інвентарі та системи крафтингу
    - Тактичні карти та інтерактивні плани

Оптимізація продуктивності

1. Ефективне використання
    - Поділ UI на кілька Canvas для зменшення перемальовування
    - Sprite Atlas для зменшення кількості матеріалів
2. Інструменти налагодження
    - Frame Debugger для аналізу рендерингу UI
    - Editor Tools для швидкого прототипування

2.9.4 Використання JSON для збереження даних у грі

Формат JSON (JavaScript Object Notation) став стандартом де-факто для роботи з даними в сучасній розробці ігор, особливо в Unity. Його універсальність та простота роблять його ідеальним вибором для різних аспектів гри.

Основні переваги JSON в розробці ігор

1. Універсальність та простота
    - Текстовий формат, зрозумілий як людям, так і машинам
    - Легко читається та редагується в будь-якому текстовому редакторі
    - Підтримується практично всіма мовами програмування

2. Ефективність у роботі
    - Менший обсяг даних порівняно з XML
    - Гнучка структура, що дозволяє створювати складні вкладені об'єкти
3. Ідеальна інтеграція з Unity
    - Вбудована підтримка через класи JsonUtility (для простих випадків)
    - Проста робота з ScriptableObjects

Практичне застосування JSON в іграх

1. Система збереження прогресу
    - Зберігання стану гравця (health, inventory, progress)
    - Серіалізація позицій NPC та об'єктів
    - Запис досягнень та статистики
2. Діалогові системи
    - Зберігання всієї діалогової дерева
    - Варіанти відповідей та умови їх появи
    - Можливість швидкого редагування сценаріїв
3. Конфігураційні файли
    - Налаштування балансу гри (характеристики зброї, ворогів)
    - Параметри рівнів та локацій
    - Мови та локалізація

2.10 Опис структури програми

2.10.1 Загальний опис архітектури

Дана система створена для того, щоб забезпечити плавний ігровий досвід у 2D-середовищі, де гравець досліджує світ, взаємодіє з NPC,

проходить діалоги, бере участь у мініігрaх та переміщується між локаціями.

Програма побудована на основі поділу відповідальностей: кожен клас виконує окрему функцію, а їхня взаємодія забезпечує цілісність роботи гри. Структура дозволяє легко додавати нові мініігри, сцени, NPC та діалоги без зміни існуючих механік, дотримуючись принципів інкапсуляції та інтерфейсного програмування.

2.10.2 Класи та їх функціональні обов'язки

MenuController.cs

- Відповідає за управління UI головного меню.
- Обробляє натискання на кнопки: "Почати гру", "Продовжити", "Налаштування звуку", "Вийти".
- У режимі паузи (Escape) викликає паузу меню та дозволяє відновити гру.

SoundHolder.cs

- Singleton, який зберігає налаштування звуку між сценами.
- Має булевий прапорець soundEnabled, що визначає, чи ввімкнено звук.
- Інші компоненти звертаються до нього для увімкнення/вимкнення звуків.

GameController.cs

- Центральний керуючий скрипт станів гри.
- Містить змінну GameState, яка може бути: FreeRoam, Dialog, MiniGame.

- Змінює стан гри відповідно до того, що викликає гравець (наприклад, під час діалогу або мінігри — блокує рух).
- Також відповідає за увімкнення/вимкнення звуків згідно з SoundHolder.

PlayerController.cs

- Основна логіка руху гравця за допомогою Rigidbody2D.
- Обробляє натискання клавіш для переміщення (WASD / Arrow Keys).
- Обробляє натискання клавіші E для взаємодії з об’єктами.
- Використовує Physics2D.OverlapCircle або аналог для перевірки наявності взаємодії.
- Якщо об’єкт має інтерфейс Interact, викликає метод Interact().

VectorValue.cs

- Об’єкт ScriptableObject, який зберігає координати останньої позиції гравця.
- Застосовується при переході між сценами, щоб зберегти точку появи гравця.

Interact.cs

- Інтерфейс, який реалізується всіма об’єктами, з якими гравець може взаємодіяти.
- Має єдиний метод Interact(), який викликається з PlayerController.
- Дає можливість розширити гру новими типами взаємодії без змін у гравцеві.

DoorController.cs

- Реалізує логіку переміщення між сценами.
- Має посилання на VectorValue, куди записує позицію гравця перед зміною сцени.
- Використовує SceneManager.LoadScene, щоб завантажити нову локацію.

EnemyController.cs

- Реалізує інтерфейс Interact.
- При натисканні E запускає MiniGameManager.
- Передає посилання на потрібну міні-гру, визначає тип анімацій чи сценарій.

MiniGameManager.cs

- Реалізує логіку конкретної міні-гри (наприклад, Камінь, ножиці, бумага).
- Активує відповідні UI-вікна, запускає анімації рук та обробляє результат.
- Після завершення гри повертає стан GameController до FreeRoam.

NPCController.cs

- Також реалізує Interact.
- При натисканні E запускає діалог.
- Має посилання на DialogData (ScriptableObject з текстом діалогу).
- Викликає метод ShowDialog() у DialogManager.

DialogManager.cs

- Виводить текст діалогу на екран.
- Анімує появу та зникнення діалогового вікна.
- Розбиває діалог на репліки, реагує на клавішу продовження.
- По завершенню діалогу повертає гравця до режиму FreeRoam.

DialogData.cs

- ScriptableObject, який зберігає набір діалогів.
- Має поля ID, lines (список реплік).
- Дозволяє розділити діалоги за персонажами чи сценаріями.

2.10.3 Алгоритми функціонування

1. Рух гравця

- Якщо поточний стан гри — FreeRoam, PlayerController дозволяє рух.
- Input.GetAxis() зчитує напрямок руху.
- Rigidbody2D.MovePosition() переміщує гравця.
- Паралельно в залежності від шару виконується відтворення відповідного звуку кроків (камінь, трава тощо).

2. Взаємодія з об’єктами

- Під час натискання E запускається перевірка колайдерів поруч.
- Якщо об'єкт реалізує Interact, викликається його метод Interact().
- Це може запустити:
    - діалог (NPCController);
    - міні-гру (EnemyController);
    - перехід (DoorController).

3. Діалоги

- NPCController викликає DialogManager.ShowDialog(dialogData).
- DialogManager анімує вікно та почергово виводить репліки.
- Після кожної репліки очікується клавіша для продовження.
- Після останньої репліки діалог закривається.

4. Мініігри

- EnemyController викликає MiniGameManager.StartMiniGame().
- GameController перемикає стан гри на MiniGame.
- Активується UI вікно міні-гри (не сцена, а Canvas), починаються анімації.
- Після завершення гри результат обробляється, і GameController повертається в FreeRoam.

5. Переміщення між сценами

- При взаємодії з об’єктом типу DoorController, його метод:
    - записує координати в VectorValue,
    - викликає SceneManager.LoadScene,
    - після завантаження нової сцени PlayerController ставить гравця у збережену позицію.

2.10.4 Основні переваги архітектури

- Модульність: кожен компонент має чітке призначення.
- Розширюваність: легко додавати нові діалоги, NPC, міні-ігри.
- Інтерфейсність: використання Interact забезпечує єдиний підхід до взаємодій.
- Мінімальна залежність між компонентами, що спрощує тестування та підтримку.

- Гнучка робота з UI: діалоги та мініігри реалізовані як Canvas-вікна, а не окремі сцени.

2.11 Опис роботи розробленої системи

2.11.1 Використані технічні та програмні засоби

Інструменти які використовували під час створення гри перераховані в Табл.2.3.

Табл.2.3 - Використані під час створення гри інструменти

Інструмент	Призначення
Unity 2022.3 LTS	Ігровий рушій, використовується як основне середовище розробки. Забезпечує обробку фізики, анімацій, сценою, UI тощо.
C#	Основна мова програмування для логіки гри.
Visual Studio	Середовище для написання та налагодження коду.
Tilemap, Rigidbody2D, Collider2D	Компоненти Unity для побудови 2D-оточення, обробки фізики та колізій.
ScriptableObject	Структура даних для зберігання діалогів, позицій між сценами і звуків
Animator	Анімація елементів: діалогових вікон, персонажів та міні-ігор.
Canvas (UI)	Інтерфейс користувача - меню, діалоги, міні-ігри.
AudioSource + AudioClip	Система відтворення звуків: кроки, кліки, перемоги, натискання кнопок.

2.11.2 Виклик та завантаження програми

1. Запуск гри

- Після компіляції проєкту у Unity або запуску Build.exe (у разі створення standalone-білду), завантажується початкова сцена.
- У сцені автоматично активується об’єкт з компонентом MenuController, який:
    - зберігае налаштування звуку з SoundHolder.
    - Дозволяє гравцю розпочати нову гру або вийти.

2. Початок гри

- Натискання кнопки "Start" у меню викликає функцію Interact() у DoorController.
- Відбувається завантаження основної сцени гри через SceneManager.LoadScene().
- Після завантаження сцени:
    - Виконується ініціалізація об’єктів: гравець, NPC, двері, вороги тощо.
    - PlayerController зчитує останнє положення гравця з VectorValue і ставить його в потрібну позицію.
    - GameController встановлює стан гри FreeRoam.

3. Завантаження даних

- Дані про звук зчитуються із Singleton-класу SoundHolder, який вже присутній на сцені або позначений як DontDestroyOnLoad.
- Діалоги підвантажуються з об’єкта DialogData, який передається через посилання з NPCController.

2.11.3 Поведінка системи після завантаження

Після завантаження сцени гра повністю готова до взаємодії:

- Гравець керує персонажем (PlayerController).
- Можна взаємодіяти з об’єктами (Interact).
- Сцени змінюються через DoorController.
- Міні-ігри викликаються через EnemyController, а діалоги — через NPCController.

Усе це координується через GameController, який постійно контролює, чи можна гравцеві рухатися, і чи не знаходиться він у діалозі або міні-ігрі.

2.11.4 Опис інтерфейсу користувача

Інтерфейс користувача (UI) у грі створено з урахуванням простоти, інтуїтивної зрозумілості та адаптивності. Всі вікна, кнопки та діалогові елементи реалізовані з використанням Canvas-системи Unity та компонентів UI, таких як:

- Button (кнопка) — використовується для керування діями користувача в меню та під час гри;
- Text (TMP) — для відображення тексту в діалогах, кнопках, сповіщеннях;
- Image — для фону вікон, декоративних елементів, монет у мінііграх;
- Animator — відповідає за плавну появу/зникнення вікон (діалогів, мініігор, повідомлень);
- EventSystem — обробляє взаємодію миші та клавіатури з елементами інтерфейсу.

Інтерфейс адаптовано під клавіатурне керування, але також підтримує мишу. Усі кнопки мають візуальне підсвічування при наведенні та анімацію натискання.

Ігровий UI умовно поділяється на три категорії:

1. Меню (головне, пауза, налаштування) - дозволяє керувати грою;
2. Ігровий інтерфейс - включає діалогові вікна, сповіщення, міні-ігри;
3. Технічний інтерфейс - невидимі компоненти, які керують логікою взаємодій (наприклад, тригери подій, звукові налаштування тощо).

2.11.5 Головне меню

Головне меню є першим, що бачить гравець після запуску гри. Воно реалізоване як Canvas, що займає весь екран, із фоном та трьома інтерактивними кнопками. Назви та призначення кнопок вказані в табл. 2.4

Табл. 2.4 - Кнопки в меню

Назва кнопки	Призначення
Start	Починає гру, завантажуючи основну сцену з ігровим світом.
Exit	Закриває гру.
Sound	Вмикає або вимикає звук у грі (відображає стан у вигляді іконки).

Візуальні елементи меню:

- Фон - статичне зображення, що створює атмосферу гри;
- Кнопки - розміщені вертикально по центру та в верхньому лівому кутку;
- Анімації - плавне появлення та зникнення меню з кнопками;
- Звукові ефекти - кліки кнопок .

Логіка взаємодії:

- Натискання на "Старт" викликає метод Interact() із DoorController, що завантажує нову сцену.
- Кнопка "Звук" викликає SetSound…() із MenuController і змінює поточний стан.
- Кнопка "Вийти" завершує застосунок через Application.Quit().

Рис. 2.1 - Головне Меню

2.11.6 Меню паузи

Меню паузи активується під час гри натисканням клавіші Esc. Воно реалізовано як повноекранне Canvas-вікно з напівпрозорим затемненням фону, що дозволяє користувачу зорієнтуватися в ситуації, не виходячи з гри. Основні елементи меню паузи відображені в Табл. 2.5.

