ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
«УНІВЕРСИТЕТ ЕКОНОМІКИ ТА ПРАВА «КРОК»

КВАЛІФІКАЦІЙНА РОБОТА

Тема: «Комп'ютерна гра «Lost Element» 2D платформер із сайтом вікіпедії та використанням алгоритмів генерації залів, інтелектуальної поведінки ботів (комплексна робота)»

Ступінь вищої освіти - бакалавр
Спеціальність – 122 «Комп'ютерні науки»
Освітня програма «Комп'ютерні науки»

ПОЯСНЮВАЛЬНА ЗАПИСКА

Виконав: здобувач 4 курсу
групи КН-21
Максим ІВАНОВ
Керівник: зав. кафедри комп'ютерних наук
к.е.н., с.н.с, доцент
Сергій МІЧКІВСЬКИЙ

Засвідчую, що кваліфікаційна робота оформлена відповідно до ДСТУ 3008:2015 та не містить запозичень з праць інших авторів без відповідних посилань.
Здобувач: _____
(підпис)

м. Київ – 2025 рік

ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД

«УНІВЕРСИТЕТ ЕКОНОМІКИ ТА ПРАВА «КРОК»

ЗАТВЕРДЖУЮ:
завідувач кафедри
комп’ютерних наук
_____Сергій МІЧКІВСЬКИЙ
«____»____20___р

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Іванов Максим Сергійович

Тема роботи	Комп'ютерна гра "Lost Element" 2D платформер із сайтом вікіпедії та використанням алгоритмів генерації залів, інтелектуальної поведінки ботів (комплексна робота)
Номер та дата наказу про затвердження теми	№121-7 від 24 грудня 2024 року
Коротка постановка завдання	Розробити 2D комп’ютерну гру, платформер з використанням алгоритмів генерації залів, інтелектуальної поведінки ботів, а також UI/UX для гри; Розробити сайт вікіпедію з внутрішнім магазином, а також UI/UX для сайту.
Посилання на джерела інформації (не більше п'яти найменувань, які рекомендує науковий керівник)	1. Моделювання поведінки неігрових персонажів у комп'ютерних іграх : Н.О. СОКОЛОВА, В.В.ГНАТУШЕНКО, М.С. МІЩЕНКО, О.А. АТАМАНЧУК, Національний технічний університет «Дніпровська політехніка» 2. Чеботарьова І. Б., Черкашина Г. І. Основні тренди UI/UX дизайну 2024. Поліграфічні, мультимедійні та web-технології : матеріали Молодіжної школи-семінару IX Міжнар. наук.-техн. конф., 14-28 травня 2024 р. – Харків : ТОВ «Друкарня Мадрид», 2024. – Т. 2. – С. 40-47
Вимоги до кваліфікаційної роботи	Кваліфікаційна робота має передбачити теоретичне, системотехнічне або експериментальне дослідження складного спеціалізованого завдання або практичної проблеми в галузі комп’ютерних наук, яке характеризується комплексністю та невизначеністю умов і потребує застосування теорій і методів інформаційних технологій.

Дата видачі завдання 27 грудня 2024 р

Керівник Сергій МІЧКІВСЬКИЙ

Здобувач освітнього ступеня бакалавра Максим ІВАНОВ

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання	Примітка
	Підготовчий етап		
1	Вибір напряму дослідження	02.12.2024 р.	*виконано*
2	Формування теми та призначення керівника	16.12.2024 р.	*виконано*
3	Затвердження теми кваліфікаційної роботи	23.12.2024 р.	*виконано*
4	Затвердження завдання на кваліфікаційну роботу	27.12.2024 р.	*виконано*
	Основний етап		
5	Розробка концепції кваліфікаційної роботи	13.01.2025 р.	*виконано*
6	Підбір та вивчення джерел інформації з напряму дослідження. Огляд існуючих аналогів	20.01.2025 р.	*виконано*
7	Затвердження розширеної постановки завдання. Підготовка та подання керівникові розділу 1 кваліфікаційної роботи	10.03.2025 р.	*виконано*
8	Проєктування. Підготовка та подання керівникові розділу 2 кваліфікаційної роботи	24.03.2025 р.	*виконано*
9	Підготовка доповіді для експертизи стану виконання кваліфікаційної роботи (проміжний контроль)	31.03-04.04.2025 р.	*виконано*
10	Реалізація. Підготовка та подання керівникові розділу 3 кваліфікаційної роботи	07.04.2025 р.	*виконано*
11	Підготовка та подання керівнику першого варіанту всієї кваліфікаційної роботи	14.04.2025 р.	*виконано*
12	Доопрацювання кваліфікаційної роботи з урахуванням зауважень керівника та представлення керівникові доопрацьованого варіанту кваліфікаційної роботи	21.04.2025 р.	*виконано*
	Завершальний етап		
13	Представлення рукопису для перевірки на плагіат	28.04-04.05.2025 р.	*виконано*
14	Підготовка презентації та доповіді на передзахист	05.05-11.05.2025 р.	*виконано*
15	Передзахист кваліфікаційної роботи	12.05-16.05.2025 р.	*виконано*
16	Доопрацювання роботи за результатами передзахисту	19.05-06.06.2025 р.	*виконано*
17	Експертиза роботи керівником та зовнішнім експертом	09.06-15.06.2025 р.	*виконано*
18	Доопрацювання доповіді та презентації для захисту	09.06-15.06.2025 р.	*виконано*
19	Захист кваліфікаційної роботи	16.06-22.06.2025 р.	*виконано*

Керівник Сергій МІЧКІВСЬКИЙ

Здобувач освітнього ступеня бакалавра Максим ІВАНОВ

Іванов М.С. Комп'ютерна гра «Lost Element» (2D платформер із сайтом вікіпедії та використанням алгоритмів генерації залів, інтелектуальної поведінки ботів; комплексна робота).

Пояснювальна записка кваліфікаційної роботи за спеціальністю 122 - Комп'ютерні науки (освітня програма - Комп'ютерні науки) СО Бакалавр. - ВНЗ "Університет економіки та права "КРОК", Навчально-науковий інститут інформаційних та комунікаційних технологій, кафедра комп'ютерних наук, Київ, 2025.

Розроблено комп'ютерну гру «Lost Element» з елементами головоломок, яка поєднує сюжетну глибину, оригінальні ігрові механіки та привабливий візуальний стиль.

Ключові слова: Lost Element, відеогра, розробка гри, стратегія, платформер

Рис. 14 Бібліограф.: 25 найм.

Ivanov M.S. Computer game "Lost Element" 2D platformer with a Wikipedia site and the use of algorithms for generating halls, intelligent behavior of bots (complex work)

Project explanatory note by specialty 122 - Computer science. - "KROK" University, Educational and Scientific Institute of information and communication technologies, Department of Computer Science, Kyiv, 2025.

A computer game "Lost Element" with puzzle elements has been developed, combining plot depth, original game mechanics, and an attractive visual style.

Keywords: Lost Element, video game, game development, strategy, platformer

Fig. 14. Bibliography: 25 Items.

ЗМІСТ

ВСТУП

Актуальність теми. Комп'ютерні ігри продовжують залишатися однією з найпопулярніших форм цифрових розваг. З кожним роком ринок відеоігор зростає, охоплюючи дедалі ширшу аудиторію. Сучасні гравці очікують не лише якісної графіки, а й глибокого сюжету, унікальної атмосфери та нових ігрових механік. Особливу нішу займають пригодницькі ігри з елементами головоломок та дослідження світу – саме такий жанр обирають користувачі, які цінують занурення в атмосферу, нестандартні рішення та відчуття відкриття.

Гра *Lost Element* створюється як інноваційний проєкт, що поєднує пригодницький жанр із містичними елементами, головоломками та дослідженням загадкового світу. Основна ідея полягає у пошуках втраченого стихійного артефакту, від якого залежить доля всього ігрового світу. Гра розрахована на одного гравця та буде доступною на персональному комп’ютері з можливістю портування на мобільні платформи.

Метою даного проєкту є створення пригодницької комп’ютерної гри *Lost Element* з елементами головоломок, яка б поєднувала сюжетну глибину, оригінальні ігрові механіки та привабливий візуальний стиль.

Завдання дослідження:

- провести огляд сучасного ринку пригодницьких ігор, включно з казуальними, головоломками та сюжетними проєктами;
- обрати технологічний стек для реалізації гри: мову програмування, рушій, стиль візуалізації (2D або 3D);
- розробити структуру застосунку, реалізувати геймплей, головоломки, взаємодію з об'єктами та сюжетні події;
- створити зручний, доступний і візуально привабливий інтерфейс користувача;
- провести тестування гри для виявлення та усунення можливих помилок.

Об'єктом дослідження є ігровий процес пригодницької комп’ютерної гри з нелінійним сюжетом і головоломками.

Предмет дослідження механіки взаємодії гравця з ігровим середовищем, реалізація інтелектуальних систем для оповіді, а також дизайн світу та інтерактивних елементів.

Методами дослідження аналіз ігрових рушіїв, алгоритмів поведінки персонажів у сюжетних іграх, розробка сценаріїв, побудова дерева діалогів та ігрової логіки.

Практичне значення розроблена гра Lost Element дозволяє збагатити ринок незалежних пригодницьких ігор, пропонуючи гравцям унікальне поєднання дослідження, логічних викликів та сюжету. Застосування сучасних алгоритмів та інструментів відкриває можливість подальшого розвитку гри та її портативності на інші платформи.

Структура роботи. Кваліфікаційна робота складається зі вступу, трьох розділів, висновків та списку посилань (25 найменувань). Пояснювальна записка містить 14 рисунків. Загальний обсяг пояснювальної записки становить 49 сторінок, основний зміст викладено на 47 сторінках.

РОЗДІЛ 1
ПОСТАНОВКА ЗАВДАННЯ НА РОЗРОБКУ

1.1 Предметна область

2D платформери – це жанр відеоігор, у яких гравець керує персонажем, що рухається по рівнях, долаючи перешкоди, стрибаючи, збираючи об'єкти та виконуючи інші завдання. Ігри цього жанру характеризуються чітким поділом на рівні або етапи, де кожен наступний етап зазвичай складніший за попередній, що вимагає від гравця точного контролю та стратегічного планування.

Основною механікою 2D платформерів є переміщення персонажа в горизонтальній або вертикальній площині, де важливим елементом є вміння правильно управляти рухами персонажа для успішного проходження рівнів. Це може включати стрибки, біг, ковзання та інші дії, необхідні для подолання перешкод. Жанр відомий своєю доступністю, оскільки ігрові механіки зазвичай є інтуїтивно зрозумілими та не вимагають складних навичок.

2D платформери стали популярними ще на ранніх етапах розвитку індустрії відеоігор і продовжують зберігати свою привабливість завдяки простоті та ефективності у використанні технологій. Вони не вимагають великих апаратних ресурсів для реалізації, що робить цей жанр доступним як для великих студій, так і для інді-розробників. У цьому жанрі також часто застосовуються цікаві художні стилі та звукові рішення, що надає їм особливого шарму [1].

