Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук (або центр післядипломної освіти, або навчально-науковий центр заочної форми навчання)
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Ігровий програмний застосунок у жанрі Metroidvania з елементами карткової гри. Ігрові механіки. Кодування. Діалогова система.
(тема)

Виконав:
студент 4 курсу, групи ПЗПІ-20-1

Лабунець В.В.
(прізвище, ініціали)

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

Керівник ст.викл. кафедри ПІ Саманцов О.О.
(посада, прізвище, ініціали)

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

2024 р.

Харківський національний університет радіоелектроніки

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

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

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Лабунцю Владиславу Валерійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Ігровий програмний застосунок у жанрі Metroidvania з елементами карткової гри. Ігрові механіки. Кодування. Діалогова система.
Затверджена наказом по університету від 20.05. 2024р. № 471 Ст
2. Термін подання студентом роботи до екзаменаційної комісії 08.06.2024
3. Вихідні дані до роботи Розробити ігровий застосунок в жанрі Metroidvania з елементами карткової гри, його ігрові механіки і діалогову системи, мовою програмування JavaScript використовуючи середу розробки Ct.js.
4. Перелік питань, що потрібно опрацювати в роботі
Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проектування програмного забезпечення, опис прийнятих програмних рішень, тестування розробленого програмного забезпечення, впровадження програмного забезпечення, висновки, додатки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	11.03.2024	*виконано*
2	Створення специфікації ПЗ	10.04.2024	*виконано*
3	Проектування ПЗ	15.04.2024	*виконано*
4	Розробка ПЗ	20.05.2024	*виконано*
5	Тестування ПЗ	25.05.2024	*виконано*
6	Оформлення пояснювальної записки	01.06.2024	*виконано*
7	Підготовка презентації та доповіді	03.06.2024	*виконано*
8	Попередній захист	05.06.2024	*виконано*
9	Нормоконтроль, рецензування	07.06.2024	*виконано*
10	Здача роботи у електронний архів	07.06.2024	*виконано*
11	Допуск до захисту у зав. кафедри	09.06.2024	*виконано*

Дата видачі завдання 8 квітня 2024р.

Студент (ка)_____ Лабунець В.В,
(підпис)

Керівник роботи _____ ст.викл. кафедри ПІ Саманцов О.О.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавра, 60 стор., 29 рис., 1 табл., 10 джерел.

ГРА МЕТРОЇДВАНІЯ, ІГРОВИЙ ПРОГРАМНИЙ ЗАСТОСУНОК, ІГРОВА МЕХАНІКА, ДІАЛОГОВА СИСТЕМА, JAVASCRIPT, CT.JS, HTML5

Об’єкт розробки – ігровий програмний застосунок в жанрі метроїдванія.

Мета розробки – проєктування та розробка ігрових механік та діалогової системи ігрового програмного застосунку в жанрі метроїдванія.

Метод рішення – середовище розробки Ct.js, мова програмування JavaScript.

У результаті розробки створено ігровий програмний застосунок з системою діалогів та механіками гри відповідними жанру метроїдванія.

METROIDVANIA GAME, GAME SOFTWARE, GAME MECHANICS, DIALOGUE SYSTEM, JAVASCRIPT, CT.JS, HTML5

The object of development is a game software application in the metroidvania genre.

The purpose of the development is to design and develop game mechanics and a dialogue system of a game software application in the metroidvania genre.

The solution method is the Ct.js development environment, the JavaScript programming language.

As a result of the development, a game software application with a dialogue system and game mechanics corresponding to the metroidvania genre was created.

Я, Лабунець Владислав Валерійович, студент гр. ПЗПІ-20-1, здобувач вищої освіти на першому (бакалаврському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Ігровий програмний застосунок у

жанрі Metroidvania з елементами карткової гри. Ігрові механіки. Кодування. Діалогова система», що буде представлена до екзаменаційної комісії для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Усі запозичення з друкованих та електронних джерел мають відповідні посилання.