табл. 2.5 - елементи меню паузи

Назва кнопки	Призначення
Продовжити	Закриває меню паузи та повертає гравця у гру.
Вийти	Завершує гру

Особливості реалізації:

- При натисканні Esc, PlayerController активує PauseMenu UI, яке стає видимим.
- Усі рухи гравця та взаємодії блокуються (використовується Time.timeScale = 0).
- При натисканні "Продовжити" викликається метод Resume(), який приховує меню та повертає Time.timeScale = 1 .
- При натисканні "Вийти" викликається метод Application.Quit().

Візуальні характеристики:

- Фон: напівпрозора чорна панель, що перекриває всю сцену.
- Кнопки: розміщені вертикально по центру,.
- Анімації: появлення/зникнення меню.

Користувацький сценарій:

- Гравець натискає Esc → з’являється меню паузи.
- Натискає "Повернутись" → повернення до гри.
- Натискає "Вийти" → гра закривається.

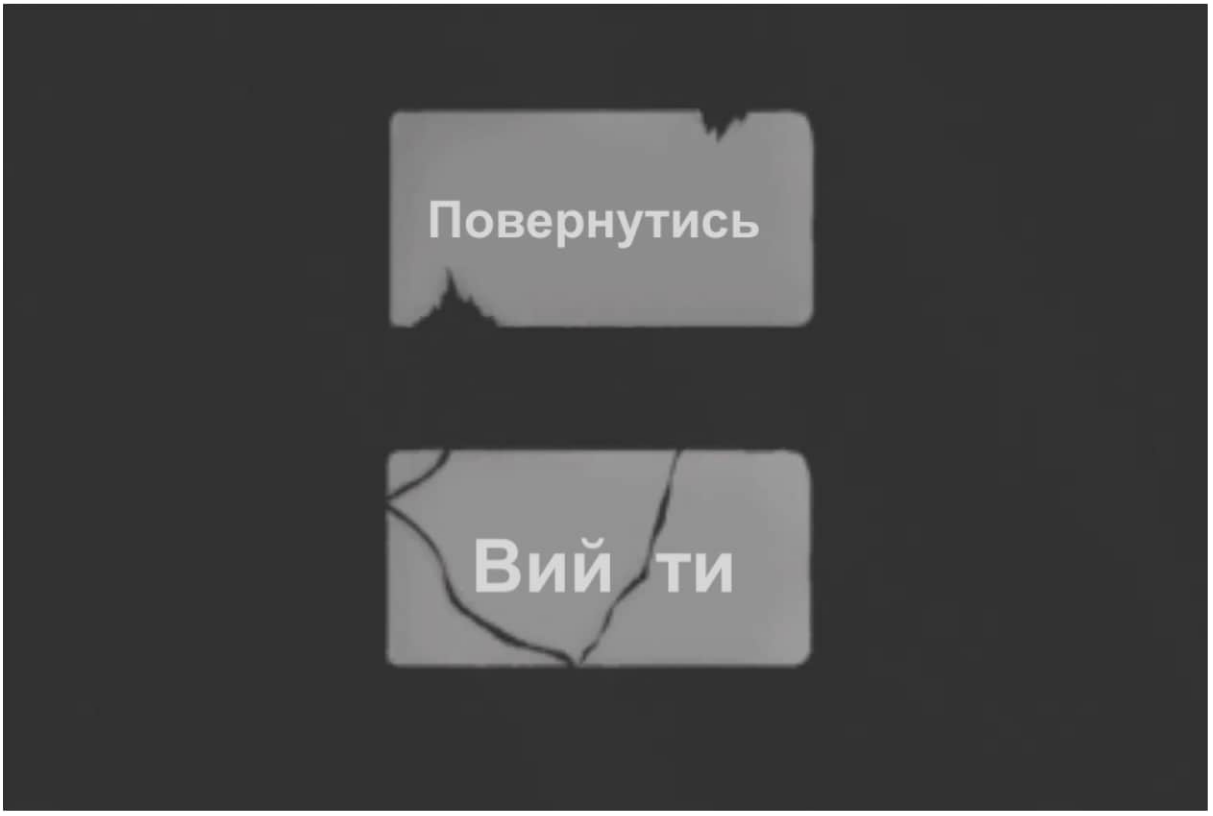


Рис.2.2 - Меню паузи

2.11.7 Основне ігрове вікно користувача

У розробленій грі відсутнє постійне головне інтерфейсне вікно — весь екран повністю присвячений геймплею. Такий підхід дозволяє максимально занурити гравця в атмосферу ігрового світу, уникнувши зайвих інформаційних шарів, які можуть відволікати від дослідження середовища, персонажів та сюжетної лінії.

Особливості основного екрану:

- Повна відсутність UI-елементів під час вільного переміщення.
- Камера плавно слідкує за гравцем, дозволяючи краще відчути простір.
- Вся візуальна інформація гравцю подається тільки контекстно — через діалоги, події, або взаємодії.

Інтерактивність:

- Гравець може вільно рухатися, натискати E для взаємодії з об'єктами (NPC, ворогами, дверима тощо).
- Тільки після цього можуть з’явитися відповідні вікна: діалог, міні-гра, повідомлення, перехід між сценами тощо.
- Інтерфейс "викликається за потребою" — це можуть бути:
    - Діалогове вікно, яке з’являється поверх;
    - Мінігра, яка відкривається повністю замінюючи ігрову сцену;
    - Меню паузи — лише по натисканню Esc.

Переваги підходу:

- Мінімалізм у UI дозволяє:
    - створити атмосферу схожу на ігри Undertale, OneShot, Omori;
    - фокусувати гравця на контенті, а не на індикаторах;
    - підвищити естетичну якість;
    - уникнути візуального перевантаження.

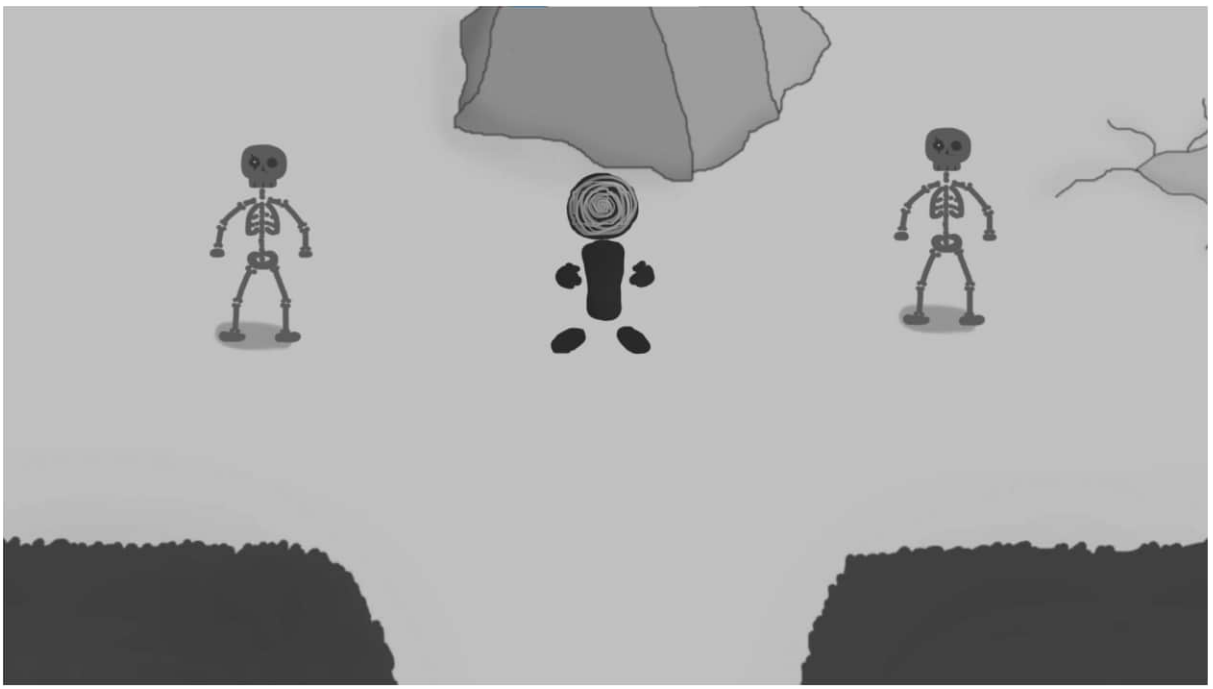


Рис. 2.3 - Основне ігрове вікно

2.11.8 Діалогове вікно

Діалогове вікно - це ключовий елемент взаємодії між гравцем та неігровими персонажами (NPC). Воно активується лише під час безпосереднього контакту з персонажем, що дозволяє зберігати екран чистим під час звичайного геймплею.

Механіка активації:

- Гравець підходить до NPC.
- При натисканні E викликається метод ShowDialog() у DialogManager, який:
    - Вмикає діалогове вікно;
    - Відтворює анімацію ""виїжджання" вікна знизу екрану;
    - Починає виведення тексту з ефектом друкування.

Візуальні характеристики:

- Вікно займає нижню частину екрана.
- Має м’які рамки, стилізованими під ретро-гри.
- Текст з’являється поступово, символ за символом - це створює ефект "живого діалогу", підвищуючи емоційне сприйняття.

Закриття діалогу:

- Після завершення діалогу, натискання кнопки E викликає анімацію зникнення вікна - воно плавно "виїжджає вниз" з екрана.
- Управління повертається гравцю (GameController змінює стан гри з Dialog на FreeRoam).

Технічна реалізація:

- Вікно прив'язано до окремого Canvas з аніматором, який керує анімаціями з’явлення та зникнення.
- Ефект "друкування" реалізовано через корутину (IEnumerator) у DialogManager, яка поступово додає символи до текстового поля.

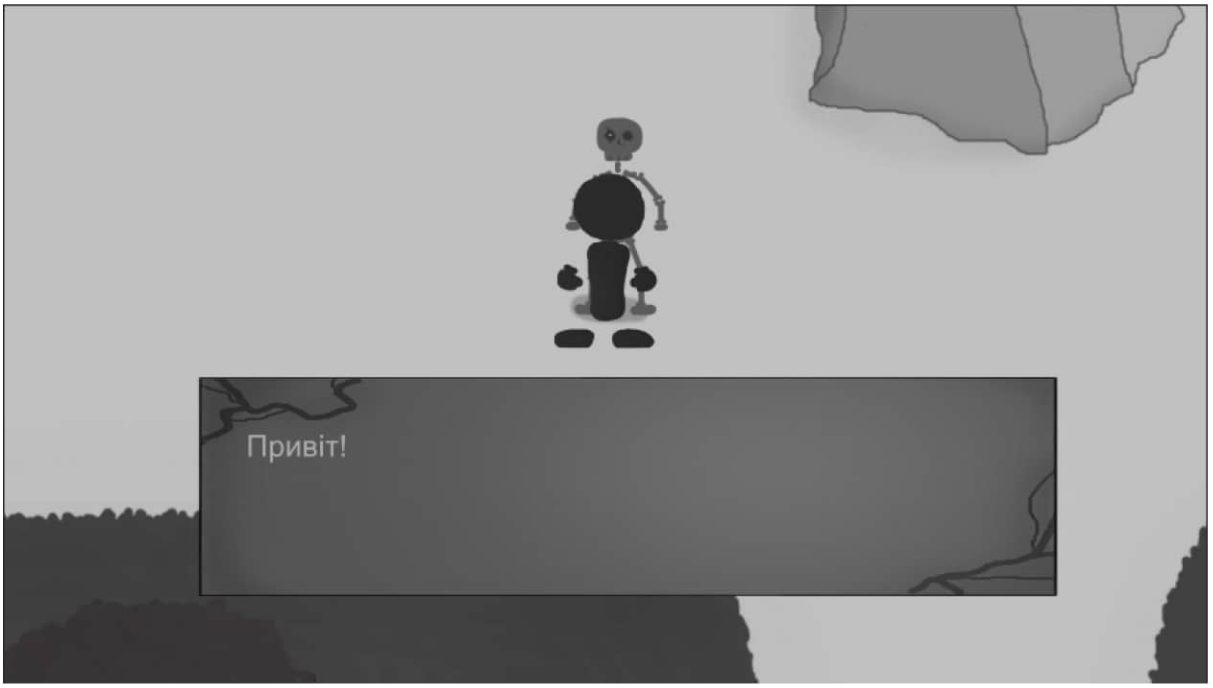


Рис. 2.4 - Діалоговое вікно

2.11.9 Вікно міні-гри

Міні-гра в проєкті реалізована у вигляді класичної гри "Камінь, ножиці, папір", яка активується при взаємодії з певними противниками у світі гри. Як і у випадку з діалоговим вікном, інтерфейс міні-гри з'являється лише в момент запуску і повністю випадає знизу вгору, не перекриваючи інші інтерфейси.