Ігрова індустрія є однією з найпотужніших і швидко зростаючих галузей у сфері розваг. Відеоігри вже давно вийшли за межі простої розваги – це спосіб комунікації, мистецтво, і навіть освітній інструмент. Популярність незалежних студій (indie-розробників) зросла завдяки зручним рушіям (Unity, Unreal Engine) та онлайн-платформам поширення (Steam,Epic Games, itch.io). (рис. 1.1) [1] Саме в цьому просторі виростає попит на

атмосферні, сюжетні ігри, які не зосереджуються на екшні, а на глибокому досвіді – роздумах, емоціях, візуальній естетиці.

Рисунок 1.1 – Steam (онлайн платформа)

Джерело: [18].

З моменту свого виникнення 2D платформери зазнали значних змін, зокрема завдяки розвитку технологій. Раніше ігри цього жанру мали обмежену графіку і механіки, що диктувалося технологічними можливостями і обмеженнями апаратних засобів. Проте з кожним поколінням технологій платформи для створення ігор ставали дедалі потужнішими, що дозволило дизайнерам експериментувати з новими механіками та візуальними ефектами.

Однією з основних змін стало використання більш складних графічних технологій, таких як сучасні 2D рендерингові системи, що дозволяють створювати більш детальні фони, плавні анімації та ефекти, що раніше були недоступними. Наприклад, за допомогою новітніх графічних процесорів стало можливим досягати високої деталізації, створюючи враження об'ємності в 2D світах. Крім того, розвиток технологій дозволив інтегрувати фізичні та

реалістичні ефекти в ігровий процес, наприклад, динамічне освітлення, тіні, реалістичну поведінку об'єктів і персонажів.

Ці технологічні досягнення значно підвищили якість ігрового процесу та забезпечили нові можливості для творчого вираження розробників. Разом з цим, 2D платформери не втратили своїх класичних рис – вони залишаються простими і доступними, що дозволяє зберігати популярність жанру серед різних аудиторій, від новачків до досвідчених гравців.

Розвиток технологій також позитивно вплинув на доступність 2D платформерів на різних платформах, зокрема на мобільних пристроях, де використання потужних чіпів дозволяє запускати ігри з високим рівнем графіки без шкоди для продуктивності.

Жанр 2D платформерів залишається актуальним завдяки своїй універсальності та здатності адаптуватися до сучасних технологій і тенденцій у геймдизайні. Незважаючи на розвиток 3D графіки та складніших ігор, 2D платформери продовжують користуватись попитом через їх здатність забезпечити простий та інтуїтивно зрозумілий геймплей, що підходить як для новачків, так і для досвідчених гравців.

Важливим аспектом є те, що цей жанр залишається популярним серед інді-розробників, оскільки створення таких ігор не потребує великих ресурсів, а значить, вони можуть бути розроблені навіть на обмежених бюджетах. Крім того, 2D платформери відзначаються високим рівнем доступності для різних платформ, включаючи мобільні пристрої, що розширює їх аудиторію.

Таким чином, жанр 2D платформерів зберігає свою значущість у відеоігровій індустрії, продовжуючи залучати гравців завдяки своїй простоті, універсальності та можливості для творчих експериментів [3].

1.2 Огляд аналогів

Платформер – це жанр відеоігор, у якому головна механіка полягає у пересуванні персонажа по різних платформах з метою подолання перешкод, збору предметів або досягнення кінцевої цілі рівня. Основними елементами є

стрибки, ухилення, іноді – бойова система. Часто платформери будуються на рефлексах, точному таймінгу та механічній складності [6].

Типові характеристики платформера:

- двовимірне або тривимірне середовище (найчастіше – 2D);
- численні рівні з підвищенням складності;
- наявність ворогів або пасток;
- акцент на швидкість реакції та точність рухів;
- лінійна або напіввідкрита структура гри.

Огляд аналогів є невід'ємною частиною дослідження та розробки будь-якого проєкту в індустрії відеоігор, зокрема при створенні нової гри. Порівняння з іншими іграми дозволяє виявити як сильні, так і слабкі сторони існуючих рішень, а також забезпечує чітке розуміння поточних трендів, які домінують у геймдизайні. Це дозволяє розробникам уникнути повторення помилок попередників і визначити, яким чином можна зробити свою гру унікальною та конкурентоспроможною.

Аналіз аналогічних ігор дозволяє визначити сучасні технологічні та дизайн-тенденції. Це дає змогу зрозуміти, які механіки, графічні рішення та стилі взаємодії з гравцем є актуальними на поточний момент. Знання цих трендів допомагає розробникам вибрати відповідні інструменти для створення гри, що відповідає сучасним вимогам і залучає широку аудиторію.

Аналізуючи ігри, схожі на плановану, можна не лише виявити успішні елементи, але й вивчити проблеми, з якими зіткнулися розробники. Це дозволяє уникнути поширених помилок у дизайні, управлінні ресурсами чи оптимізації гри, а також допомагає сформулювати вимоги до програмування та тестування.

Знання сильних та слабких сторін аналогів дозволяє розробникам зосередитися на важливих аспектах гри та оптимізувати процес її створення. Це також допомагає створити чіткий план тестування, що дозволяє вчасно виявити й усунути недоліки до релізу гри, а також досягти високої якості продукту.

GRIS (Nomada Studio, 2018, рис. 1.2). – пригодницька гра, платформер, головоломка.

Особливості: стильна візуальна естетика, мінімалістичний інтерфейс, глибока емоційна історія без слів.

Схожість з "Lost Element": акцент на атмосфері, дослідженні та символізмі, художній підхід до візуального стилю.

Відмінність: відсутність чіткої логіки в головоломках, більше фокусу на художній інтерпретації, ніж на вирішенні задач.

Рисунок 1.2 – Скріншот з гри Gris

Джерело [20]

Little Nightmares (Tarsier Studios, 2017, рис. 1.3) – пригодницький хорор, платформер.

Особливості: похмурий світ, візуальний сюжет, головоломки, що пов'язані з оточенням.

Схожість з "Lost Element": сильна атмосфера, непряма подача сюжету, взаємодія з середовищем.

Відмінність: орієнтована на хорор і виживання, а не на пошук внутрішньої гармонії чи балансу, що є основою в "Lost Element".

Рисунок 1.3 – Скріншот з гри Little Nightmares

Джерело [21]

Journey (Thatgamecompany, 2012, рис. 1.4) – експериментальна пригода

Особливості: мінімалізм, мандрівка в символічному світі, музика як засіб розповіді.

Схожість з "Lost Element": безмовна, глибока атмосфера; шлях героя як символічний і внутрішній.

Відмінність: менше логічних завдань, більше фокус на естетичному досвіді, тоді як у "Lost Element" акцент на інтерактивності та головоломках.

Dead Cells (Motion Twin, 2018, рис. 1.5) – roguelike-платформер, екшен.

Особливості: динамічний геймплей, бій, прокачування, процедурна генерація

Схожість з "Lost Element": візуальний стиль з 2D-анімацією, абстрактне середовище

Відмінність: акцент на бої, швидкість і реакцію, тоді як у "Lost Element" основна увага приділяється розумінню світу та дослідженню, а не швидкій реакції

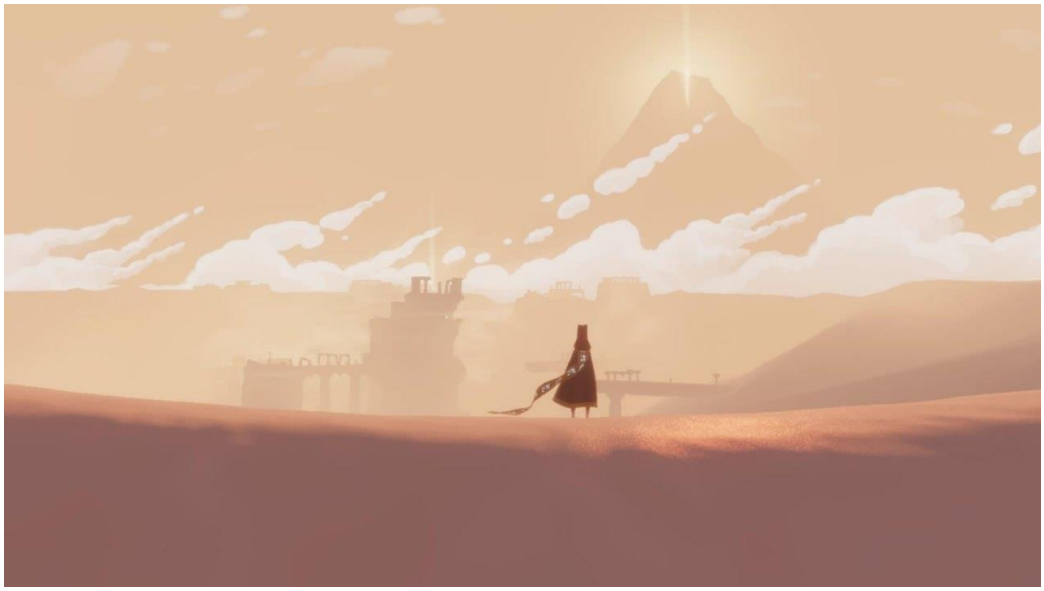

Рисунок 1.4 – Скріншот з гри Journey

Джерело [22]

Рисунок 1.5 – Скріншот з гри Dead Cells

Джерело [23]

Висновки з аналізу. На основі аналізу можна виділити кілька ключових напрямків, які вдало поєднує "Lost Element":

Від *GRIS* – художній підхід до візуального стилю та глибокий символізм.

Від *Little Nightmares* – використання атмосфери й візуального сторітелінгу.

Від *Journey* – філософська складова, без слів, але з сильною ідеєю.

І водночас, на відміну від Dead Cells, "Lost Element" не потребує швидкої реакції, а дає змогу вдумливо взаємодіяти зі світом.

Унікальні риси "Lost Element":

- поєднання чотирьох стихій у геймплейній механіці;
- розгалужений сюжет з варіативністю фіналу;
- складні головоломки, що інтегровані у світ, а не відокремлені від нього;
- повільний, але емоційно насичений темп гри.

1.3 Постановка задачі

Правила гри: – базові механіки: персонаж матиме базовий набір механік: ходьба, стрибки, біг та атаки ближнього бою; – дослідження локацій: гравець зможе самостійно обирати маршрути для проходження.

Локації будуть мати кімнати з загадками, які персонаж повинен вирішити щоб отримати нагороду. Також будуть присутні секретні кімнати, знайти їх можна досліджуючи локацію. – взаємодія: гравець зможе взаємодіяти з деякими предметами, будівлями, персонажами, тощо. – додаткові механіки: система збереження буде реалізована у вигляді точок з якими гравець має взаємодіяти.

При загибелі персонажу, його буде перенесено до останньої точки збереження. Учасниками гри люди віком від 15-35 років, до їхнього числа можуть входити як фанати ретро та платформерів, так і люди, які полюбляють дослідження та відкритий світ. При розробці планується використання автоматизованих функцій: – перехід анімацій персонажу; – оновлення карти при дослідженні; – генерація локацій при новій грі; – спавн ворогів у відповідних зонах; – поведінка ворогів: патрулювання, атака, переслідування; – збереження статусу ворогів – вбиті вороги не відроджуються, поки персонаж

не покине локацію; – спеціальні тригери подій для зустрічі з босом та запуску кат-сцени; – система діалогів.