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

ЗМІСТ

Вступ 8
1 Аналіз предметної галузі 9
1.1 Аналіз предметної галузі 9
1.1.1 Аналіз ринку 9
1.1.2 Аналіз концепції жанру 9
1.1.3 Аналіз ігрових механік 12
1.1.4 Аналіз аналогів 15
1.2 Виявлення та вирішення проблем 17
1.3 Постановка задачі 17
1.3.1 Цільова аудиторія 18
2 Формування вимог до програмної системи 19
2.1 Формування концепції гри 19
2.2 Формування функціональних вимог 20
2.3 Формування системних вимог 21
3 Архітектура та проєктування програмного забезпечення 23
3.1 UML проєктування ПЗ 23
3.2 Проєктування архітектури ПЗ 26
3.3 Проєктування структури зберігання даних 27
3.4 Приклади найцікавіших алгоритмів та методів 29
3.5 Створення дизайну системи 31
4 Опис прийнятих програмних рішень 36
4.1 Основні елементи гри 36
4.2 Створення системи переміщення 39
4.3 Створення бойової системи 41
4.4 Створення логіки ворожих істот 43
4.5 Створення діалогової системи 47
5 Тестування програмного забезпечення 50
5.1 Проєктування тестування 50
5.2 Проведення тестування 50

5.2.1 Тестування інсталяції 50
5.2.2 Функціональне тестування 51
5.2.3 Тестування інтерфейсу користувача 52
5.2.4 Тестування відмовостійкості та відновлення 52
5.2.5 Тестування продуктивності 53
5.2.6 Тестування конфігурації 53
5.3 Результати тестування 54
6 Впровадження програмного забезпечення 55
6.1 Соціальні мережи 55
6.2 Вивантаження на платформу 57
Висновки 59
Перелік джерел посилання 60
Додаток А 61
Додаток Б 62
Додаток В 68
Додаток Г 87

ВСТУП

Темою кваліфікаційної роботи є ігрові механіки та діалогова система ігрового програмного застосунку в жанрі Metroidvania з елементами карткової гри. Ця тема є дуже актуальною в контексті сучасної індустрії відеоігор. Ігрова індустрія стає дедалі більше з кожним роком, залучаючи все більше нових користувачів. Як результат, індустрії необхідний безперервний потік нових ігор, для задоволення попиту, що постійно зростає. А поява нових технологій лише розширюють можливості розробників та спрощують створення та реалізацію нових ігрових механік.

Жанр 2D ігор Metroidvania є одним з найпопулярніших у світі, і тому, розробка нових ігор у цьому напрямку завжди зустрічається з великим інтересом. Основна функція ігрових механік у жанрі Metroidvania – надання користувачеві можливості досліджувати ігровий світ, що представляє собою 2D платформер з великими зв’язаними локаціями, дружніми і ворожими істотами та історією переплетеною з геймплеєм. Діалогова система, у свою чергу, має функцію допомогти користувачу взаємодіяти із істотами ігрового світу та краще розуміти його історію.

Метою роботи є проєктування та розробка ігрових механік та системи діалогів для ігрової системи, яка складається з застосунку під платформи Windows, macOS, Linux, Android та iOS. Для розробки ігрового застосунку використовувався ігровий редактор Ct.js, заснований на технологіях HTML5, CSS, JavaScript та фреймворку PIXI.js для роботи з графікою 2D-ігор.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

1.1.1 Аналіз ринку

З моменту своєї появи ігрова індустрія з кожним роком зростає, і ця тенденція тільки посилюється. Одним з найпопулярніших жанрів 2D ігор я є Metroidvania. Статистика показує, що саме він є одним з найбільш комерційно успішних. Шанувальники відеоігор активно шукають стильні та глибокі ігри, і жанр Metroidvania зазвичай задовольняє ці вимоги.

1.1.2 Аналіз концепції жанру

Взагалі досить складно сказати, коли виник жанр відеоігор Metroidvania. Вважається, що датою виникнення цього жанру є середина 80-х років[1], коли була видана класична гра «Metroid» від Nintendo (див. рис. 1.1).

Рисунок 1.1 – Інтерфейс гри «Metroid» 1986 року.

Хоч «Metroid» і не була першою у своєму роді, проте саме вона називається тією грою, яка найбільше вплинула на жанр. Не останню роль у визначенні жанру Metroidvania зіграло те, що ця гра набула немалої популярності.

Nintendo намагалися створити пригодницьку гру та при цьому зробити її нелінійною, як це було зазвичай, для того щоб виділити її серед решти ігор тієї епохи. Гравець міг вільно повертатися в раніше відвідані локації для будь-яких цілей, наприклад, для отримання бонусів. Окрім нелінійності важливим нововведенням було те, що різні бонуси, які гравець отримував чи знаходив протягом гри, залишалися з ним, а не діяли тимчасово, як це було прийнято в класичних рівневих платформерах того часу.

Ще однією серією ігор грою, яка визначила жанр можна вважати серію платформерів «Castlevania» в готичному антуражі, яка саме у цей час набирала популярності (див. рис. 1.2).

Рисунок 1.2 – Інтерфейс гри «Castlevania» 1986 року.