Структура вікна:

- Вікно міні-гри з’являється анімовано з нижньої частини екрана при виклику методу у MiniGameManager.
- Центральну частину інтерфейсу займають:
    - Ліва рука (гравець);
    - Права рука (противник).
- Над руками — текстове поле результату (вихід: перемога, поразка, нічия).
- В нижній частині інтерфейсу - 3 кнопки:
    - Камінь
    - Ножиці
    - Папір
- В верхньому правому куті інтерфейсу - 1 кнопка:
    - Вийти (повертає у FreeRoam)

Ігровий процес:

1. Гравець натискає одну з кнопок вибору.
2. Після вибору:
    - Програється анімація обох рук, яка імітує жест (камінь, ножиці, або папір).
    - Паралельно система генерує вибір противника випадково.
3. Після завершення анімації:
    - У верхній частині екрана з’являється результат раунду у вигляді повідомлення (напр. “Перемога!”, “Нічия”, “Поразка”).
4. Гравець може зіграти ще раз або натиснути кнопку "Вийти", яка закриває міні-гру.

Технічна реалізація:

- Вікно прив’язано до Canvas з аніматором, який відповідає за появу/зникнення.
- Управління логікою вибору та результатом реалізовано через MiniGameManager.
- Анімації рук відтворюються з використанням компонентів Animator для кожного спрайту руки.
- Вивід результату — окреме текстове поле, яке оновлюється після кожного раунду.

Переваги:

- Інтуїтивний інтерфейс, зрозумілий навіть без підказок.
- Анімована взаємодія надає міні-грі візуальної привабливості.
- Вікно не вибиває з атмосфери, залишаючись у стилі решти UI гри.

Рис. 2.5 - Вікно міні-ігри

Нижче, на рисунках 2.6 та 2.7, наведено блок-схему, яка показує структуру розробленої системи, та uml-діаграму яка показує взаємодію з програмою.

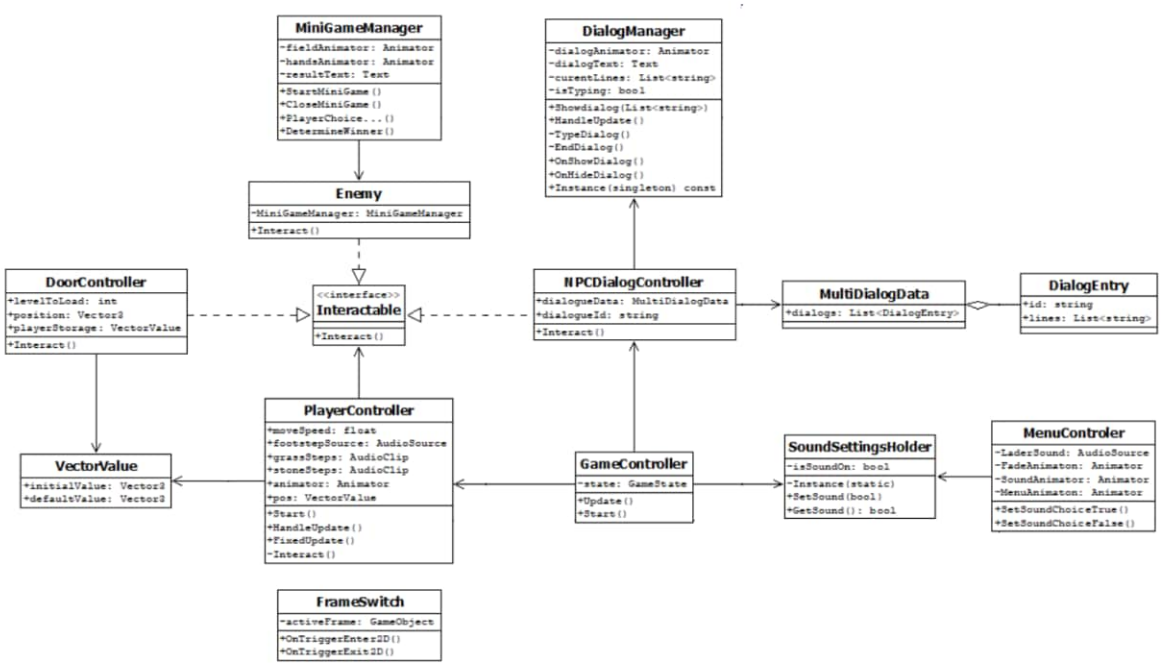


Рис. 2.6 - UML-діаграмма

Рис. 2.7 - блок-схема яка показує структуру розробленої системи

Файл: PlayerController.cs

Опис:

Це головний контролер, який відповідає за переміщення гравця по світу гри.

Також обробляє:

- Взаємодію з об'єктами через Interact.cs;
- Відкриття головного меню;
- Взаємодію з поверхнями (наприклад, для програвання різних звуків кроків).

Файл: Interact.cs

Опис:

Скрипт, що забезпечує єдиний інтерфейс для взаємодії гравця з будь-якими об'єктами у світі.

При натисканні клавіші взаємодії (E) визначає, чи поруч є об’єкт з відповідним інтерфейсом (Interactable), і викликає його метод.

Файл: DoorController.cs

Опис:

Відповідає за перемикання сцен (локацій).

Зберігає координати точки виходу і використовує їх при переміщенні гравця між сценами.

Файл: VectorValue.cs

Опис:

Скрипт для збереження координат гравця між сценами.

Ці координати використовуються PlayerController.cs після завантаження нової сцени, щоб помістити гравця у правильне місце.

Файл: GameController.cs

Опис:

Універсальний контролер стану гри.

Контролює:

- Активність гравця;
- Активність діалогів, боїв, меню;
- Стан звуків.

Використовується іншими скриптами для визначення поточного стану гри.

Файл: SoundHolder.cs

Опис:

Зберігає стан звуку (увімкнено / вимкнено).

Цей об'єкт має бути доступним у всіх сценах, і тому часто є синглтоном.

Файл: MenuController.cs

Опис:

Контролює головне меню гри.

Виконує такі функції:

- Перехід до геймплею (Старт);
- Налаштування звуку (вмикання / вимикання);
- Вихід з гри;
- Активація паузи під час гри.

Файл: FrameSwitch.cs

Опис:

Відповідає за вмикання або вимикання певних зон у сцені.

Може використовуватись для управління логікою сцен, перемикачів або спеціальних умов.

Файл: EnemyController.cs

Опис:

Відповідає за виклик міні-ігор або інших дій, коли гравець підходить до ворога.

Може взаємодіяти з MiniGameManager.cs для запуску гри "Камінь, ножиці, папір".

Файл: MiniGameManager.cs

Опис:

Контролює логіку та анімацію міні-гри.

Основні функції:

- Відображення вікна міні-гри;
- Обробка вибору гравця;
- Анімація рук;
- Визначення переможця / результату раунду.

Файл: NPCController.cs

Опис:

Викликає початок діалогу при взаємодії з NPC.

Працює з DialogManager.cs та передає йому відповідний ID діалогу.

Файл: DialogManager.cs

Опис:

Відповідає за логіку і анімацію діалогу.

- Запускає діалогове вікно (піднімає його знизу);
- Виводить текст з ефектом "написання";
- Контролює завершення діалогу та повернення до вільного режиму.

Файл: DialogData.cs

Опис:

Містить усі діалоги гри у вигляді ScriptableObject.

Кожен діалог має свій унікальний ID, за яким DialogManager.cs знаходить відповідні репліки.

2.12 Реалізація додаткових функцій

У процесі розробки гри було прийнято рішення реалізувати систему переміщення між різними частинами ігрового світу двома окремими методами:

1. Переходи між окремими сценами Unity
2. Переходи між зонами в межах однієї сцени

Такий підхід дозволяє поєднувати переваги обох методів:

- забезпечити чітке розділення великих ігрових локацій,
- оптимізувати завантаження та зменшити витрати ресурсів за допомогою локальних переходів всередині однієї сцени.

Обидва підходи були реалізовані як додаткові функціональні можливості, які суттєво покращують ігровий досвід та роблять світ гри більш живим і гнучким у розширенні.

2.12.1 Призначення та можливості

Система переходів між локаціями та зонами має на меті зробити світ гри більш динамічним, розгалуженим і цікавим для дослідження. Основне її призначення полягає в організації простору гри на логічні сегменти, які забезпечують зручну навігацію та плавний геймплей без затримок або перевантаження пам’яті пристрою.

Можливості системи:

- Плавні переходи між сценами: реалізовані за допомогою скрипта DoorController.cs. При взаємодії гравця з дверима відбувається зміна сцени, а також встановлення координат гравця у новій сцені за допомогою об’єкта VectorValue.cs.

- Активація/деактивація зон усередині однієї сцени: через скрипт FrameSwitch.cs, який керує вмиканням та вимиканням певних частин карти (наприклад, кімнат, коридорів або дворів) без повного завантаження нової сцени.
- Економія ресурсів: зони дозволяють уникнути дублювання сцен, спрощують навігацію і пришвидшують завантаження.
- Гнучкість розширення світу: з такою структурою можна легко додавати нові сцени або розширювати існуючі без потреби перероблювати основну архітектуру гри.
- Збереження стану гравця при переходах: реалізовано за допомогою передачі даних між сценами та об’єктів типу Singleton (наприклад, GameController.cs та SoundHolder.cs).

Ці функції дозволяють реалізувати у грі відчуття великого ігрового світу з живою структурою, при цьому зберігаючи технічну ефективність та простоту розробки.

2.12.2 Основні принципи роботи

У розробці гри було реалізовано дві окремі системи переходу між локаціями, кожна з яких базується на різних технічних підходах, але має спільну мету - забезпечити плавне та логічне переміщення гравця в ігровому світі.

1. Перехід між сценами (Scene-to-Scene)

Цей метод базується на завантаженні нової сцени Unity через SceneManager.LoadScene(). Основні принципи:

- Використання DoorController.cs — скрипт, який при взаємодії (через Interact.cs) ініціює завантаження нової сцени.

- Збереження координат гравця перед виходом із поточної сцени через скрипт VectorValue.cs, щоб у новій сцені встановити правильну стартову позицію.
- Плавне завантаження — за рахунок простоти сцен та збереження даних, перехід виглядає миттєвим і без затримок.

2. Перехід між зонами в межах однієї сцени

Цей підхід застосовується для великих локацій, розбитих на зони (кімнати, території), які вмикаються або вимикаються у межах однієї сцени:

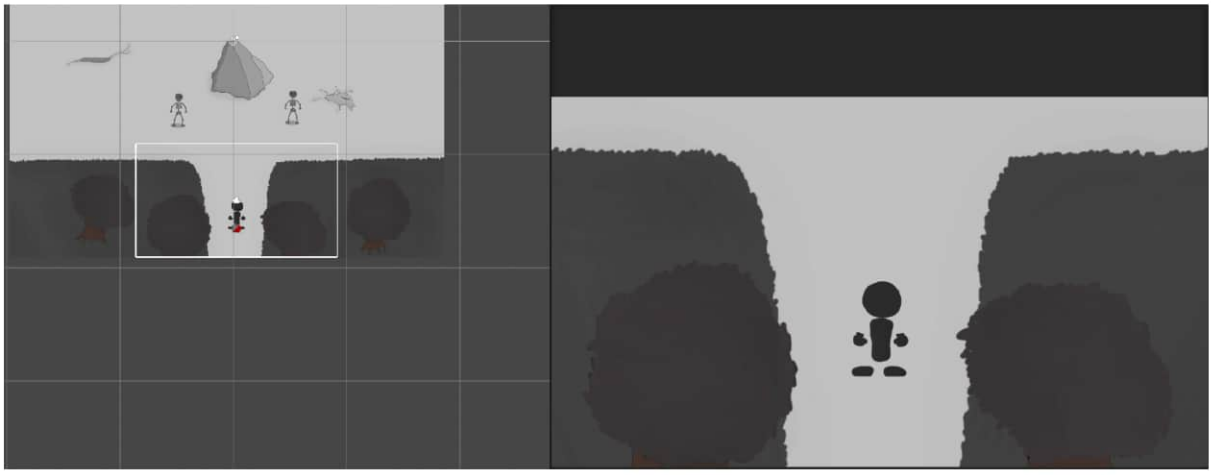
- Скрипт FrameSwitch.cs відповідає за керування зонами — активує або деактивує GameObject-и з певними тегами або іменами.
- Принцип активності: у будь-який момент часу активною є лише одна зона, решта — вимкнені, що зменшує навантаження на систему.
- Збереження контексту: на відміну від повного переходу сцени, при зміні зони не змінюється загальний стан гри або об'єктів Singleton.