Lost Element – це пригодницька однокористувацька комп'ютерна гра з елементами головоломок, дослідження світу та містичного наративу. Гравець опиняється у загадковому світі, що втратив баланс між чотирма первинними стихіями: вогнем, водою, повітрям і землею. Легенда говорить про існування п'ятого, забутого елементу – саме його гравець має віднайти, щоб врятувати світ від повного знищення.

Позначимо елементи гри в рисунках (рис. 1.6, рис. 1.7) яких користувач показував геймплейні відрізки.

Ігровий процес передбачає:

- дослідження атмосферного світу з численними локаціями (руїни, печери, стародавні храми, магічні ліси);
- розв'язання головоломок, що ґрунтуються на взаємодії зі стихіями;
- поступове відкриття сюжету через діалоги, візуальні підказки та знахідки;
- вибір, що впливає на розвиток історії та фінал гри.

Гравець не має явного "наставника" – тільки інтуїцію, фрагменти знань і власні рішення допоможуть знайти шлях до істини.

Опис жанру і гри

Lost Element (рис. 1.7) поєднує кілька жанрових рис:

Основний жанр: пригодницька гра (англ. Adventure)

Піджанр: головоломка/містичний квест (англ. Puzzle / Mystery Quest)

Гравець не просто рухається за сценарієм – він активно досліджує світ, вирішує головоломки, розкриває приховані таємниці й збирає фрагменти забутого знання. Важливу роль відіграє і візуальне оформлення – символізм, стародавні знаки, вплив кожної стихії на світ.

Рисунок 1.6 – Перша частина гри Lost Element

Джерело: розроблено автором

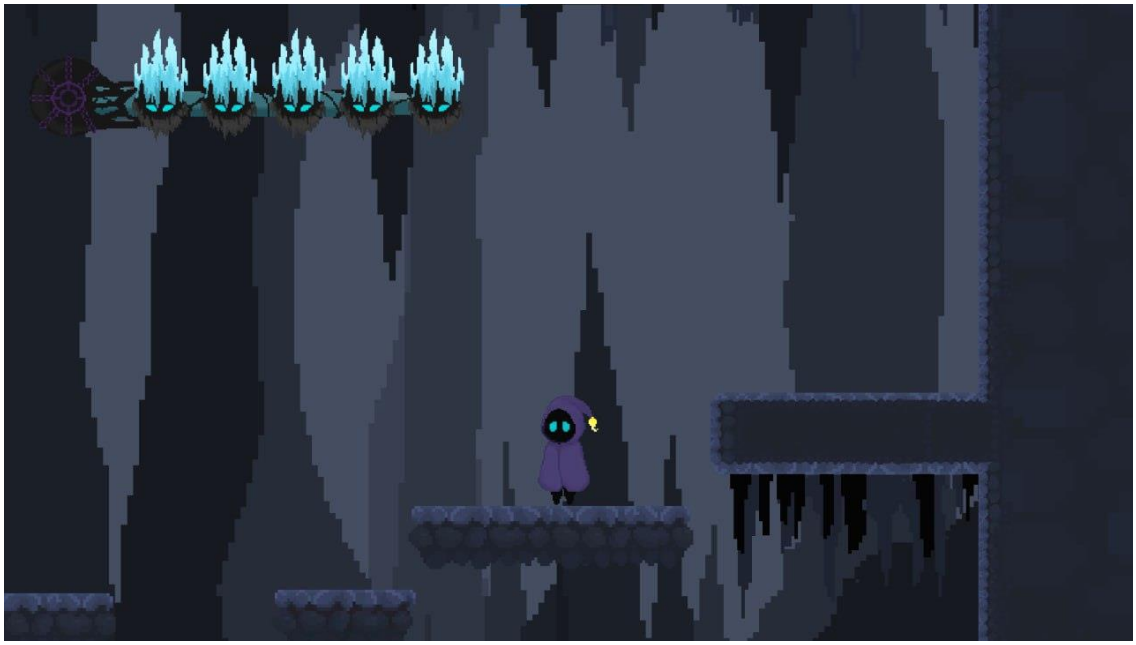

Рисунок 1.7 – Друга частина гри Lost Element

Джерело: розроблено автором

Причина вибору жанру:

- *інтерактивний сюжет* – дозволяє гравцеві глибоко зануритися в історію, розвивати емоційний зв’язок із подіями та середовищем;

- *поєднання логіки та інтуїції* – головоломки не тільки розважають, але й підсилюють атмосферу;
- *містика і дослідження* – сучасні гравці цінують глибокий, загадковий світ, у якому є що відкривати;
- *гнучкість реалізації* – гра чудово підходить як для ПК, так і для мобільних платформ;
- *інноваційність* – у грі можна поєднати елементи природи та логіки, створивши унікальні механіки (наприклад, вплив стихій на фізику об’єктів).

Висновки до розділу 1

У першому розділі була здійснена ґрунтовна оцінка основних аспектів розробки 2D платформерів, а також визначено ключові завдання для реалізації ігрового проєкту. У підрозділі 1.1 було описано жанр 2D платформерів як невід'ємну частину індустрії відеоігор, що зберігає свою популярність завдяки простоті механік, доступності та можливостям для творчих експериментів. Попри розвиток 3D технологій, 2D платформери продовжують бути актуальними через їх здатність забезпечити інтуїтивно зрозумілий геймплей і привернути широку аудиторію.

В підрозділі 1.2 був проведений огляд аналогічних ігор, де розглянуті різноманітні приклади проєктів, що мають спільні риси з майбутньою грою. Це дозволило виявити як успішні елементи, що підвищують якість гри, так і можливі слабкі сторони, яких слід уникати. Такий аналіз допомагає точніше визначити шляхи для вдосконалення геймплейного досвіду і зробити проєкт більш конкурентоспроможним.

У підрозділі 1.3 були сформульовані конкретні задачі для розробки гри, що включають інтеграцію унікальних механік, таких як використання чотирьох стихій, а також розробку складних головоломок і емоційно насиченого сюжету. Це дозволить створити продукт, який не лише

відповідатиме сучасним вимогам гравців, але й запропонує нові підходи до взаємодії з ігровим світом.

Таким чином, проведений аналіз жанру, огляд аналогів і постановка задач забезпечили чітке бачення основних напрямків для подальшої розробки гри. Розглянуті аспекти дозволяють визначити основні особливості проєкту та його унікальність, що є важливим етапом у створенні конкурентоспроможного та якісного продукту на сучасному ринку відеоігор.

РОЗДІЛ 2
ПРОЄКТУВАННЯ

2.1 Моделювання поведінки

Моделювання поведінки програмного продукту є суттєвим складовим процесу розробки програмного забезпечення. Воно не лише включає детальну документацію усіх можливих сценаріїв використання продукту, але також забезпечує чітку візуалізацію взаємодії користувача із системою. Визначаються два основних типу діаграм, які використовуються для моделювання поведінки: діаграму варіантів використання та діаграму діяльності.

Діаграма випадків використання (Use Case Diagram, або UCD) – це один з основних інструментів моделювання функціональних вимог до системи. Вона демонструє взаємодію між користувачами (акторами) та системою. Кожен актор виступає як представник зовнішнього користувача або іншої системи, яка взаємодіє з розробленим продуктом. Усі дії, які виконує система, структуровані в варіанти використання (use cases).

Діаграма варіантів використання забезпечує візуальне зображення основних функціональних характеристик продукту та методів взаємодії користувача із системою. Вона дозволяє зрозуміти, які конкретні дії будуть доступні для користувачів, а також які варіанти поведінки система повинна бути здатна підтримувати.

Значущість UCD забезпечує чітке визначення функцій, які система повинна виконувати, а також способів взаємодії користувачів з продуктом. Іншими словами, це модель, яка ілюструє, що конкретно буде здійснювати система для різних категорій користувачів. Використовуючи цю діаграму, можна оцінити, які функції можуть бути спрощені або автоматизовані. Це сприяє підвищенню зручності та ефективності системи. Діаграма варіантів застосування є важливим засобом, який допомагає розуміти роботу системи, а також дії, що виконуються в ній, та механізми взаємодії користувачів з

продуктом. Вона забезпечує можливість спланувати ефективний процес розробки та забезпечує комфорт кінцевому користувачеві [14].

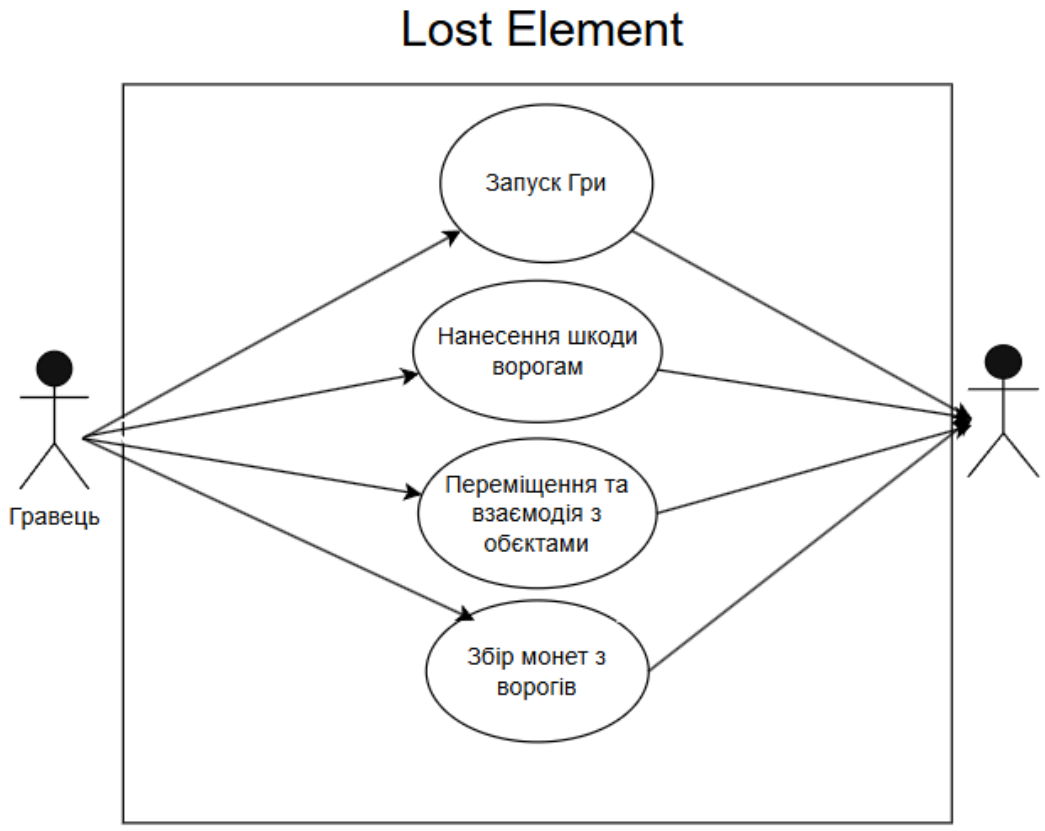

Рисунок 2.1 – Use Case Diagram (Гра)

Джерело: розроблено автором

Інструкція до рисунку 2.1:

- гравець – це основний актор , який виконує дії;
- стрілка – гравець задає дію;
- овал – заготовлена функція;
- стрілочка(до гри) – виконання функції , яку задав користувач;

Як у гравця є такі можливості: ми можемо зайти в головне меню, натискати на різні клавіші, через кнопку запустити рівень також можна переміщувати головного героя по рівню, можна використовувати зброю ближнього бою або вийти в голвне меню.