Перша частина серії «Castlevania» 1986 року, складалася з окремих рівнів, які мали проходитись послідовно і більше напам’ятала класичний платформер. Але вже

в наступних «Vampire Killer» 1986 року та «Castlevania II: Quest Simon's» 1987 року, були деякі намагання зробити геймплей більш нелінійним та пригодницьким.

Під час розробки ще одної ігри серії «Castlevania: Symphony of the Night», виданої 1997 року на PlayStation, для того щоб залучити нових гравців, які не мають великого досвіду в іграх, та збільшити тривалість геймплею, було прийнято рішення декілька змінити концепцію гри (див. рис. 1.3). У неї було впроваджено ідеї, використані до цього у «Super Metroid» 1994 року, третій частині серії ігор «Metroid», такі як наявність відкритого світу, що складається з різних локацій, для проходження та дослідження яких вимагалося отримати різні ключові предмети, а також можливість покращення характеристик героя. Змінений геймплей дуже сподобався гравцям, тому наступні ігри серії продовжували його копіювати та доповнювати.

Рисунок 1.3 – Інтерфейс гри «Castlevania: Symphony of the Night» 1997 року.

Потрібно розуміти, що ігрова індустрія в ті роки була зовсім новою і тільки починала розвиватися, тому тоді ще не було прийнято ділити ігри за жанрами, кожний з яких мав певні відмінності. Metroidvania як ігровий жанр сформувався вже набагато пізніше, коли розробники намагаючись зробити популярну гру

повторювали певні механіки та особливості відомих проектів. Але саме з появою «Super Metroid» та «Castlevania: Symphony of the Night» була визначена формула, яка і полягла у основу сучасного жанру Metroidvania. Навіть слово Metroidvania – це поєднання назви «Metroid» і останніх літер слова «Castlevania».

1.1.3 Аналіз ігрових механік

Перші представники Metroidvania як оформленого жанру з'явилися ще в середині 2000-х. А по-справжньому популярним цей жанр зумів стати лише у 2010-х, коли було випущено чимало відомих представників жанрів[2]. Як і будь-який інший жанр Metroidvania має свої відмінні характеристики. Основні риси жанру включають у себе:

- нелінійна структура світу (див. рис. 1.4);
- нелінійний зріст складності;
- прогресія заснована на предметах;
- наявність здібностей та апгрейдів;
- секрети та таємні шляхи;
- особлива бойова система;
- атмосфера і стиль.

Один з найголовніших елементів будь-якої гри жанру Metroidvania – це, звичайно, наявність великого світу, що складається з багатої кількості окремих локацій та рівнів, деякі з яких можуть бути тимчасово закриті, а інші навіть можуть бути секретними[3]. І користувач має можливість вільно переміщуватись по них залежно від його власних вподобань та здібностей персонажа. Іншими словами користувач має можливість проходити гру у зручній для нього послідовності.

Також важливою особливістю світу у грі Metroidvania є те, що складність проходження завжди зростає нелінійно. Тобто навіть на початку гри користувач може потрапити а рівень, який не зможе пройти через надмірну його складність. І тоді йому доведеться змінити шлях проходження гри та повернутись у початкову локацію вже після підняття рівня чи отримання здібностей, щоб закінчити дослідження.