Спільні риси обох підходів:

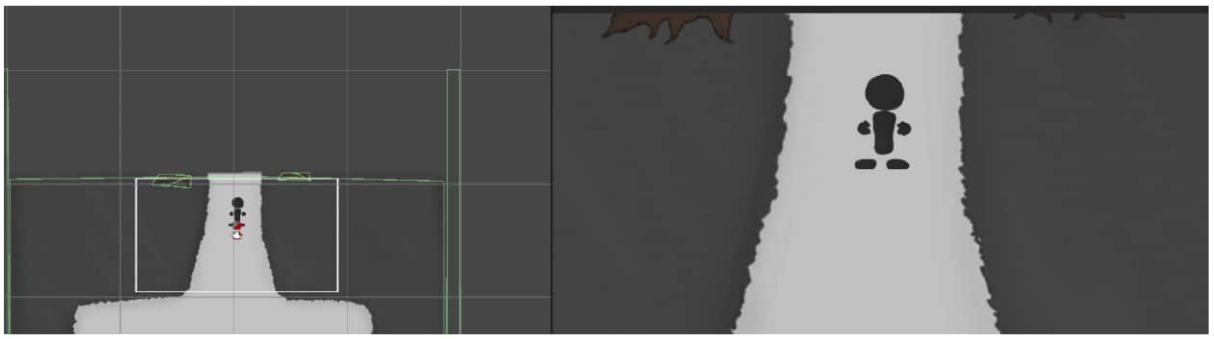
- Інтеграція з системою взаємодії (Interact.cs) — гравець ініціює перехід лише при безпосередній взаємодії з певним об’єктом.
- Контроль з боку GameController.cs — не дозволяє переходити між локаціями під час діалогів або міні-ігор.
- Використання менеджерів стану для збереження позиції, звуків і режиму гри.

Ці принципи дозволили побудувати адаптивну систему переміщення, що легко масштабується і підтримує як класичний підхід зі сценами, так і більш гнучкий - з зонами в одній сцені.

Приклад переходу між зонами одієї сцени відображений на рис.2.8 а) та б)



а)



б)

Рис.2.8 - Перехід від зони А до зони Б однієї сцени

Приклад переходу між сценами, використовуючи “сходи” відображений на рис. 2.9.

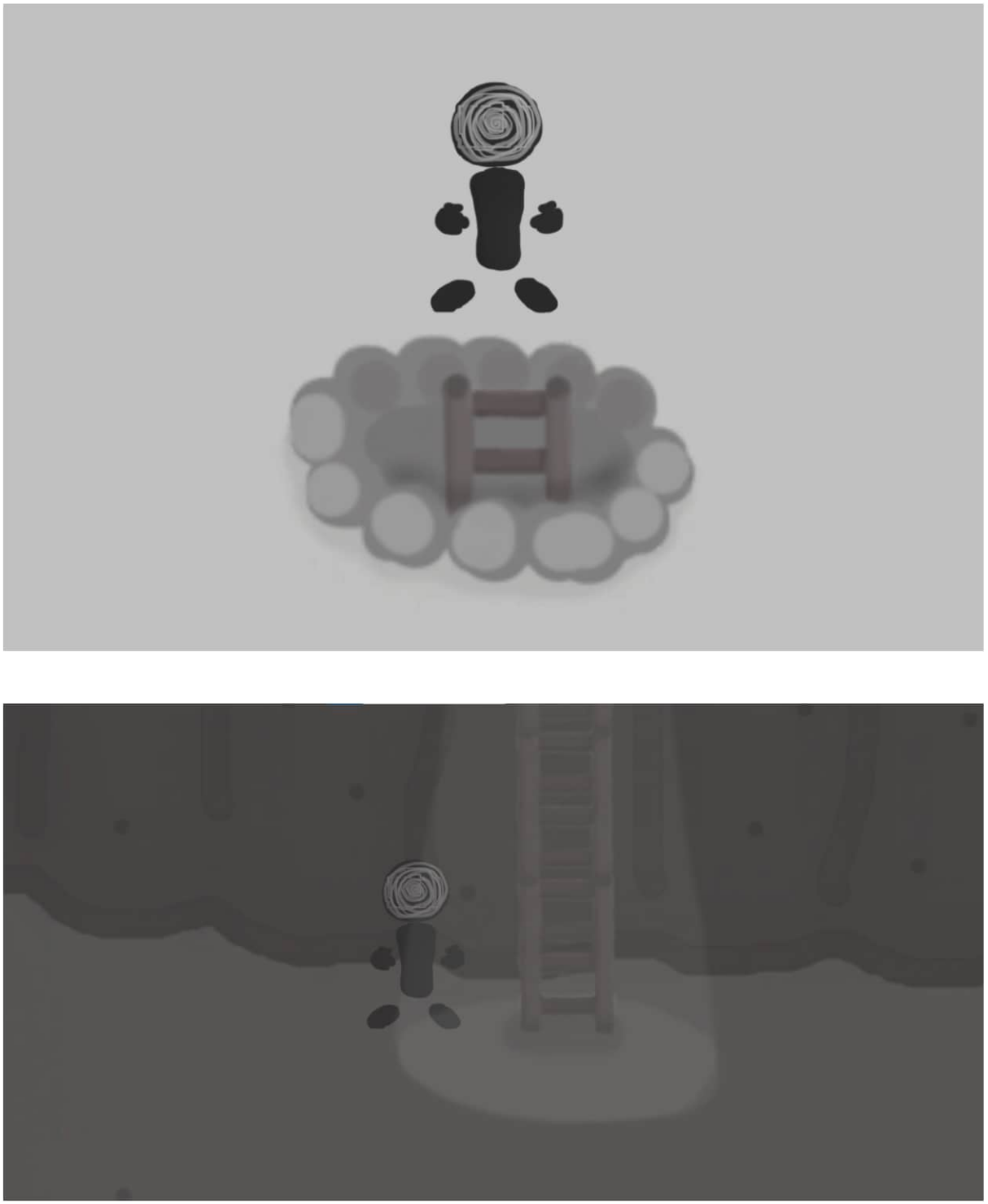


Рис. 2.9 - приклад переходу з сцени А до сцени Б

2.13 Інтеграція додаткових функцій у загальний інтерфейс

Під час розробки гри було реалізовано декілька додаткових функцій, які органічно інтегруються в основну структуру інтерфейсу користувача. Основна мета — забезпечити безшовну та зручну взаємодію гравця з усіма можливостями гри без перевантаження інтерфейсу зайвими елементами.

1. Перехід між локаціями

Було реалізовано два типи переходів:

- Перехід між сценами (повна зміна локації):
    - Активується при взаємодії з певними об'єктами (наприклад, сходами).
    - Інтерфейс не змінюється — зміна відбувається плавно, збереження координат гравця здійснюється автоматично.
    - Після переходу гравець з'являється в новій сцені з тими ж параметрами (позиція, звук тощо).
- Перехід між зонами в межах однієї сцени:
    - Реалізований через поділ сцени на логічні зони (наприклад, кімнати або ділянки).
    - Інтерфейс користувача залишається незмінним, лише змінюється активна зона.
    - Цей підхід дозволяє оптимізувати завантаження та зберегти контекст гри.

Обидва типи переходів не вимагають окремих кнопок чи індикаторів в інтерфейсі — гравець просто підходить до відповідного об'єкта та натискає клавішу взаємодії (E).

2. Міні-гра

Міні-гра "Камінь, ножиці, папір" була вбудована в загальний ігровий процес так, щоб не порушувати цілісності інтерфейсу:

- Вікно міні-гри з'являється аналогічно до діалогового вікна — знизу вгору з анімацією.
- Всі інтерактивні елементи (руки, кнопки вибору, кнопка виходу) знаходяться в межах одного компактного вікна.
- Результат (перемога, нічия або поразка) виводиться вгорі вікна у вигляді тексту.
- Після завершення — вікно міні-гри закривається і гравець повертається до основної сцени.

Це дозволяє сприймати міні-гру як частину основного інтерфейсу, а не окрему систему.

3. Система діалогів

Діалогове вікно також повністю інтегроване в інтерфейс:

- Вікно з'являється лише під час взаємодії з NPC, не заважає основному геймплею.
- Має анімацію появи/зникнення, що зберігає візуальну послідовність.
- Друкований текст реалізує ефект “живого” спілкування.
- Після завершення діалогу інтерфейс автоматично повертається в початковий стан.

4. Меню паузи

Меню паузи відкривається натисканням ESC і містить:

- Кнопку продовження
- Кнопку виходу в головне меню

Воно перекриває сцену, але при цьому є напівпрозорим, щоб гравець бачив фон. Меню не має окремого інтерфейсного шару — лише одне вікно, що зникає після натискання кнопки

2.14 Опис повної логіки взаємодії користувача з грою

Повна логіка взаємодії користувача з грою, вказана на рисунку 2.10.

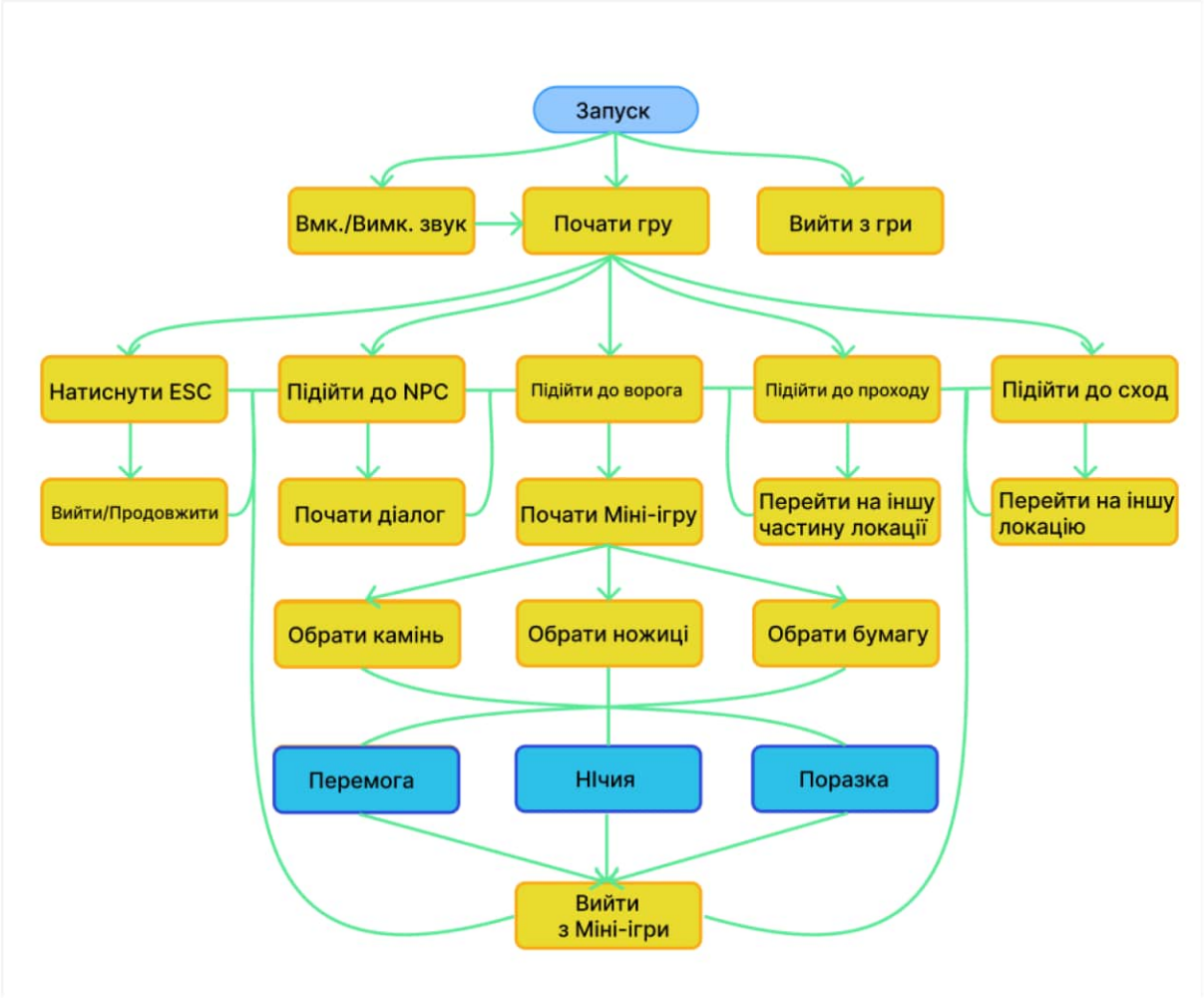


Рис.2.11 - Діаграма повної логіки ігри

1. Запуск гри

Після запуску гри користувач потрапляє до головного меню, яке виконує роль початкової точки входу в ігровий процес. Меню містить три основні опції:

- Вмик./вимк. звуку
    - Користувач може активувати або деактивувати звуковий супровід гри.
    - Обране значення зберігається та враховується в подальших сценах через систему SoundHolder.cs.
- Почати гру
    - Перехід до активної ігрової сцени, де гравець може пересуватись, взаємодіяти з об'єктами й NPC, починати діалоги або міні-ігри.
- Вийти з гри
    - Завершення роботи програми.