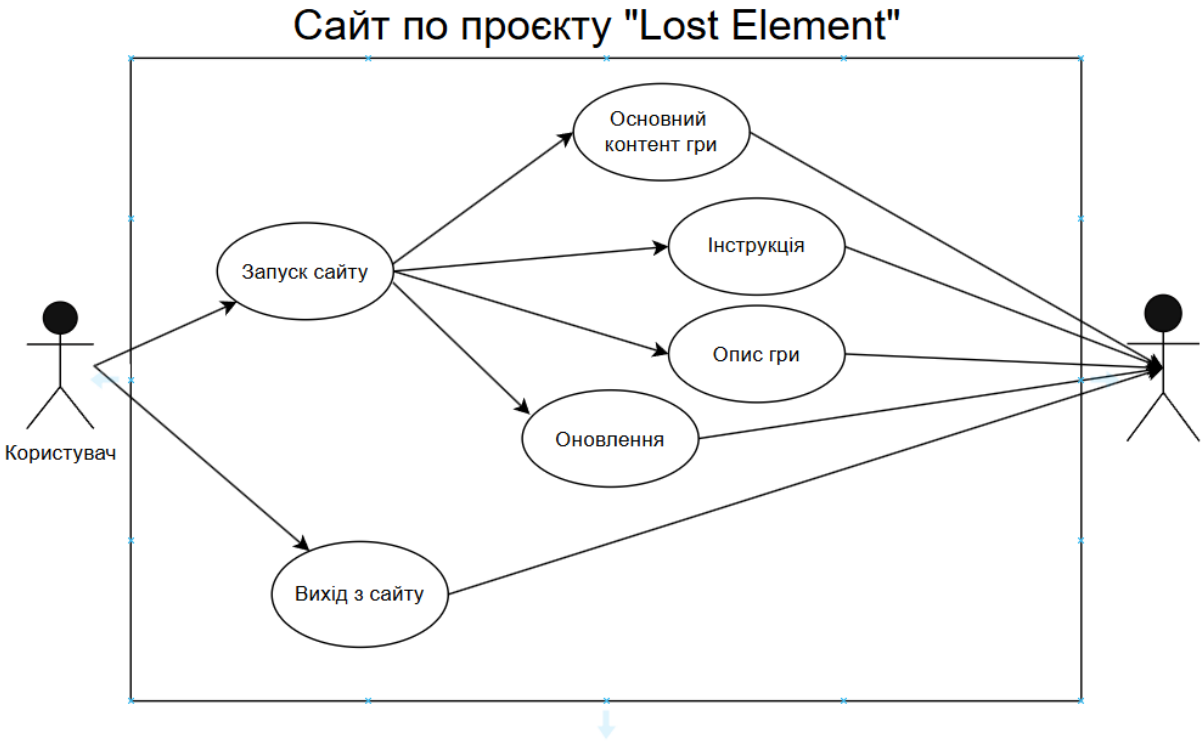

Рисунок 2.2 – Use Case Diagram (Сайт)

Джерело: розроблено автором

Інструкція до рисунку 2.2:

- користувач – це основний актор , який виконує дії;
- стрілка – відвідувач задає дію;
- овал – заготовлена функція;
- стрілочка(до сайту) – виконання функції , яку покаже сайт;

На сайті проєкту "Lost Element" користувач має наступні алгоритми взаємодії: він може зайти на сайт через кнопку "Запуск сайту", переглядати основний контент гри, який містить інструкцію та опис гри, а також дізнаватися про останні оновлення. Якщо необхідно, користувач може вийти зі сайту через кнопку "Вихід з сайту". Це дозволяє зручно навігувати між основними розділами сайту, отримувати інформацію про гру та оновлення.

*Activity Diagram (*діаграма діяльності) слугує інструментом для моделювання динаміки процесів в системі. Вона дозволяє продемонструвати покрокові операції чи активності, які виконуються в рамках одного варіанту

використання або в рамках цілого процесу. На відміну від діаграми варіантів використання, яка концентрується на взаємодії між користувачами і системою, діаграма діяльності фокусується на внутрішньому процесі, що відбувається в системі, ілюструючи, як інформація та дані переміщуються між етапами.

Діаграми діяльності зазвичай служать інструментом для ілюстрації складних бізнес-процесів, алгоритмів або специфічних операцій, що здійснюються в рамках продукту. Вони забезпечують можливість наочне відображення порядку дій, умов переходу між етапами та альтернативних сценаріїв розвитку подій.

Діаграма діяльності відображає черговість кроків, що виконуються протягом процесу, та залежності між ними, умови переходів та можливі варіанти розвитку ситуації. Вона особливо придатна для опису складних алгоритмів, бізнес-процесів і механізмів роботи програмних систем, де потрібно особливо чітко зрозуміти, яким чином здійснюються різні етапи виконання операцій.

У рамках нашої гри, діаграма діяльності демонструє процес, в якому користувач взаємодіє із системою, створюється рівень, проходяться етапи гри, а також оцінюється результат кожного з цих етапів.

Діаграма, представлена в цьому розділі, демонструє процес, який починається з головного меню гри. Користувач може вибрати варіант продовження гри, генерації нового рівня або початку гри спочатку. Залежно від дій користувача, система генерує нові рівні або обробляє перемогу чи програш. Процес може бути повторений до досягнення кінцевого результату, наприклад, завершення гри або досягнення певного рівня.

Критичним моментом цієї діаграми є її здатність розкрити потенційні сценарії розвитку подій на кожному рівні, зокрема перевірку стану учасника після закінчення кожного рівня та можливість перезапустити гру або потрапити на наступний рівень [16].

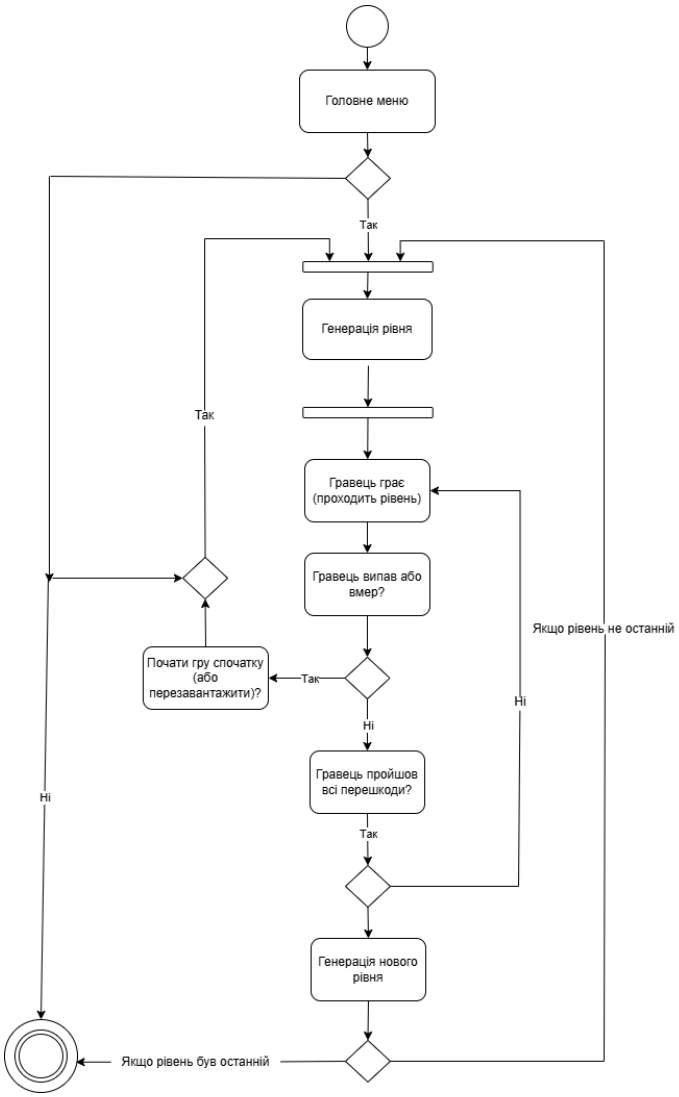

Рисунок 2.3 – Diagram Activity

Джерело: розроблено автором

Інструкція до рисунку 2.3:

- гравець – в діаграмі він позначений як коло;
- прямокутник – позначає функцію, яку виконує гравець;
- ромб – перевірка на дію (так чи ні?);
- стрілка – гравець задає дію;
- полукруг – кінцева точка гри;

Короткий опис діаграми:

При завантаженні гри з головного меню наш головний герой може виконувати декілька функцій:він може переміщуватись по рівню,вбивати вогорів які мають ші (штучниі інтелект),переходити між рівнями (з рандомною

генерацією їх), до деякиїх функцій які задає гравець можлива перевірка.І так до кінця поки не закінчиться гра.

2.2 Моделювання поведінки

Діаграма класів є одним з основних елементів моделювання структури програмного забезпечення, зокрема для 2D гри «Lost Element». Вона описує різні класи, їхні властивості (атрибути) та методи, а також взаємозв'язки між ними.

Класова діаграма (Class Diagram) в UML (Unified Modeling Language) відображає всі основні об'єкти гри, їх атрибути та методи. Вона допомагає розробникам зрозуміти, як класова структура взаємодіє в рамках програми. У нашому випадку ця діаграма демонструє основні компоненти гри, такі як гравець, вороги, менеджер кімнат та інші елементи, які складають функціональність гри.

Для нас ця діаграма класів є важливою частиною дипломної роботи, оскільки вона дозволяє наочно продемонструвати структуру нашої гри «Lost Element». Завдяки діаграмі класів ми можемо чітко показати, як організовані основні компоненти гри, як вони взаємодіють між собою, і як різні класи виконують свої функції в рамках загальної архітектури.

Діаграма класів дозволяє структурувати проєкт і розподілити відповідальність між класами. Це дуже важливо для розуміння того, як я реалізував різні механіки гри, такі як генерація рівнів, поведінка ворогів та інтерфейс користувача. Використання чіткої і логічної структури допомагає не лише забезпечити масштабованість і зручність у подальшій роботі, але й сприяє кращому розумінню проєкту як цілого [17].