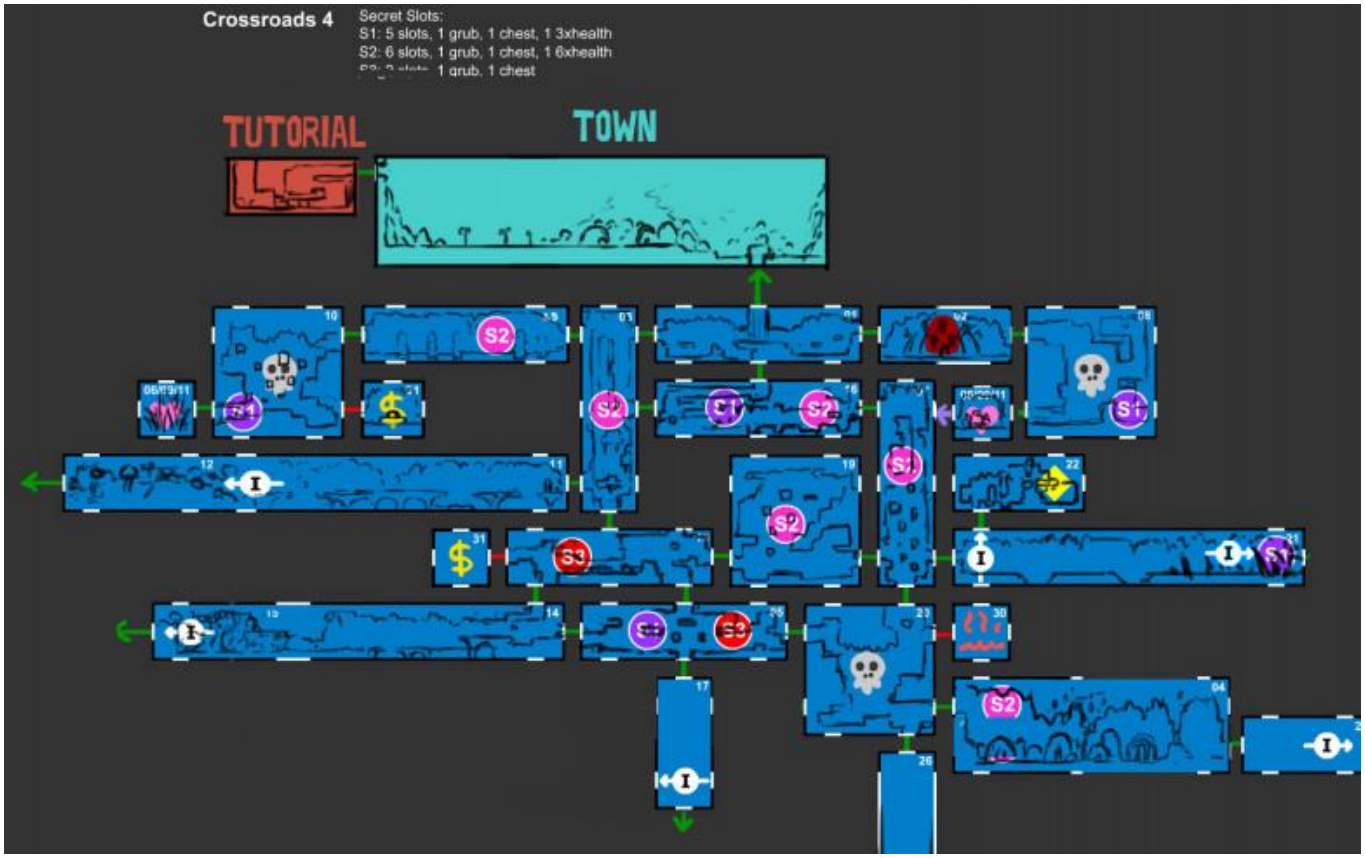


Рисунок 1.4 – Приклад нелінійного світу.

Важливу роль у іграх Metroidvania відіграють зібрані предмети. Для переходу до інших рівнів та просування в грі гравець зазвичай повинен знаходити та збирати різноманітні предмети, такі як ключі, картки, аптечки або нові здібності. Ці предмети дозволяють гравцю не тільки відкривати нові області, а й полегшувати боротьбу з ворогами та розблоковувати секрети.

Користувач часто прогресує через здобуття нових здібностей або навичок. Наприклад, гравець може отримати здібність стрибати вище, літати, що відкриває нові можливості для дослідження світу.

Як вже було сказано, ще одною особливістю Metroidvania прийнято вважати наявність різноманітних секретів та можливостей відкривати додаткові шляхи для проходження гри. У таких іграх часто можна зруйнувати чи відсунути якийсь предмет та стену у певному куточку, щоб відкрити прохід до в іншої локації, знайти захований розробниками секрет чи іншу нагороду. Не всім подобається необхідність

занадто ретельно досліджувати кожну кімнату та повертатись до попередніх рівнів, але це є однією з ключових особливостей жанру.

Також ігри жанру Metroidvania часто завжди містять досить багато бойових елементів. Гравець майже постійно зустрічається з різноманітними ворогами та босами, що потребують не тільки реакції та досвіду, але й стратегічного підходу для перемоги. Багато з ворогів мають свої унікальні атаки, сильні та слабкі місця. Гравець має їх знати та використовувати для своєї вигоди.

Ще буде важливо згадати про особливу атмосферу і стиль, що є ключовими аспектами жанру. Саме вони відіграють важливу роль у створенні неповторного та захоплюючого геймплею. Найчастіше у таких іграх використовують науково-фантастичний чи фентезійний сеттинг, а художній стиль гри вражає візуальною привабливістю і деталізацією своїх образів та визначається стильним інтерфейсом, фонами, персонажами та ворогами (див. рис. 2.5). Музика підкреслює тематику гри та може бути напруженою і захоплюючими під час бойових сцен, або ж спокійною і мелодійною під час дослідження світу.