2. Під час гри

У геймплеї гравець має декілька варіантів взаємодії, які залежать від його розташування в середовищі гри та дій, які він обирає:

2.1. Натиснути клавішу ESC

- Відкривається меню паузи, що перекриває ігровий екран.
- У меню доступні:
    - Продовжити гру – повертає гравця в гру без втрати прогресу.
    - Вийти в головне меню – повертає до початкового екрану гри.

Цей механізм дозволяє гравцеві зупинити гру в будь-який момент.

2.2. Підійти до NPC

- Якщо гравець наближається до NPC і натискає клавішу E, активується система діалогів.
- Діалогове вікно з'являється знизу екрану з анімацією.
- Текст з’являється поступово, імітуючи процес "друку".
- Після завершення діалогу гравець автоматично повертається до основного режиму гри.

Цей механізм побудований на NPCController.cs і DialogManager.cs.

2.3. Підійти до ворога

- При наближенні до ворожого персонажа та натисканні E, запускається міні-гра "Камінь, ножиці, папір".
- Вікно міні-гри плавно піднімається знизу екрана та містить:
    - Три кнопки вибору (камінь, ножиці, папір)
    - Окрему кнопку "Вийти" для повернення до гри
- Після вибору варіанту запускається анімація обох рук (гравця і ворога), після чого на екрані з’являється результат:
    - Перемога – якщо вибір гравця перемагає ворога.
    - Нічия – якщо вибори однакові.

      Поразка – якщо переміг ворог.

Цей блок реалізується через EnemyController.cs та MiniGameManager.cs.

2.4. Підійти до проходу

- Якщо гравець наближається до спеціального проходу та натискає E, відбувається перехід у іншу зону поточної сцени.
- Такий підхід дозволяє розширити простір однієї сцени без потреби в завантаженні нової.

Це реалізовано через скрипт FrameSwitch.cs.

2.5. Підійти до сходів

- Якщо гравець наближається до сходів і натискає E, гра завантажує іншу сцену — тобто, виконується повноцінний перехід до нової локації.

Цей перехід реалізований через DoorController.cs, який зберігає координати гравця через VectorValue.cs.

3. Результати взаємодій

Усі вищезгадані взаємодії приводять до одного з кількох результатів:

- Відкриття/закриття діалогового або ігрового інтерфейсу.
- Переміщення в межах або між сценами.
- Перемога/нічия/поразка в міні-грі.
- Тимчасове припинення гри або вихід з неї.

Управління всіма цими режимами здійснюється через GameController.cs, який змінює активний стан гри: вільне пересування, діалог, міні-гра тощо.

Підсумок

Дана система логіки побудована так, щоби забезпечити гравцю:

- Інтуїтивне управління та логічні переходи.
- Відсутність перевантаженого інтерфейсу - всі дії контекстні.
- Мінімальні дії для отримання максимального результату.
- Просте, але чітке розділення між активностями: діалог, міні-гра, переміщення.

2.15 Висновок до Розділу

У даному розділі було детально розглянуто структуру, функціональність та реалізаційні аспекти створеної гри. На етапі **постановки завдання** було визначено основні цілі системи — створення інтерактивного ігрового середовища з можливістю взаємодії гравця з об’єктами, NPC, а також участі у міні-іграх.

Розгляд **обраного стеку технологій** дозволив обґрунтувати доцільність використання Unity та мови програмування C# як основи проєкту. Завдяки цьому вдалося реалізувати гнучку архітектуру з чітким розподілом відповідальностей між компонентами.

У розділі було також описано **функціональне призначення системи**, її основні модулі та логіку взаємодії між ними. Були детально розглянуті всі використані скрипти та їхня роль у роботі проєкту, включаючи такі компоненти як PlayerController, GameController, MiniGameManager, DialogManager, DoorController та інші.

Окрему увагу приділено **користувацькому інтерфейсу**. Головною ідеєю інтерфейсу стала мінімалістична концепція, де всі елементи з’являються лише у потрібні моменти — під час діалогів, взаємодії, паузи або міні-ігри. Такий підхід дозволив зберегти ігрову атмосферу без зайвого візуального перевантаження.

Також було реалізовано та інтегровано **додаткові функції**, зокрема систему переходів між локаціями як через зміну сцен, так і в межах однієї сцени через зони. Ці можливості були логічно включені до загального інтерфейсу без додаткових складностей для гравця.

Таким чином, реалізована система відповідає поставленим вимогам, є гнучкою, масштабованою та придатною для подальшого розвитку. Усі компоненти взаємодіють між собою у межах єдиної структури, що забезпечує стабільну та зрозумілу роботу гри як для гравця, так і для розробника.

Розділ 3
Тестування Програми

У цьому розділі розглядається процес тестування та валідації розробленої гри з метою перевірки її коректної роботи, відповідності функціональним вимогам та зручності використання для кінцевого користувача.

Основною метою тестування є виявлення можливих помилок, некоректної поведінки або логічних недоліків у реалізованих механіках, інтерфейсах і взаємодіях. Це дозволяє забезпечити стабільність гри, передбачувану поведінку системи у різних сценаріях, а також досягти високої якості програмного продукту.

Валідація ж покликана підтвердити, що реалізовані функції відповідають початковим вимогам, які були визначені на етапі проєктування. Зокрема, перевіряється правильність роботи таких компонентів як система діалогів, логіка міні-ігор, переміщення між локаціями, робота головного меню, а також реакція інтерфейсу на дії гравця.

У наступних підрозділах буде представлено опис проведених тестів, їхні результати, виявлені недоліки (за наявності) та способи їх усунення. Це дозволить об'єктивно оцінити готовність програми до використання та можливість її подальшого масштабування.

3.1 Методологія тестування

Тестування програмного забезпечення - це процес перевірки програми з метою виявлення помилок, дефектів або невідповідностей між очікуваними та фактичними результатами роботи системи. Тестування дозволяє гарантувати стабільність, надійність і якість продукту перед його розповсюдженням або передачею кінцевому користувачу.

3.1.1 Основні методи тестування:

1. **Модульне тестування**

   Перевіряє окремі модулі або функії програми незалежно один від одного. Часто використовується під час розробки для швидкого виявлення помилок у логіці.
2. **Інтеграційне тестування**

   Тестує взаємодію між модулями. Допомагає виявити проблеми, що виникають при з'єднанні кількох компонентів системи.
3. **Системне тестування**

   Перевірка всієї системи в цілому, з точки зору відповідності специфікації.
4. **Функціональне тестування**

   Оцінює, чи виконує система свої функції згідно з вимогами користувача.
5. **Регресійне тестування**

   Проводиться після внесення змін у програму, щоб перевірити, чи не з’явилися нові помилки у вже протестованому функціоналі.
6. **Тестування «чорного ящика»**

   Тестувальник не має доступу до внутрішньої структури коду, тестує тільки вхідні та вихідні дані.

7. **Тестування «білого ящика»**

   Тестувальник має повний доступ до коду та логіки реалізації. Аналізується структура програми.

3.1.2 Обрана методологія

У процесі створення гри були обрані такі типи тестування:

- **Функціональне тестування** — як основний метод для перевірки виконання ключових сценаріїв: взаємодія з об'єктами, запуск діалогів, перехід між локаціями, робота меню та міні-ігри.
- **Інтеграційне тестування** — для перевірки взаємодії між скриптами, такими як PlayerController, GameController, DialogManager, MiniGameManager та іншими.
- **Тестування "чорного ящика"** - використовувалося під час фінального тестування інтерфейсу та логіки гри з точки зору користувача, без заглиблення у код.

Такий підхід дозволив всебічно перевірити програму: як на рівні окремих модулів, так і в контексті повноцінного ігрового процесу. Він також забезпечив достатню гнучкість і дозволив оперативно виявляти та виправляти помилки в логіці або взаємодії компонентів.

3.2 Функціональне тестування

Функціональне тестування — це один з ключових етапів перевірки розробленої гри, під час якого оцінюється, наскільки коректно працюють усі основні функції програми згідно з технічними та користувацькими вимогами. У цьому методі ми не заглиблюємось у внутрішню структуру коду, а перевіряємо поведінку системи на основі вхідних дій гравця та очікуваних результатів.

Основна мета:

- Переконатися, що усі геймплейні механіки працюють відповідно до задуму.
- Перевірити логіку діалогів, взаємодій, переміщень та міні-ігор.
- Забезпечити стабільність програми під час виконання базових сценаріїв.

Що саме тестувалося:

1. Система управління гравцем
    - Переміщення гравця у всіх напрямках за допомогою клавіш WASD.
    - Реакція гравця на колізії з об'єктами та межами карти.
    - Взаємодія з об’єктами при натисканні клавіші E.
2. Діалогова система
    - Активація діалогу при підході до NPC.
    - Коректне відображення вікна діалогу (висування знизу, друкування тексту).
    - Завершення діалогу при натисканні клавіші продовження.

3. Головне меню
    - Кнопка «Старт» запускає гру зі стартової локації.
    - Кнопка «Звук» змінює стан звуку (вкл/викл) і зберігає його.
    - Кнопка «Вийти» коректно закриває програму.
4. Меню паузи
    - Активується натисканням клавіші Esc.
    - Кнопка «Продовжити» відновлює гру.
    - Кнопка «Вийти» вимикає гру.
5. Переміщення між локаціями
    - Взаємодія з дверима переміщує гравця у нову сцену.
    - У разі зонування - перемикання зон в межах однієї сцени без помилок.
6. Міні-гра (камінь-ножиці-папір)
    - Відкривається при взаємодії з NPC-противником.
    - Вибір опції гравцем викликає анімацію та логіку результату.
    - Кнопка «Вийти» коректно закриває вікно гри.
7. Збереження/відтворення налаштувань
    - Перевірено, що вимкнення/ввімкнення звуку зберігається при зміні сцен.

Інструменти, які використовувалися:

- Unity Editor для емуляції поведінки гравця.
- Виведення логів (Debug.Log) для перевірки виклику потрібних методів.
- Ручне проходження гри кількома тестувальниками для імітації дій користувача.

3.2.1 Процесс тестування

Тестування розпочалося з перевірки основної механіки - переміщення гравця по ігровому світу. Усі напрямки руху працювали коректно, зіткнення з об'єктами та стінами були обмежені колізією, що підтверджує правильну реалізацію фізики та управління.

Далі перевірялася взаємодія з об'єктами через натискання клавіші E. У грі використовується інтерфейс Interact, який успішно обробляє команди гравця — при наближенні до NPC або дверей система викликала відповідні функції без затримок чи збоїв.

Особливу увагу було приділено системі діалогів. Після взаємодії з NPC на екрані з’являлось діалогове вікно, яке анімовано виїжджало знизу, а текст відображався поступово, створюючи ефект «написання в реальному часі». Усі діалоги послідовно прокручувались за натисканням кнопки, не спостерігалося ніяких зависань чи пропусків реплік. результат перевірки відображено на рис. 3.12.

Рис.3.12 - Стабільний перехід діалогів при натискані E

Було протестовано головне меню гри, яке включає три основні кнопки — «Старт», «Вийти» та «Звук». Усі елементи працювали стабільно: запуск гри, вихід із неї та перемикання звуку відбувались коректно. Налаштування звуку зберігалося навіть при переході між сценами.

Після цього перевірялося меню паузи, яке активується натисканням клавіші Esc. Вікно з напівпрозорим затемненням екрану з’являлось стабільно, а кнопки «Продовжити» та «Вийти» повертали гравця у гру та закривали її, без будь-яких проблем.