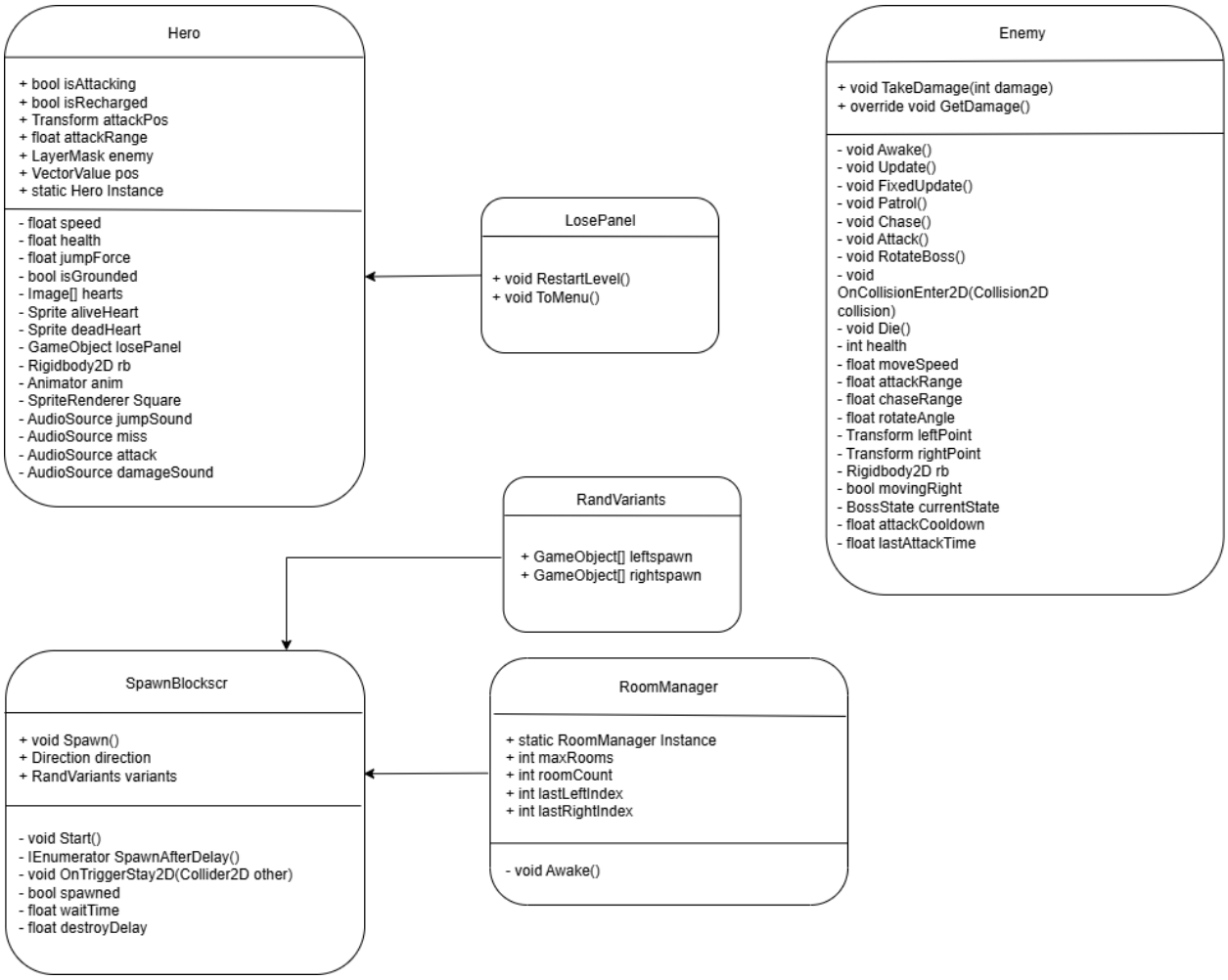

Рисунок 2.4 – Class Diagram

Джерело: розроблено автором

Інструкція до рисунку 2.4:

- квадрат – це функцію яку задає гравець;
- стрілка – це коди які прікліпленні до батьквіського об’єкта;
- + - це публічна змінна;
- - - це приватна змінна;

Короткий опис діаграми:

У нас є код героя до якого прикріпленний дочірний обєкт, який має свої публічна та приватні функціїї [LosePanel].Далі на рисунку можна побчати клас ворога (Enemy) в цьому класі є змінні , що відповідають за анімацію, переміщення, фізика та інші. В класі під назвою (SpawnBlocks) позначаємо генерацію блоків на рівні зі своїми перевірками та змінними. До нього також

прікрепленні дочірні об’єкти. Далі іде class (RoomManager) , який відповідає за обмеження генерації залів.

На діаграмі класів (Class Diagram) для «Lost Element» представлені основні класи, їхні властивості та методи, а також зв'язки між ними. Вона допомагає візуалізувати структуру гри та зрозуміти взаємодію об'єктів у програмі:

- *hero* має зв'язок з *LosePanel,* оскільки при смерті героя активується панель програшу;
- *hero* також взаємодіє з *Enemy*, оскільки гравець атакує ворогів і отримує від них пошкодження;
- *enemy* і *RoomManager* взаємодіють через генерацію кімнат, в яких можуть з'являтися вороги;
- *spawnBlockscr* взаємодіє з *RandVariants*, оскільки варіанти спауна визначають, що саме з'являється на рівні;
- *roomManager* управляє кімнатами, використовуючи спавн-блоки для їх розміщення.

Нижче приведу пояснення до кожної з показаних приватних та публічних класів докладніше таких як (Hero,Enemy, RandVariants, SpawnBlockscr, RoomManager*)*

Hero (Герой):

- static Hero Instance – об'єкт героя в сцені, який дає можливість іншим скриптам взаємодіяти з героєм (наприклад, змінювати його здоров'я чи перевіряти його стан);
- float speed – швидкість переміщення героя, яка впливає на його переміщення по сцені;
- float health – поточне здоров'я героя, яке можна змінювати в процесі гри.
- float jumpForce – сила стрибка героя, визначає, на яку висоту герой може стрибнути;
- bool isGrounded – перевірка, чи стоїть герой на землі або на платформі. Важливо для запобігання виконання стрибка в повітрі;

- Image[] hearts – масив, який використовується для відображення сердець на інтерфейсі користувача, що показують здоров'я героя;
- Sprite aliveHeart, deadHeart – спрайти, які відображають серце героя в різних станах: живе і мертве;
- GameObject losePanel – панель, що з’являється, коли герой помирає, і пропонує варіанти перезапустити рівень або повернутися в головне меню;
- Rigidbody2D rb – фізичне тіло героя, яке відповідає за взаємодію з фізичними об'єктами на сцені (наприклад, сила тяжіння);
- Animator anim – аніматор для відображення анімацій героя, наприклад, для бігу, стрибка чи атаки;
- AudioSource jumpSound, miss, attack, damageSound – звукові ефекти для різних дій героя (стрибок, атака, отримання шкоди тощо);
- void RestartLevel() – метод для перезапуску рівня після смерті героя.
- void ToMenu() – метод для повернення на головне меню гри після смерті героя.

Enemy (Ворог):

- void TakeDamage(int damage) – метод для прийому шкоди від героя або іншого об'єкта в грі;
- void GetDamage() – метод для обробки отриманої шкоди, наприклад, зменшення здоров’я;
- void Awake() – метод для ініціалізації ворога в момент його появи на сцені;
- void Update() – метод для оновлення логіки ворога кожного кадру (наприклад, рух чи атака);
- void FixedUpdate() – метод для обробки фізичних взаємодій, наприклад, рух ворога за допомогою Rigidbody2D;
- void Patrol() – метод для патрулювання ворога в межах певної зони;
- void Chase() – метод для переслідування героя, якщо він потрапляє в зону агресії ворога;

- void Attack() – метод для атакування героя або іншого об'єкта;
- void RotateBoss() – метод для обертання ворога (можливо, босса) в напрямку героя або іншого об'єкта;
- Transform leftPoint, rightPoint – точки, між якими ворог пересувається під час патрулювання;
- float health, moveSpeed, attackRange, chaseRange, rotateAngle – змінні для здоров’я, швидкості руху, діапазону атаки, діапазону переслідування та кута повороту ворога;
- Rigidbody2D rb – фізичне тіло ворога, яке відповідає за його рухи та зіткнення з іншими об'єктами;
- BossState currentState – поточний стан ворога (наприклад, патрулювання, переслідування, атака);
- float attackCooldown – час між атаками ворога.

RandVariants (Випадкові варіанти):

- GameObject[] leftspawn, rightspawn – масиви об'єктів, які будуть спаунитися з лівого і правого боку сцени, відповідно;
- void Spawn() – метод для спаунінгу об'єктів (наприклад, ворогів або платформ) на рівні;
- Direction direction – параметр для визначення напрямку, в якому об'єкти будуть спаунитися;
- RandVariants variants – випадкові варіанти спаунінгу, які визначають, який саме об'єкт буде створено.

SpawnBlockscr (Генерація блоків):

- void Start() – метод для ініціалізації об'єктів в момент початку гри або сцени;
- IEnumerator SpawnAfterDelay() – метод, який затримує спаун об'єкта на певний час після початку гри;
- void OnTriggerStay2D(Collider2D other) – метод для обробки зіткнень об'єкта з іншими об'єктами;

bool spawned – змінна для визначення, чи об'єкт вже спаунено на сцені.

float waitTime – час затримки перед спауном;

float destroyDelay – час до того, як об'єкт буде знищений після спауна.

RoomManager (Управління кімнатами):

- static RoomManager Instance – одиничний екземпляр класу RoomManager, який дозволяє іншому коду взаємодіяти з ним;
- int maxRooms – максимальна кількість кімнат, які можуть бути згенеровані;
- int roomCount – поточна кількість створених кімнат;
- int lastLeftIndex, lastRightIndex – індекси останніх згенерованих кімнат, що відповідають за їхню послідовність;
- void Awake() – метод, який ініціалізує клас RoomManager і готує його до роботи при завантаженні сцени.

2.3 Опис архітектури продукту

Архітектура проєкту побудована на принципах об'єктно-орієнтованого програмування з використанням компонувальної моделі Unity, що дозволяє створювати добре організовану та розширювану гру.

Перший етап розробки продукту включав використання Unity, додавання до нього скриптів, асетів та звуків гри. Було створено першу сцену (локацію), яка є головною та містить стартову точку спавну гравця. На цій локації були побудовані платформи, кожна з яких має свій BoxCollider2D, що відповідає за фізичне тіло блоків платформ.

Другий етап полягав у додаванні героя на локацію. Герой має своє фізичне тіло (Rigidbody2D) та код для управління ним. Це один із найважливіших етапів, оскільки всі процеси взаємодії героя з навколишнім середовищем додаються безпосередньо через скрипт героя.

Третій етап – створення головного меню з функціонуючими кнопками. Меню знаходиться як окрема сцена зі своїми позначеннями:

- play – кнопка запуску гри для переходу на основну сцену;
- exit – кнопка для виходу з гри.

Сцена головного меню, окрім кнопок, має також звукове супроводження, яке було взяте з інтернет-джерел без порушення авторських прав. Звуки додають атмосферу та цікавість грі.

Четвертий етап – розробка інших сцен із випадковою генерацією залів. Це було досягнуто завдяки заготовленим варіантам розстановки об'єктів на сцені. При кожному переході на іншу локацію скрипт (ChangeLocation) випадковим чином генерує один із варіантів. Також при смерті гравця локація автоматично змінюється, ускладнюючи проходження. Таким чином, гра набуває більшої цікавості.

П'ятий етап – створення мобів та боссів, які мають штучний інтелект. Їхні задачі включають: патрулювання зони, атакування гравця при потраплянні в зону противника, переслідування героя, різні варіації атак, а також втечу від гравця, якщо їхнє здоров'я падає до низького рівня, щоб уникнути смерті.

Останній етап – створення сайту вікі (LostElementWIKI), який розміщується на комп'ютері і створений за допомогою XAMPP (набір для створення локального сервера сайту). Код сайту написаний мовою PHP. Він має простий інтерфейс та дизайн, з великою кількістю інформації про сюжет гри та допоміжними статтями для проходження гри та управління героєм.

Архітектура сайту:

Сайт для гри *Lost Element* реалізований за допомогою XAMPP – набору програм для створення локального серверу. Вебсайт розроблений на основі PHP, що дозволяє динамічно генерувати контент та забезпечує легке керування інформацією.

Головним завданням сайту є надання користувачам доступу до повної інформації про гру, а також допомога у проходженні через інтерактивні статті, інструкції та підказки. На сайті можна знайти:

- інформацію про сюжет гри: Детальний опис історії, світу гри та персонажів;
- допомогу для гравців: Покрокові інструкції, поради щодо проходження різних етапів гри та управління героєм;

- оновлення та новини: Сторінка, що інформує користувачів про нові випуски, оновлення та зміни в грі;

Вебсайт має простий, але ефективний інтерфейс, що забезпечує зручність користування.

Висновок до розділу 2

У другому розділі було проведено детальний огляд архітектури і поведінки гри Lost Element, що дає можливість ефективно уявити, як користувач взаємодіє з системою, а також організувати основні механіки гри та функціональність сайту. У підрозділі 2.1 було описано використання діаграм варіантів використання (Use Case Diagram) для моделювання того, як гравець взаємодіє з ігровими елементами, а також для визначення функцій сайту, що допомагає користувачам зручно орієнтуватися в системі та отримувати корисну інформацію. Такий підхід дає можливість чітко побачити, які саме дії будуть доступні гравцям і користувачам сайту, що робить взаємодію з продуктом більш інтуїтивною та зручною.

У підрозділі 2.2 було розглянуто моделювання діяльності через діаграми діяльності (Activity Diagram), що дозволяє детально розглянути операції та активності всередині гри. Ці діаграми показують, як зміни дій гравця впливають на розвиток гри, а також як відбувається переміщення між рівнями та перевірка результатів кожного етапу. Такий підхід дозволяє побудувати систему, що може легко адаптуватися до різних варіантів розвитку подій у грі.

У підрозділі 2.3 було розглянуто діаграму класів (Class Diagram), що демонструє внутрішню структуру гри. Вона дає змогу краще зрозуміти, як класи розподіляють функції, що робить систему зрозумілішою і легшою для розширення. Моделювання класів дає змогу побачити, як об'єкти взаємодіють між собою, а також як реалізуються основні механіки гри, такі як поведінка героя, ворогів, генерація рівнів і управління кімнатами. Це дозволяє створити чисту і підтримувану структуру коду.

Також описує архітектуру сайту, який розроблений за допомогою XAMPP та PHP, що забезпечує належну організацію локального сервера для надання користувачам корисної інформації. Веб-сайт відіграє важливу роль у підтримці гри, надаючи доступ до сюжету, інструкцій та оновлень. Це дозволяє гравцям і користувачам сайту отримувати всю необхідну інформацію для полегшення процесу проходження гри.

Загалом, проведене моделювання поведінки гри та сайту дозволяє чітко зрозуміти, як організовані основні механіки та як користувачі взаємодіють з системою. Використання таких методів проєктування забезпечує зручність у розробці, підвищує зручність використання та дає можливість масштабувати проєкт у майбутньому.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ЗАСТОСУНКУ ГРИ LOST ELEMENT

3.1 Особливості реалізації та конструювання програмного продукту

Мова програмування. Для розробки гри було обрано мову програмування C#, яка є основною мовою для роботи з ігровим двигуном Unity. Вибір C# обумовлений кількома важливими аспектами:

Простота вивчення і використання: C# має зрозумілу та чітку синтаксичну структуру, що дозволяє швидко освоїти мову та ефективно застосовувати її для розробки складних систем.

Продуктивність: мова дозволяє створювати високопродуктивні ігри, що важливо для реалізації ігрових механік, що потребують швидкої обробки даних і високої продуктивності графіки.

Тісна інтеграція з Unity: C# є основною мовою програмування в середовищі Unity, що робить процес розробки ігор ефективним і зручним завдяки вбудованим функціям і можливостям для оптимізації ігрового процесу.

Середовище розробки. Для розробки було обрано Unity, оскільки цей ігровий двигун має численні переваги:

– кросплатформеність: Unity дозволяє створювати ігри, які можуть працювати на різних платформах, включаючи ПК, мобільні пристрої та ігрові консолі. Це забезпечує більший охоплення аудиторії;
– інтуїтивний інтерфейс: Unity має зручний та зрозумілий інтерфейс, що дозволяє швидко розпочати розробку та працювати з графікою, анімаціями, фізикою і багатьма іншими аспектами гри;
– підтримка 2D і 3D ігор: Unity забезпечує чудові можливості для створення як 2D, так і 3D ігор, що є важливим аспектом при роботі з різними типами проєктів.

Платформа. Розробка гри здійснювалася для Windows з планами подальшого перенесення на мобільні платформи (iOS, Android). Unity забезпечує можливість легко адаптувати проєкт для різних платформ, що

робить його гнучким для розробників і дозволяє працювати з мінімальними зусиллями.

Бібліотеки та інструменти [19]:

- *Unity Engine:* основний інструмент для розробки, який дозволяє працювати з графікою, фізикою, анімаціями і звуками в реальному часі. Unity підтримує роботу з різними типами контенту, що дає змогу реалізувати всі аспекти гри без необхідності інтеграції зовнішніх рішень;
- *Cinemachine:* Бібліотека для управління камерою, яка дозволяє створювати динамічні та складні анімації камери, що додає гнучкості в розробку ефектних сцен і забезпечує захоплюючий ігровий досвід;
- *TextMesh Pro:* Потрібна бібліотека для роботи з текстами в Unity. Вона забезпечує високоякісні шрифти, ефекти і стильове оформлення текстів у грі, що покращує візуальну складову гри;
- *Tilemap:* Інструмент для створення 2D карт, що дозволяє швидко створювати рівні гри, що складаються з плиток (tiles), тим самим прискорюючи процес розробки та мінімізуючи помилки при ручному введенні;
- *A Pathfinding Project*:* Ця бібліотека дозволяє реалізувати ефективні алгоритми пошуку шляху, які є необхідними для штучного інтелекту (ШІ) у грі, зокрема для руху персонажів або ворогів по рівнях;
- *Physics2D:* Вбудована бібліотека для роботи з фізикою в 2D-просторі. Вона дозволяє реалізувати реалістичну взаємодію об'єктів в грі, зокрема для руху, падіння, зіткнень і інших фізичних ефектів;
- *[SerializeField]:* Атрибут, який дозволяє змінним з *private* доступом бути видимими та редагованими в інспекторі Unity. Це забезпечує зручність налаштування параметрів без порушення принципів інкапсуляції в програмуванні. Використання *[SerializeField]* дозволяє зберегти приватність змінних і одночасно надавати можливість редагувати їх значення безпосередньо в редакторі Unity, що є особливо корисним для налаштування таких параметрів, як швидкість персонажа, рівень здоров'я чи інші характеристики, що впливають на ігровий процес.

Програми та інструменти для графіки та звуку:

- *Aseprite:* Спеціалізована програма для створення 2D піксельної графіки, яка була використана для розробки піксельних спрайтів і анімацій для персонажів та об'єктів гри. Aseprite дозволяє створювати піксельні анімації в реальному часі, що є корисним для 2D ігор, особливо при роботі з великими обсягами графіки;
- *Adobe Photoshop:* Використовувалася для створення та обробки 2D спрайтів і текстур. Цей інструмент дозволяє створювати детальну графіку, яку можна використовувати у грі для створення анімацій і елементів інтерфейсу;
- *Tiled Map Editor:* Інструмент для створення рівнів за допомогою плиток, який дозволяє швидко створювати складні карти, що складаються з маленьких плиток, що значно полегшує процес картографії в 2D іграх;
- *Audacity:* Програма для запису та обробки аудіо, що використовувалася для створення і редагування звукових ефектів і музики для гри, що робить її атмосфернішою та емоційно насиченою.

Приклад коду для переміщення персонажа в Unity:

csharp

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

public class Hero : Entity
{
    [SerializeField] private float speed = 3f;
    [SerializeField] private float health;
    [SerializeField] private float jumpForce = 10f;
    private bool isGrounded = false;

    public bool isAttacking = false;
    public bool isRecharged = true;

    public Transform attackPos;
    public float attackRange;
    public LayerMask enemy;
    public VectorValue pos;



    [SerializeField] private Image[] hearts;

    [SerializeField] private Sprite aliveHeart;
```

```
[SerializeField] private Sprite deadHeart;
[SerializeField] private GameObject losePanel;



private Rigidbody2D rb;
private Animator anim;
private SpriteRenderer Square;

[SerializeField] private AudioSource jumpSound;
[SerializeField] private AudioSource miss;
[SerializeField] private AudioSource attack;
[SerializeField] private AudioSource damageSound;
public static Hero Instance { get; set; }

private States State
{
    get { return (States)anim.GetInteger("state"); }
    set { anim.SetInteger("state", (int)value); }
}
void Awake()
{
    lives = 5;
    health = lives;
    Instance = this;
    transform.position = pos.intialValue;
    rb = GetComponent<Rigidbody2D>();
    anim = GetComponent<Animator>();
    Square = GetComponentInChildren<SpriteRenderer>();
    isRecharged = true;
}

private void FixedUpdate()
{
    ChekGround();
}

void Update()
{
    if (isGrounded && health > 0) State = States.idle;

    if (Input.GetButton("Horizontal"))
        Run();

    if (isGrounded && health > 0 && Input.GetButtonDown("Jump"))
        Jump();
    if (Input.GetKeyDown(KeyCode.Q))
        Attack();


    if (transform.position.y < -6f)
        GetDamage();

    if (health > lives)
        health = lives;

    for (int i = 0; i < hearts.Length; i++)
    {
        if (i < health)
            hearts[i].sprite = aliveHeart;
        else
            hearts[i].sprite = deadHeart;
```

```
            if (i < lives)
                hearts[i].enabled = true;
            else
                hearts[i].enabled = false;
        }


    }
    void Run()
    {

        if (isGrounded) State = States.run;

        Vector3 dir = transform.right * Input.GetAxis("Horizontal");
        transform.position = Vector3.MoveTowards(transform.position,
transform.position + dir, speed * Time.deltaTime);
        Square.flipX = dir.x < 0.0f;


    }

    void Jump()
    {
        rb.AddForce(transform.up * jumpForce, ForceMode2D.Impulse);
        jumpSound.Play();