Окрема частина тестування стосувалася міні-гри «Камінь, ножиці, папір». Була перевірена логіка запуску з боку противника, коректність анімацій, вибору гравця та відображення результату. Після завершення раунду результат гри відображався над анімованими руками, а натискання кнопки «Вийти» повертало гравця в основний режим гри. Резульат перевірки міні-гри відображено на рис. 3.13.

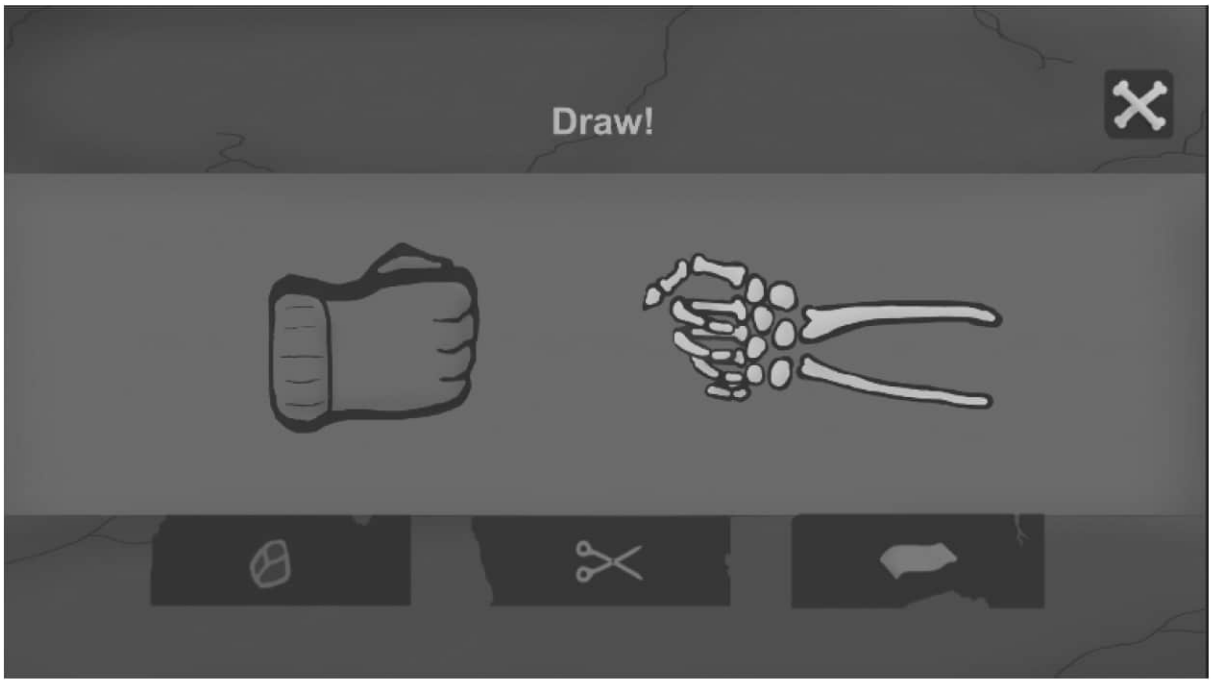


Рис. 3.13 - Коректне відображення результату гри

Також успішно протестовано механіку переміщення між локаціями. Була перевірена як система переходу між окремими сценами, так і переміщення в межах однієї сцени між її зонами. В обох випадках координати зберігалися та коректно оброблялися, а перехід відбувався миттєво. Результат перевірки відображений на рис. 3.14 та 3.15

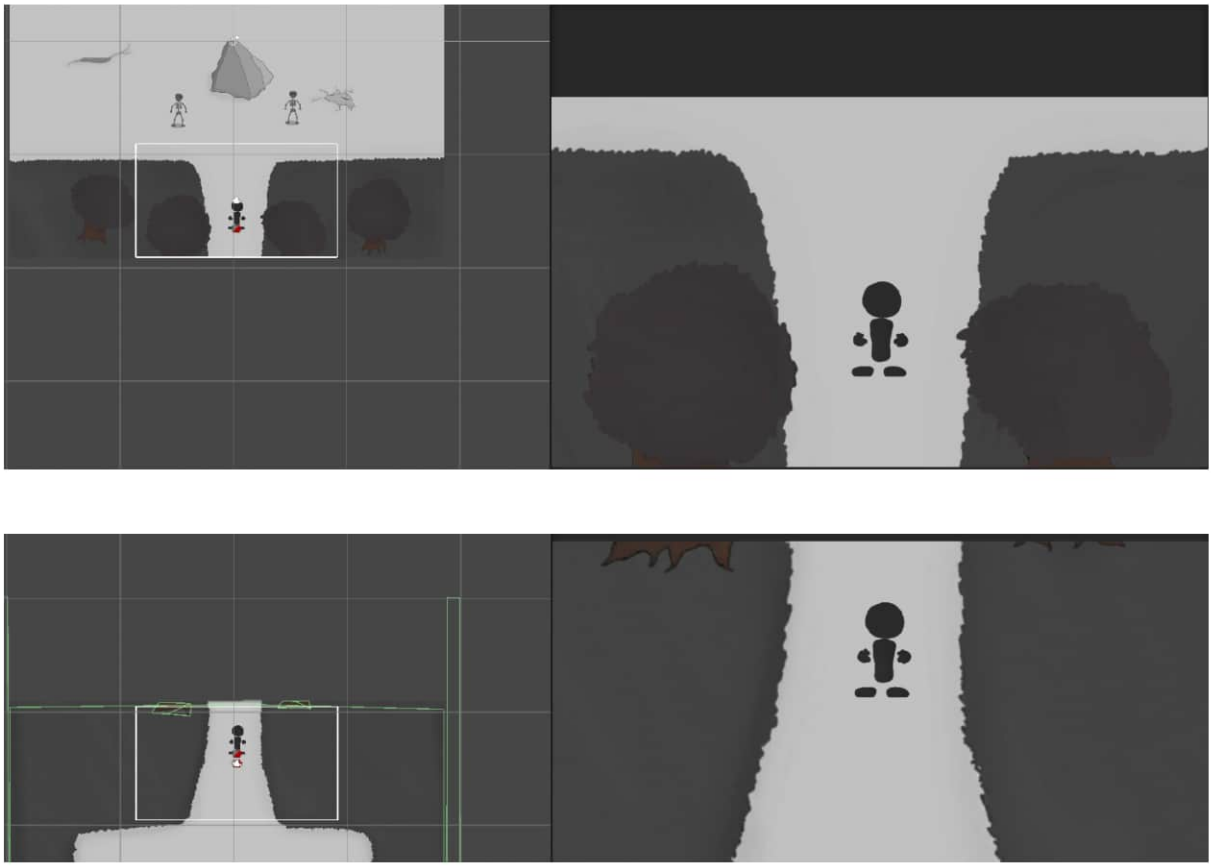


Рис.3.14 - Перехід від зони до зони

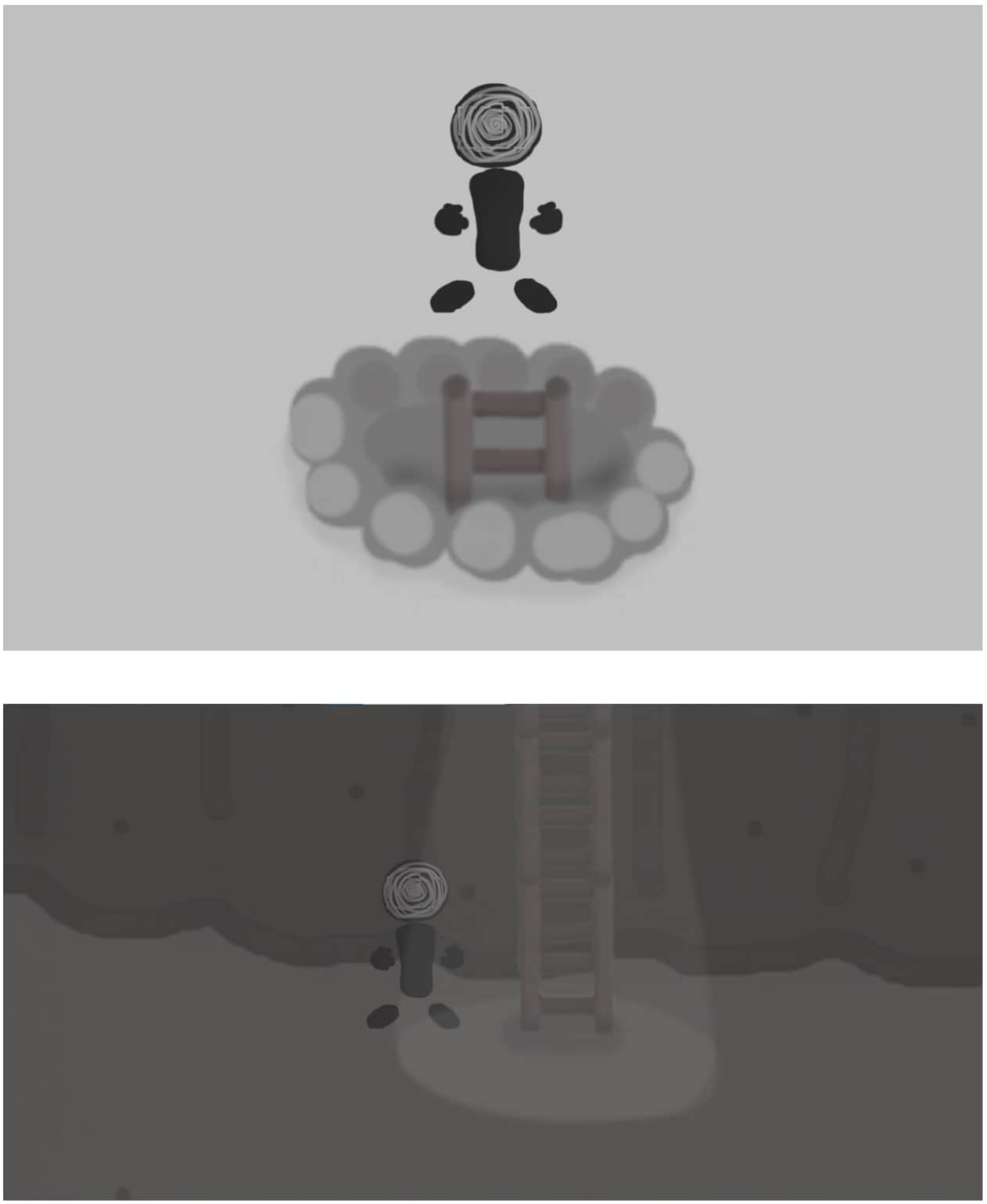


Рис. 3.15 - перехід між сценами

Завершальним етапом стало тестування системи звуку. Була перевірена логіка зберігання вибору гравця (увімкнений/вимкнений звук), і це налаштування зберігалося під час усіх переходів та при повторному вході в гру.

Результат тестування виявився повністю позитивним — жодної помилки чи нестабільної поведінки помічено не було. Всі функції працюють відповідно до запланованого функціоналу, інтерфейс інтуїтивний і не створює перешкод у взаємодії з грою. Це дозволяє стверджувати, що система є функціонально завершеною, стабільною та готовою до використання кінцевим користувачем.

3.3.Інтеграційне тестування

Інтеграційне тестування — це етап перевірки, під час якого окремі модулі програми об’єднуються в єдину систему, і тестується їх взаємодія між собою. На відміну від функціонального тестування, яке зосереджене на перевірці окремих функцій, інтеграційне тестування дає змогу виявити помилки, що виникають при з’єднанні різних частин програми, передачею даних, подій або станів між ними.

У контексті розробленої гри особливу увагу було приділено тестуванню зв’язків між такими основними модулями, як:

1. PlayerController, Interact та GameController

   Гравець може взаємодіяти з будь-яким об’єктом через інтерфейс Interact, який викликається в PlayerController. Усі об’єкти (NPC, вороги, двері тощо) мають реалізацію цього інтерфейсу.

2. NPCController, DialogManager та DialogData

   Перевірялося, чи правильно NPC ініціюють діалог, чи передається коректний ID діалогу, чи обробляються репліки через DialogManager, і чи зчитуються вони з DialogData. Було важливо переконатися, що немає випадків втрати даних, зависання діалогу або повторення вже прочитаних реплік.

3. EnemyController, MiniGameManager та GameController

   При взаємодії з ворогом гра повинна переключитись у режим міні-гри. Перевірялося, чи EnemyController коректно ініціює міні-гру, і чиотримує MiniGameManager сигнал початку.