    }
    private void OnDrawGizmosSelected()
    {
        Gizmos.color = Color.red;
        Gizmos.DrawWireSphere(attackPos.position, attackRange);
    }
    private IEnumerator AttackAnimation()
    {
        yield return new WaitForSeconds(1f);
        isAttacking = false;
    }
    private IEnumerator AttackCoolDown()
    {
        yield return new WaitForSeconds(0.5f);
        isRecharged = true;
    }
    void ChekGround()
    {
        Collider2D[] collider =
Physics2D.OverlapCircleAll(transform.position, 0.3f);
        isGrounded = collider.Length > 2;

        if (!isGrounded && health > 0) State = States.jump;
    }
    private void Attack()
    {
        if (isGrounded && isRecharged)
        {
            State = States.attack;
            isAttacking = true;
            isRecharged = false;

            StartCoroutine(AttackAnimation());
            StartCoroutine(AttackCoolDown());
        }
    }
```

```
    private void OnAttack()
    {
        Collider2D[] colliders =
Physics2D.OverlapCircleAll(attackPos.position, attackRange, enemy);

        if (colliders.Length == 0)
            miss.Play();
        else
            attack.Play();
        for (int i = 0; i < colliders.Length; i++)
        {
            colliders[i].GetComponent<Entity>().GetDamage();

        }
    }
    public override void GetDamage()
    {
        health -= 1;
        damageSound.Play();
        if (health == 0)
        {
            foreach (var h in hearts)
                h.sprite = deadHeart;
            Die();