4. MenuController та SoundHolder

   Під час тестування перевірялася коректна взаємодія між меню та системою налаштувань. Зокрема, перевірялось, чи MenuController читає та застосовує налаштування звуку з SoundHolder, і чи ці налаштування не втрачаються при переходах між сценами.

5. DoorController та VectorValue

   Перевірялося, чи координати гравця, що зберігаються у VectorValue, правильно зчитуються після переходу на нову сцену через DoorController. Це дозволило забезпечити правильне розміщення гравця в новій локації без помилок позиціонування.

6. Пауза та інші режими

   Важливим етапом інтеграційного тестування стала перевірка реакції гри на натискання Esc, коли гравець може перебувати в різних станах — у міні-грі, діалозі або у вільному режимі.

3.3.1 Процесс тестування

Після завершення функціонального тестування приступаємо до інтеграційного тестування, щоб переконатися, що всі окремі частини гри коректно взаємодіють одна з одною. Було вирішено зосередитися на ключових ланцюгах взаємодії: гравець — об’єкт — менеджер ігрового стану.

Одним з перших тестів стала перевірка взаємодії гравця з NPC. Під час підходу до персонажа та натискання клавіші взаємодії (E), PlayerController викликав Interact, який вів до запуску діалогу через NPCController. GameController при цьому переводив гру в режим Dialog, а DialogManager коректно відображав анімоване діалогове вікно з потрібними репліками. Діалог завершувався без збоїв, гра поверталась у режим FreeRoam, і керування гравцем відновлювалося. У результаті - успішна взаємодія трьох ключових модулів без конфліктів.

Наступним етапом стала перевірка міні-гри. При натисканні на E біля ворога запускалася логіка EnemyController, яка активувала MiniGameManager. Одночасно GameController переводив гру в режим MiniGame, блокуючи управління рухом. У міні-грі все працювало: анімація рук запускалася, гравець робив вибір, відображався результат, а після натискання кнопки "Вийти" - все поверталося у початковий стан. Жодних проблем з режимами, анімацією чи управлінням не виявлено.

Також була перевірена логіка переміщення між сценами через двері. DoorController зберігав позицію гравця у VectorValue, переключав сцену, після чого PlayerController отримував правильні координати. Гравець з’являвся в новій локації в потрібному місці, без помилок у розміщенні.

Після цього було протестовано реакцію гри на налаштування звуку. Кнопка в меню змінювала значення у SoundHolder, яке зчитувалося у GameController при запуску сцени. Музика вимикалась або вмикалась відповідно до збережених налаштувань. Функція працювала стабільно навіть при повторному запуску гри або поверненні до головного меню.

Нарешті, було протестовано відкриття меню паузи під час різних станів. Гра правильно обробляла натискання Esc лише у вільному режимі. У діалозі або міні-грі меню не з’являлось - це запобігало порушенню логіки гри. Це підтвердило стабільність режимів та правильне керування станами.

Усі ці тести довели, що всі частини гри, від діалогів до переходів між локаціями - працюють як єдиний злагоджений механізм. Система правильно обробляє сценарії взаємодій і не дає збоїв навіть у складних послідовностях дій. Інтеграція модулів виявилася успішною, що дозволяє вважати гру стабільною в роботі.

3.4 Тестування "чорного ящика"

Для перевірки роботи всієї системи було проведено ряд сценаріїв, де головна мета - протестувати зовнішню реакцію гри на дії користувача.

Наприклад, при запуску гри з головного меню користувач натискає кнопку "Старт". У результаті цього очікувалося, що гра розпочнеться, головне меню зникне, і з’явиться ігровий світ. Саме це і відбулося - підтвердження того, що інтерфейс працює правильно й логіка переходу реалізована коректно.

Також було перевірено, що при натисканні кнопки "Звук", спрайт кнопки змыюэться, а музика вимикається або вмикається. Перевірка не потребувала знання, як це реалізовано в коді - лише факту зміни звуку, що підтвердило очікувану функціональність.

Під час гри гравець підходив до NPC і натискав клавішу E. В результаті на екрані з’являлося анімоване вікно з діалогом. Текст "друкувався" поступово, що також відповідало очікуваній поведінці. Після завершення діалогу, гравець знову міг керувати персонажем. Весь сценарій був успішно відтворений, без багів або зависань.

Окремо був перевірений перехід між локаціями: при наближенні до дверей та натисканні E, відбувався перехід на іншу сцену. Після завантаження нової сцени, гравець з’являвся на правильній позиції. Тест показав, що візуальний і геймплейний результат повністю відповідає очікуванням, хоча механізм збереження позиції прихований від гравця.

Міні-гра "Камінь, ножиці, папір" також була перевірена методом чорного ящика. Гравець бачив кнопки, натискав одну з них, після чого анімація виконувалась, і виводився результат — перемога, поразка або нічия. Цей сценарій повністю відповідав поведінці, яку очікує користувач, і не викликав помилок.

В результаті, усі тести методом чорного ящика були пройдені успішно. Жодних збоїв у поведінці, логіці чи інтерфейсі гри не виявлено. Програма адекватно реагувала на всі можливі дії користувача. Це дозволяє зробити висновок, що з точки зору гравця гра стабільна, інтуїтивна та функціональна, навіть без необхідності знати її внутрішню структуру.

3.5 Висновок до розділу "Тестування програми"

У ході проведення тестування було детально перевірено функціональність, інтеграцію модулів та реакцію системи на дії користувача відповідно до методології тестування. Зокрема, було застосовано такі підходи, як функціональне тестування, інтеграційне тестування та тестування методом "чорного ящика". Кожен з методів дозволив оцінити різні аспекти роботи системи: відповідність функцій вимогам, коректну взаємодію між модулями, а також зовнішню поведінку програми в очах користувача.

У результаті всіх перевірок не було виявлено критичних помилок або збоїв. Усі основні функції системи - включно з інтерфейсами меню, діалогами, переміщенням між локаціями та міні-іграми - працюють стабільно та відповідно до заданої логіки. Реакція системи на користувацький ввід є передбачуваною, інтуїтивною та відповідає ігровим стандартам.

Таким чином, можна зробити висновок, що створене програмне забезпечення відповідає вимогам технічного завдання, є працездатним, функціонально повним і готовим до використання кінцевими користувачами.

Загальний висновок до дипломної роботи

У межах даної дипломної роботи було повністю реалізовано повноцінну 2D гру в жанрі RPG з унікальним стилем та власною внутрішньою логікою. Головною метою проєкту було створення гри, яка зможе передати атмосферу, емоційність і глибину, притаманну справжнім авторським творам, а не типовим комерційним проєктам великих студій. Цей задум став відповіддю на сучасні тенденції в ігровій індустрії, де багато великих ігор втрачають унікальність через надмірну стандартизацію, шаблонність сюжетів та орієнтацію виключно на графіку.

Під час дослідження предметної області було розглянуто як стан сучасної ігрової індустрії, так і діяльність інді-розробників, які довели, що навіть одна людина здатна створити гру, яка запам’ятається мільйонам. Такі імена, як Тобі Фокс, Лукас Поуп або Скот Коутон стали натхненням для реалізації власного авторського проєкту. Було також виконано аналіз рушіїв для розробки ігор, що дозволило обґрунтувати вибір саме Unity - завдяки його кросплатформеності, зручності у використанні, широкій спільноті та великому набору інструментів для 2D-ігор.

У процесі реалізації гри було створено 13 взаємодіючих між собою скриптів, які відповідають за керування персонажем, взаємодію з об’єктами, переміщення між зонами, логіку діалогів, мініігри та роботу інтерфейсів. Було також розроблено адаптивну систему інтерфейсів, яка з’являється лише в момент необхідності (діалоги, пауза, мінігра), що робить інтерфейс ненав’язливим і дозволяє повністю зануритись у гру.

Додаткову увагу приділено модульності та розширюваності проєкту. Код написаний таким чином, щоб гру можна було легко доповнювати новими зонами, персонажами, діалогами або мініграми. Особливістю проєкту є також використання двох способів організації локацій - через окремі сцени та поділ однієї сцени на зони.

Окремим етапом стало тестування гри, під час якого застосовано функціональне, інтеграційне тестування та метод "чорного ящика". Кожна з основних підсистем була перевірена на наявність помилок та логічних конфліктів, і в результаті жодних критичних помилок виявлено не було.

У підсумку, можна зробити висновок, що гра повністю реалізована згідно з поставленими завданнями. Вона є стабільною, функціонально повною та придатною до подальшого розширення. Проєкт також демонструє, як сучасні інструменти розробки дозволяють індивідуальному розробнику створити якісний продукт, що має потенціал для вдосконалення, релізу або навіть комерційного розповсюдження.

Список використаних джерел

1. Unity Technologies – Офіційна документація [Електронний ресурс] URL - https://docs.unity3d.com/
2. Unreal Engine – Офіційна документація [Електронний ресурс] URL - https://docs.unrealengine.com/
3. GameMaker – Офіційна документація [Електронний ресурс] URL - https://manual.yoyogames.com/
4. Масштаб Геймдеву на момент 2024-2026 років [Електронний ресурс] URL - https://newzoo.com/resources/blog/global-games-market-revenue-estimates-and-forecasts-in-2024
5. Інді студії та їх успіх [Електронний ресурс] URL - https://indigomusic.com/feature/indie-giants-success-stories-from-undertale-to-hollow-knight
6. Майбутне геймдеву [Електронний ресурс] URL - https://www.globenewswire.com/news-release/2024/08/02/2923707/0/en/Video-Games-Market-Size-Expected-to-Reach-USD-664-96-Billion-By-2033.html
7. Тобі фокс та його успіх [Електронний ресурс] URL - https://gamerant.com/undertale-creator-toby-fox-success-deltarune-nintendo-pokemon-smash-bros/
8. Wikipedia – Software Testing – оглядова енциклопедична стаття про тестування [Електронний ресурс] URL - https://en.wikipedia.org/wiki/Software_testing

Додаток А. Фрагмент лістингу гри

Скрипт PlayerController .cs

```
using UnityEngine;

public class PlayerController : MonoBehaviour {
    public float moveSpeed = 5f;//Швидкісь Ігрока
    private Rigidbody2D rb;//фізичне тіло ігрока
    private Vector2 movement;//Координати Ігрока
    private Animator animator;//Аніматор

    public NPCDialogueController dt;//Контролер Діалогів
    public VectorValue pos;//Координати ігрока

    [SerializeField] Animator MenuAnimator; // Аніматор Паузи
    [Header("Footstep Audio")]
    public AudioSource footstepSource;// Об'єкт з ланаштугками звуків
    public AudioClip grassSteps;//Аудіо файл Трави
    public AudioClip stoneSteps;//Аудіо Файл Камню

    void Start() {
        transform.position = pos.initialValue;//Загрузка обьектів та функцій
при запуску
        animator = GetComponent<Animator>();//
        rb = GetComponent<Rigidbody2D>();//

        if (footstepSource != null)//
            footstepSource.loop = true;//
    }

    public void HandleUpdate() {
        float moveX = Input.GetAxisRaw("Horizontal");//Отримання координат
ігрока
        float moveY = Input.GetAxisRaw("Vertical");//

        movement = new Vector2(moveX, moveY).normalized;// Налаштунок
швидкості при діагональній ходьбі

        if (movement.magnitude > 0) {//Реакція аніматора ігрока на рух.
            animator.SetBool("isMoving", true);
            animator.SetFloat("moveX", moveX);
            animator.SetFloat("moveY", moveY);

            if (footstepSource != null) {
                AudioClip surfaceClip = GetSurfaceClip();//Первірка поверхні
на якій ходить гравець

                if (surfaceClip != null && footstepSource.clip !=
surfaceClip) {
                    footstepSource.clip = surfaceClip;
                    footstepSource.Play();
```

```
                }

                if (!footstepSource.isPlaying)
                    footstepSource.Play();
            }
        }
        else {
            animator.SetBool("isMoving", false);//Реакція аніматора но
відстутність руху

            if (footstepSource != null && footstepSource.isPlaying)
{//Первірка поверхні на якій стоіть гравець
                footstepSource.Stop();
            }
        }

        if (Input.GetKeyDown(KeyCode.E)) {//виклик функції Interact
            Interact();
        }

        if (Input.GetKeyDown(KeyCode.Escape)) {//Виклик Паузи
            MenuAnimator.SetBool("Open", true);
            Time.timeScale = 0;
        }
    }


    void FixedUpdate() {
        rb.velocity = movement * moveSpeed;
    }

    void Interact() {//Функція визову функції Interact в інших об'єктах
        var facingDir = new Vector3(animator.GetFloat("moveX"),
animator.GetFloat("moveY"));//Отриманя сторони в яку дивиться ігрок
        var interactPos = transform.position + facingDir;

        var collider = Physics2D.OverlapCircle(interactPos, 0.2f,
LayerMask.GetMask("interactable"));//Пошук обьекта з тегом Interactable

        if (collider != null) {
            var dialogTrigger =
collider.GetComponent<NPCDialogueController>();
            collider.GetComponent<interactable>()?.Interact();//Виклик
функціі Interact
        }
    }


    AudioClip GetSurfaceClip() { //Функція перевірки поверхні
        Vector2 position = transform.position;
        RaycastHit2D hit = Physics2D.Raycast(position, Vector2.down, 0.1f);

        if (hit.collider != null) {
```

```
            int layer = hit.collider.gameObject.layer;
            string layerName = LayerMask.LayerToName(layer);
            Debug.Log("Standing on: " + layerName);
            switch (layerName) {
                case "Grass": return grassSteps;
                case "Stone": return stoneSteps;
                default: return null;
            }
        }
        else {
            Debug.Log("Raycast hit nothing");
        }

        return null;
    }

    void OnApplicationQuit() {//Відкат позиції ігрока при виході з гри
        pos.initialValue = pos.defaultValue;
    }
}
```