        }
    }

    public override void Die()
    {
        losePanel.SetActive(true);
        Time.timeScale = 0;
    }
}
public enum States
{
    idle,
    run,
    jump,
    attack
      }
```

Цей фрагмент відповідає за переміщення персонажа по рівню. Важливою частиною є використання Rigidbody2D для коректної фізики руху, що дозволяє реалізувати плавний рух у 2D-просторі.

Обґрунтування вибору засобів реалізації. Вибір C# та Unity був зроблений завдяки їх високій ефективності, зручності використання та широкій підтримці в індустрії відеоігор. Unity дозволяє створювати кросплатформені ігри, а також дає доступ до численних інструментів для роботи з графікою, фізикою, анімацією і звуком, що значно пришвидшує процес розробки. C# є мовою, яка безпосередньо підтримує інтеграцію з Unity і дозволяє зручно працювати з усіма компонентами гри. Такий вибір

інструментів є оптимальним для створення багатофункціональних та конкурентоспроможних проєктів у галузі відеоігор.

3.2. Конструювання

Перша частина, у якій йтиметься про розробку рівнів та переходів. Кожен рівень був зібраний із об’єктів, розташованих на сцені. Якщо говорити про задній фон, який бачить гравець, його потрібно правильно розташувати відповідно до розмірів камери. Далі йдуть налаштування в інспекторі, а також не слід забувати про код. Фон дублюється по горизонталі та розташовується відповідно до довжини рівня.

Для повноти рівня необхідно розташувати платформи та блоки, по яких пересуватиметься гравець. Їх також підлаштовують під рівень – в інспекторі додається компонент BoxCollider2D, таким чином додається фізика до об’єктів. Коли вдалося налаштувати платформи та задній фон, наступним кроком є наповнення рівня, аби він не виглядав порожнім.

Далі слід налаштувати переходи між сценами. Щоб це зробити, на сцені в Unity створюється порожній об’єкт. Його можна позначити кольорами чи іншим способом. В інспекторі йому присвоюється BoxCollider2D, щоб гравець міг з ним зіткнутися. Також потрібно написати скрипт, який відповідатиме за завантаження наступної сцени. Цей скрипт додається до створеного об’єкта, і в ньому вказується назва наступної сцени.

Надалі найголовніше в моїй частині це генерації локації та рівна вона зроблена завдяки заготовленим локаціям RandbBlocks (рис 3.1), які через скрипт з’являються в випадковому порядку, що дозволяє облегшити сам процес налаштування, в такому концепті варіантів та багів з самою локацією буде мінімізовано, і вся увага буде зроблена невелчких змінах складності для проходження.

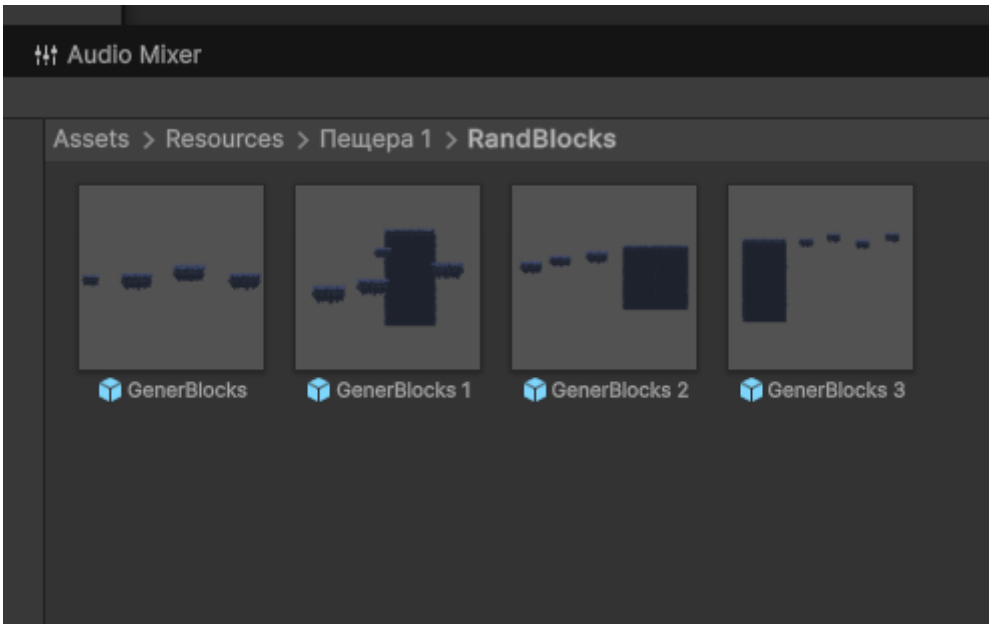

Рисунок 3.1 – RandBlocks

Джерело: розроблено автором

Не забуваємо також реалізувати користувацький інтерфейс, який відображатиме здоров'я головного героя, кількість монет та іншу інформацію. Щодо створення інтерфейсу здоров'я та лічильника монет: у самому рушії створюється Canvas, на якому розміщуються підготовлені зображення сердець. Необхідно додати скрипти, які оновлюватимуть стан сердець при зіткненні з ворогом чи перешкодою. У цьому скрипті додаються два зображення – перше відповідає за повне серце, друге – за порожнє. Потім цей скрипт додається до об'єкта гравця на сцені.

Наступне, що потрібно додати ворогів, які становитимуть загрозу для гравця. У грі буде представлено дві поведінки ворогів.

Перший ворог патрулюватиме між двома заданими точками (рис 3.2). Якщо він помітить гравця в межах невеликого радіуса, то почне його переслідувати, поки не втратить із поля зору. Весь алгоритм задається ворогу через скрипт. У цьому скрипті задаються параметри нанесення шкоди, здоров'я та швидкість руху.

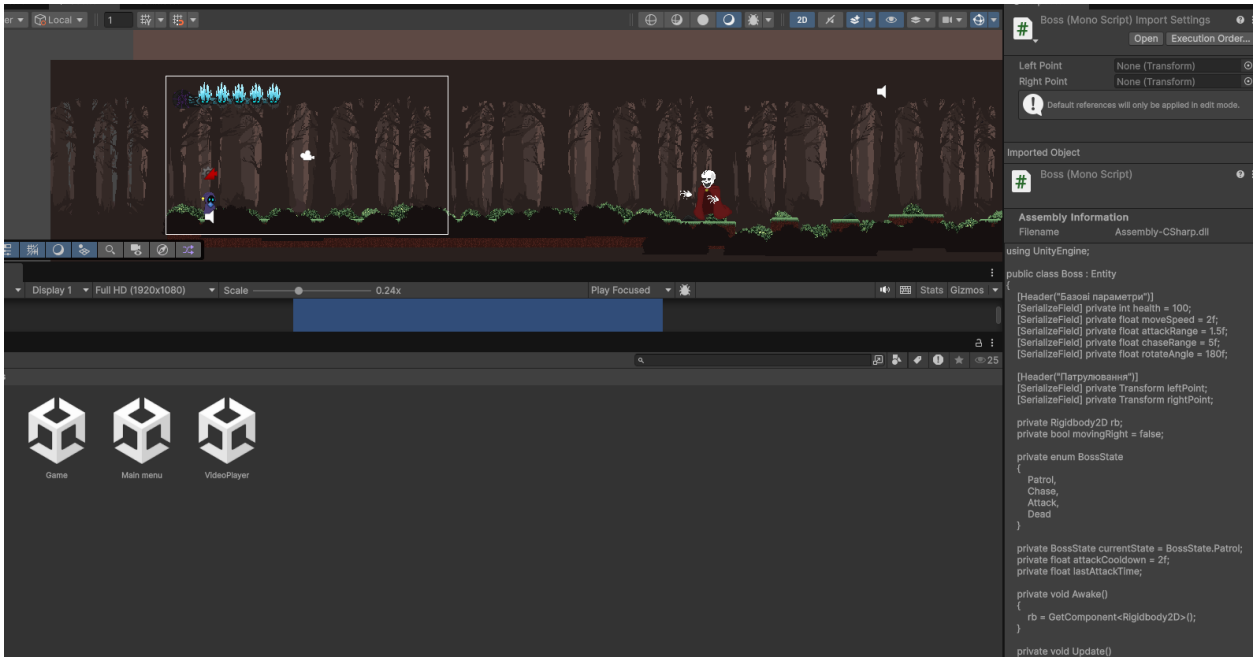

Рисунок 3.2 – First Boss

Джерело: розроблено автором

Другий ворог матиме дещо схожий алгоритм, але працюватиме трохи інакше. Цей ворог перебуватиме у повітрі, тобто він рухатиметься по повітряному простору. Як і перший, він матиме власну зону патрулювання, буде фізичним для гравця та зможе його переслідувати. Для нього створюється окремий скрипт, який відповідатиме за патрулювання, переслідування, та параметри здоров’я, швидкості руху, шкоди.

Також не слід забувати про підключення анімацій. Спочатку створюємо анімації ворога через Animator. У вкладці Animations додаємо анімації для ворога, створюючи окрему папку для збереження (рис.3.3). Кожній анімації надається відповідна назва, після чого все налаштовується на таймлайні для коректного відображення. Коли всі анімації створено, вони редагуються в Animator, де виставляються правила та умови, за яких анімації працюватимуть правильно.

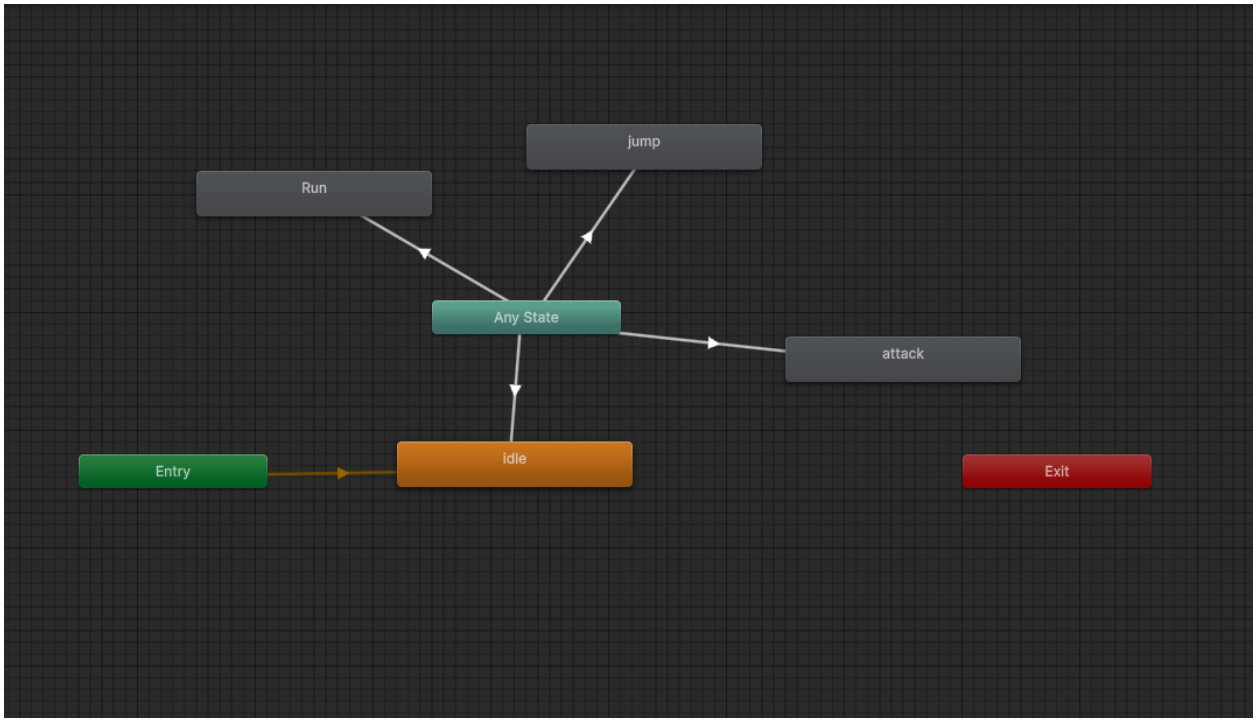

Рисунок 3.3 – Animations

Джерело: розроблено автором

Наприкінці рівня, на якому перебуватиме герой, потрібно логічно завершити геймплей. Для цього на сцені слід розмістити об’єкт або точку, до якої має дійти герой, щоб завершити рівень. Після досягнення цієї точки гравець зможе перейти до головного меню або вийти з гри.

Друга частина. У межах проєкту було розроблено сайт-вікі, який слугує внутрішньою документаційною платформою для команди розробників гри «*Lost Element*». Цей сайт реалізовано на базі локального веб-сервера XAMPP, що забезпечує швидкий і зручний доступ до інформації всередині локальної мережі без потреби в зовнішньому хостингу.

Використані інструменти:

- *xampp* – це набір безкоштовного програмного забезпечення, що включає Apache, MySQL (MariaDB), PHP та Perl. Він був використаний для розгортання локального веб-сервера;
- *mediawiki* – рушій, який було встановлено на XAMPP-сервері для реалізації функціоналу вікі-сайту. Це та ж платформа, яка

використовується для Wikipedia, що забезпечує зручний синтаксис та інтерфейс для редагування сторінок.

Структура сайту. Сайт-вікі включає наступні розділи:

- *опис гри* – загальна інформація про сюжет, жанр, цільову аудиторію;
- *ігрова механіка* – детальний опис основних і другорядних механік гри, таких як рух персонажа, бойова система, генерація рівнів тощо;
- *технічна документація* – архітектура проєкту, UML-діаграми (діаграми класів, діяльності, варіантів використання), опис API та використовуваних класів;
- *історія змін (changelog)* – хронологічний список оновлень і змін, внесених під час розробки.

Переваги такого підходу:

- *централізований доступ* до актуальної інформації всіма членами команди;
- *простота редагування* – завдяки MediaWiki не потрібні знання HTML чи PHP для внесення змін у вміст;
- *можливість резервного копіювання* – весь сайт може бути легко збережений або перенесений на інший сервер.

Висновки до розділу 3

У процесі розробки гри в Unity вдалося реалізувати базову структуру рівнів, що складаються з платформ, фону та об'єктів взаємодії. Кожен рівень ретельно налаштовувався в інспекторі, включно з фізикою об'єктів та системою переходів між сценами.

Особливу увагу приділено користувацькому інтерфейсу панелі здоров’я, лічильнику монет та іншим елементам.

Головний герой гри керується за допомогою скриптів і володіє ближньою атакою, що дозволяє йому ефективно протистояти ворогам.

У грі реалізовано дві поведінкові моделі ворогів наземного та повітряного, кожен із власною логікою патрулювання та переслідування. Для обох ворогів налаштовані анімації, здоров’я, шкода та швидкість руху.

Завершення рівня відбувається через спеціальну зону, яка дозволяє гравцю завершити гру або перейти до головного меню.

Загалом цей розділ охоплює ключові етапи побудови геймплею та логіки, що формують повноцінний ігровий досвід.

ВИСНОВКИ

У процесі виконання даної роботи було повністю реалізовано концепцію, проєктування та створення гри в жанрі 2D платформера з використанням рушія Unity. Робота складалася з трьох розділів, кожен із яких зробив свій вагомий внесок у досягнення поставленої мети створення ігрового застосунку з базовою механікою, геймплейними елементами та структурованим проєктуванням.

У першому розділі розглядалася предметна область ігри як тип програмного забезпечення. Було проаналізовано сучасний стан індустрії, визначено основні переваги та недоліки різних підходів до розробки, а також проведено огляд аналогів. Це дозволило краще зрозуміти вимоги до сучасних ігор, особливості геймплейних рішень та способи залучення користувачів. Окрему увагу приділено ігровим рушіям, зокрема Unity, як одному з найпопулярніших інструментів розробки 2D та 3D проєктів.

Другий розділ був присвячений проєктуванню гри та технічного середовища. У ньому було розроблено чотири основні діаграми: діаграму варіантів використання Use Case, яка описує взаємодію користувача з грою; дві діаграми активності Activity Diagram одна для гри, інша для сайту, які демонструють логіку роботи користувача з інтерфейсами; а також діаграма класів Class Diagram, яка структурує архітектуру застосунку на рівні об'єктно-орієнтованого програмування. Це забезпечило чітке бачення майбутньої структури гри, її основних компонентів і взаємодії між ними.

У третьому розділі було реалізовано гру в середовищі Unity. Описано основні особливості реалізації: створення ігрових рівнів, розміщення об'єктів, фону, інтерактивних елементів та платформ. Головний герой керується за допомогою скриптів, має ближню атаку й взаємодіє з ігровим світом через фізику об’єктів. Також було створено кілька типів ворогів із різною логікою поведінки: наземні з патрулюванням та переслідуванням і повітряні з аналогічним алгоритмом, адаптованим до руху в повітрі. Додатково

реалізовано користувацький інтерфейс, який відображає здоров’я гравця, кількість зібраних монет. Рівень завершується спеціальною зоною, яка переводить гравця до головного меню або дозволяє завершити гру.

У підсумку, завдяки системному підходу, вдалося поетапно пройти всі основні етапи створення 2D гри – від аналізу та проєктування до технічної частини. Отримані знання та практичні навички є фундаментальними для подальшої роботи в галузі ігрової розробки. Наша гра демонструє базові механіки, які можуть бути розширені в майбутньому для створення повноцінного ігрового продукту.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Platformer – URL: https://en.wikipedia.org/wiki/Platformer

2. Shri Shivaji. EXPLORING THE POPULARITY OF 2D GAMES // International Research Journal of Modernization in Engineering Technology and Science. 07.04.April-2025. № - 8.187. - С. 1–3.

3. C# language documentation - URL: https://learn.microsoft.com/en-us/dotnet/csharp/

4. Unity Базовый курс: 2D платформер с нуля - URL: https://unity3dschool.com/unity-bazovyj-kurs-2d.html

5. Створення ігрової сцени і скрипу на Unity - URL: https://go-mother.com/creating-game-scene-and-squeak-on-unity/

6. Платформер - URL: https://uk.wikipedia.org/wiki/%D0%9F%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%B5%D1%80?utm_source=chatgpt.com

7. Introduction to the A* Algorithm - URL: https://www.redblobgames.com/pathfinding/a-star/introduction.html

8. Створення штучного інтелекту ворогів у іграх. Поради Senior Gameplay Designer компанії Remedy - URL: https://gamedev.dou.ua/blogs/enemy-ai-in-games/

9. Створення 2D-анімації в Unity - URL: https://uk.sharpcoderblog.com/blog/creating-2d-animations-in-unity

10. Unity 2D suite for game creators - URL: https://unity.com/solutions/2d

11. 2D game creation workflow - URL: https://docs.unity3d.com/6000.1/Documentation/Manual/2d-game-creation-wokflow.html

12. Unity Add Enemies to a 2D Platformer - URL: https://www.sharpcoderblog.com/blog/unity-3d-enemy-ai-tutorial-for-2d-platformer

13. Adams, E. (2013). Fundamentals of Game Design. Берклі: New Riders Publishing, 2013. С. 236.

14.Діаграма прецедентів – URL: https://uk.wikipedia.org/wiki/Діаграма_прецедентів?utm_source=chatgpt.com

15. Choosing the resolution of your 2D art assets – URL: https://unity.com/blog/engine-platform/choosing-the-resolution-of-your-2d-art-assets

16. Діаграма діяльності – URL: https://uk.wikipedia.org/wiki/Діаграма_діяльності

17. Діаграма класів – URL: https://uk.wikipedia.org/wiki/Діаграма_класів

18. Steam - URL: https://en.wikipedia.org/wiki/Steam

19. Platform development – URL: https://docs.unity3d.com/Manual/PlatformSpecific.html

20. 1. Gris (игра) – URL: https://en.wikipedia.org/wiki/Gris_(игра)

21. Little Nightmares – URL: https://littlenightmares.fandom.com/en/wiki/Little_Nightmares

22. Journey (игра, 2012) – URL: https://en.wikipedia.org/wiki/Journey_(игра,_2012)

23. Dead Cells – URL: https://en.wikipedia.org/wiki/Dead_Cells

24. Мічківський С. Microsoft Office (Word, Excel, Outlook …) : навч. посіб. / С. Мічківський, Д. Балдик, В. Головань; Східноукр. нац. ун-т ім. В. Даля, Аграр. ф-т. – Київ : [Вид-во Східноукр. нац. ун-т ім. В. Даля], 2023. – 128 с. – URL: https://dspace.snu.edu.ua/handle/123456789/1723

25. Vysochyn I., Michkivskyy S. Using machine learning for translation and speech generation in e-book reading applications. Держава, регіони, підприємництво: інформаційні, суспільно-правові, соціально-економічні аспекти розвитку: матеріали VI Міжнародної наукової конференції (5-6 грудня 2024 р., м. Київ). Київ: Університет "КРОК", 2024. С.62-64 – URL: https://dspace.krok.edu.ua/items/6e1488e1-9e21-4282-af10-2e3bca48d2bc