Скрипт GameController .cs

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

public enum GameState {FreeRoam, Dialog, Battle}//Стани ігрока
public class GameControler : MonoBehaviour
{
    [SerializeField] PlayerController playerControler;
    [SerializeField] GameObject soundsObject;

    GameState state;

    private void Start() {

        if (SoundChoice.Instance != null && soundsObject != null)
{//Вмикання/Вимикання звуку
            soundsObject.SetActive(SoundChoice.Instance.isSoundOn);
        }

        DialogManager.Instance.OnShowDialog += () => {//Перемикання стану
            state = GameState.Dialog;
        };
        DialogManager.Instance.OnHideDialog += () => {//Перемикання стану
            if(state==GameState.Dialog)
            state = GameState.FreeRoam;
        };
```

```
    }

    private void Update() { //Перемикання стану
        if (state == GameState.FreeRoam) {

            playerControler.HandleUpdate();

        }else if ( state==GameState.Dialog)
        {
            DialogManager.Instance.HandleUpdate();
        }else if (state == GameState.Battle) {

        }
    }

}
```

Скрипт MenuController .cs

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

public class MenuControler : MonoBehaviour
{
    [SerializeField] Animator SoundAnimator;//Аніматор кнопки "Звуку"
    [SerializeField] Animator FadeAnimaton;//Аніматор появи та зникнення меню
    [SerializeField] AudioSource LaderSound;//Звук переходу до гри
    [SerializeField] Animator MenuAnimaton;//Анімації меню
    public void GameStart() {//Фуенція натиску на "Почати"
        LaderSound.Play();
        FadeAnimaton.SetBool("Fade", true);
    }

    public void ExitGame() {//Функція натиску на "Вийти"

#if UNITY_EDITOR
        UnityEditor.EditorApplication.isPlaying = false;
#else
        Application.Quit();
#endif



    }
    public void Resume() {//Функція натиску на "продовжити"
        MenuAnimaton.SetBool("Open", false);
        Time.timeScale = 1;
```

```
    }

    public void SoundOn() {//Функція перемикання кнопки "Звук"
        SoundAnimator.SetBool("Sound",true);
        LaderSound.Play();
    }

    public void SoundOff() {//Функція перемикання кнопки "Звук"
        SoundAnimator.SetBool("Sound", false);
        LaderSound.Play();
    }

    //  Функції визиваются через Animation Event в анимації "выключено" та
"Включено"
    public void SetSoundChoiceTrue() {//Функція перемикання Звуку
        if (SoundChoice.Instance != null)
            SoundChoice.Instance.isSoundOn = true;
    }

    public void SetSoundChoiceFalse() {//Функція перемикання Звуку
        if (SoundChoice.Instance != null)
            SoundChoice.Instance.isSoundOn = false;
    }
}
```

Скрипт DialogManager .cs

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

public class DialogManager : MonoBehaviour {
    [SerializeField] Animator dialogAnimator;//Аніматор Діалогового окна
    [SerializeField] Text dialogText;//Анваатор тексту

    [SerializeField] int lettersPerSecond = 20;//Швидкість написання тексту

    public event Action OnShowDialog;//Івенти при початку
    public event Action OnHideDialog;//та кінці діалогу

    public static DialogManager Instance { get; private set; }

    private void Awake() {
        Instance = this;
    }

    List<string> currentLines;
    int currentLine;
```

```
    bool isTyping;

    public IEnumerator ShowDialog(List<string> lines) {//Функція початку
ділогу
        yield return new WaitForEndOfFrame();
        OnShowDialog?.Invoke();

        currentLines = lines;
        currentLine = 0;

        dialogAnimator.SetBool("DialogOpen", true);
        StartCoroutine(TypeDialog(currentLines[currentLine]));
    }

    public void HandleUpdate() {//Перемикання фраз діалогу
        if (Input.GetKeyDown(KeyCode.E) && !isTyping && currentLines != null)
{
            currentLine++;
            if (currentLine < currentLines.Count) {
                StartCoroutine(TypeDialog(currentLines[currentLine]));
            }
            else {
                EndDialog();
            }
        }
    }

    IEnumerator TypeDialog(string line) {//функція другу тексту
        isTyping = true;
        dialogText.text = "";

        foreach (char letter in line.ToCharArray()) {
            dialogText.text += letter;
            yield return new WaitForSeconds(1f / lettersPerSecond);
        }

        isTyping = false;
    }

    void EndDialog() {//функція вимикання ділаогу
        dialogAnimator.SetBool("DialogOpen", false);
        currentLines = null;
        currentLine = 0;
        OnHideDialog?.Invoke();
    }
}
```

Скрипт MiniGamemanager .cs

```
using System.Collections;
using UnityEngine;
using UnityEngine.UI;
```

```
public class MiniGameManager : MonoBehaviour {
    [SerializeField] Animator fieldAnimator; // Анимація вікна
    [SerializeField] Animator handsAnimator; // Анимація рук

    [SerializeField] Text resultText;//Текст з результаом

    enum Choice { Rock, Paper, Scissors }//Можливі вибори

    public void StartMiniGame() {//Функція початку міні-ігри
        // Открываем окно (UI)
        fieldAnimator.SetBool("GameIsStarted", true);

        // Через секунду запускаем анимацию рук
        StartCoroutine(ShowHands());
    }

    private IEnumerator ShowHands() {//Анімація появи рук
        yield return new WaitForSeconds(0.5f);
        handsAnimator.SetBool("Game-Started", true);
    }

    public void CloseMiniGame() {//функція закриття міні-ігри
        handsAnimator.SetInteger("Choice", 0);
        // Закрываем руки
        handsAnimator.SetBool("Game-Started", false);

        // Закрываем окно
        fieldAnimator.SetBool("GameIsStarted", false);
    }

    // Вызываются кнопками
    public void PlayerChooseRock() {//Функція обирання Камню
        StartCoroutine(PlayRound(Choice.Rock));
    }

    public void PlayerChoosePaper() {
        StartCoroutine(PlayRound(Choice.Paper));//Функція обирання бумаги
    }

    public void PlayerChooseScissors() {
        StartCoroutine(PlayRound(Choice.Scissors));//Функція обирання Ножиць
    }

    private IEnumerator PlayRound(Choice playerChoice) {
        // Встановлюємо вибір ігрока
        int playerAnimValue = GetAnimValue(playerChoice);
        handsAnimator.SetInteger("Choice", playerAnimValue);

        // Генерируем вибір Ворога
        Choice enemyChoice = (Choice)Random.Range(0, 3);
        int enemyAnimValue = GetAnimValue(enemyChoice);
```

```
        handsAnimator.SetInteger("EnemyChoice", enemyAnimValue);

        // Чекаємо, щоб анимації синхронно запустились
        yield return new WaitForSeconds(1.5f); // зв'язано з довжиною
анимації

        // Показываем результат
        string result = DetermineWinner(playerChoice,
enemyChoice);//Розрахунок переможця
        resultText.text = result;
    }

    private int GetAnimValue(Choice choice) {
        // Преобразуем вибір в значення, яке розуміе аниматор
        switch (choice) {
            case Choice.Rock: return 1;
            case Choice.Paper: return 3;
            case Choice.Scissors: return 2;
            default: return 0;
        }
    }

    private string DetermineWinner(Choice player, Choice enemy) {//Функція
розрахунку переможця

        if (player == enemy)
            return "Draw!";
        else if ((player == Choice.Rock && enemy == Choice.Scissors) ||
                 (player == Choice.Scissors && enemy == Choice.Paper) ||
                 (player == Choice.Paper && enemy == Choice.Rock))
            return "You Win!";
        else
            return "You Lose!";
    }

    private void ResetHandAnimation() {//Функція ресету анімації рук
        handsAnimator.SetInteger("Choice", 0);
        handsAnimator.SetInteger("EnemyChoice", 0);
    }

}
```

Скрипт DoorController .cs

```
using System.Collections;
using UnityEngine;
using UnityEngine.SceneManagement;

public class DoorController : MonoBehaviour, interactable {

    public int LevelToLoad; // Індекс сцены, на яку буде перехід
```

```
    public Vector3 position; // Координаты ігрока після перехода
    public VectorValue playerStorage; // Хранилище позиції ігрока
    public AudioSource LaderSound;



    public void Interact() {//Функція Interact


            OnFadeComplete();
            LaderSound.Play();

    }



    public void OnFadeComplete() {//Функця зберігання та відправки Координат
ігрока
        playerStorage.initialValue = position; // Запоминаем позицию игрока
        SceneManager.LoadScene(LevelToLoad); // Загружаем новую сцену
    }
}
```

Скрипт FrameSwitch .cs

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

public class FrameSwitch : MonoBehaviour {
    public GameObject activeFrame;

    private void OnTriggerEnter2D(Collider2D other) {//Функція яка вмикае
зону при вході в неї
        if (other.CompareTag("Player")) {
            activeFrame.SetActive(true);


        }
    }

    private void OnTriggerExit2D(Collider2D other) {//Функція яка вимикае
зону при виході з неї
        if (other.CompareTag("Player")) {
            activeFrame.SetActive(false);


        }
    }
}
```

Скрипт SoundChoice .cs

```
using UnityEngine;

public class SoundSettingsHolder : MonoBehaviour {
    public static SoundSettingsHolder Instance;

    public bool isSoundOn = true;

    private void Awake() {
        if (Instance == null) {
            Instance = this;
            DontDestroyOnLoad(gameObject); // не знищується між сценами
        }
        else {
            Destroy(gameObject); // захист від дублікатів
        }
    }

    public void SetSound(bool enabled) {//Установка звуку
        isSoundOn = enabled;
    }

    public bool GetSound() {//Отримання рішеня
        return isSoundOn;
    }
}
```

Скрипт MultiDialogData .cs

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

[CreateAssetMenu(fileName = "DialogData", menuName = "Dialogues/Multi Dialog
Data")]//Створення об'єкта з діалогами
public class MultiDialogData : ScriptableObject {
    public List<DialogEntry> dialogs;//Списко діалогів
}

[System.Serializable]
public class DialogEntry {
    public string id;//назва списку
    public List<string> lines;//Строки діалогу
}
```

Скрипт NPCDialogueController .cs

```
using UnityEngine;

public class NPCDialogueController : MonoBehaviour, interactable {
    public MultiDialogData dialogueData;
    public string dialogueId;

    public void Interact() {

        var entry = dialogueData.dialogs.Find(d => d.id ==
dialogueId);//Пошук діалгогу
        if (entry == null) {//Якщо неіснуе діалогу з данною назвою
            Debug.LogWarning($"Диалог с ID {dialogueId} не
найден");//Повідомлення про помилку
            return;
        }


StartCoroutine(DialogManager.Instance.ShowDialog(entry.lines));//Виклик
потрібного діалогу
    }
}
```

Скрипт Enemy .cs

```
using System.Collections;
using UnityEngine;

public class Enemy : MonoBehaviour, interactable {
    [SerializeField] MiniGameManager miniGameManager;

    public void Interact() {
        miniGameManager.StartMiniGame();
    }
}
```

Скрипт Vectorvalue .cs

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

[CreateAssetMenu]
public class VectorValue : ScriptableObject
{
    public Vector3 initialValue;
    public Vector3 defaultValue;

}
```

Скрипт Interactable .cs

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

public interface interactable
{
    void Interact();
}
```