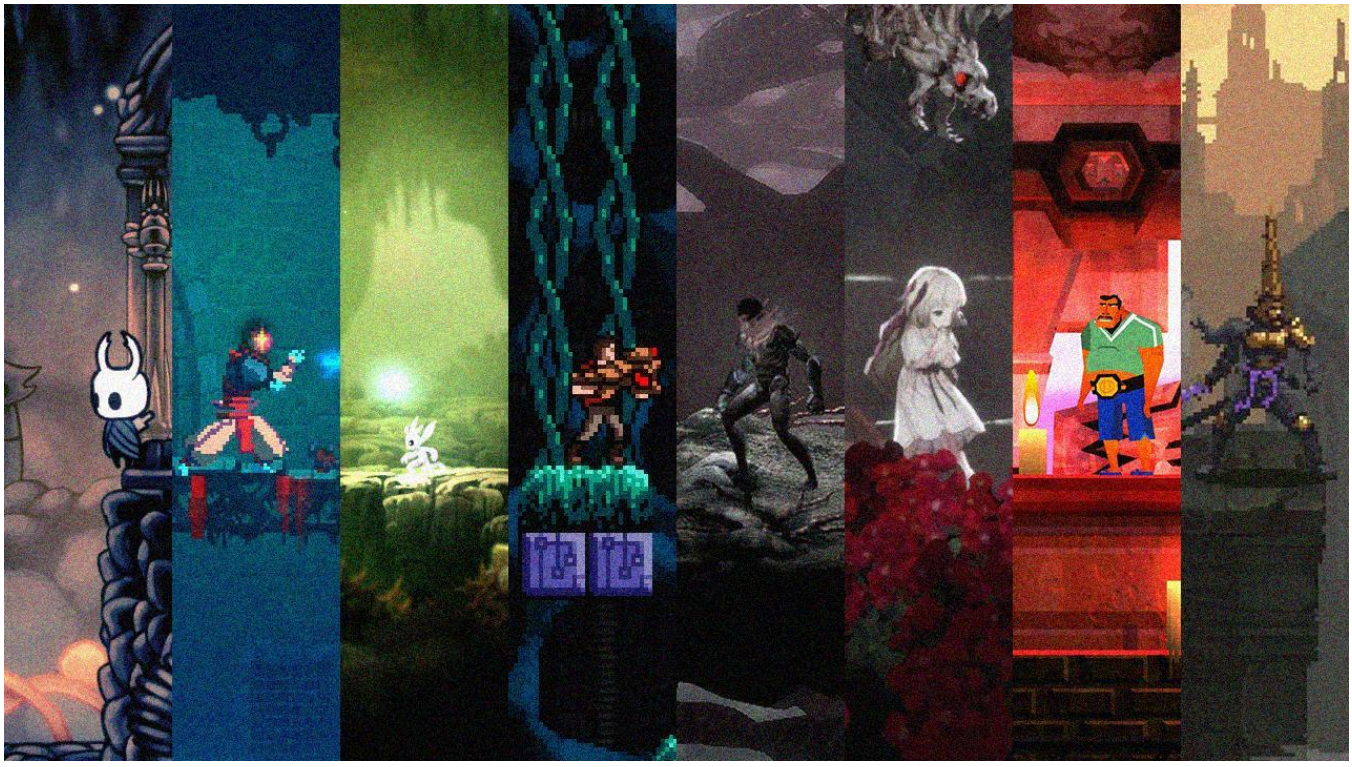


Рисунок 1.5 – Унікальність стилю жанру Metroidvania.

1.1.4 Аналіз аналогів

На сьогоднішній день на ринку існують дуже багато ігор жанру Metroidvania. Наприклад, «Hollow Knight», «Dead Cells», «Axiom Verge», «Ori and the Will of the Wisps». Звичайно, що це не всі представники популярного жанру, але їх було розглянуто через велику популярність та високі оцінки. Для розуміння того, що можна запозичити у вже створених іграх, а що було б непогано зробити краще, розглянемо кожний наведений аналог окремо та визначимо їх основні переваги та недоліки.

а) «Hollow Knight» (див. рис. 1.6);
   1) плюси;
      – великий цікавий світ для дослідження;
      – чарівна атмосфера та дуже красива графіка;
      – глибоко продумана бойова системи та ворогі;
      – захоплююча музика та дизайн звуків;
   2) мінуси;
      – занадто висока складність для деяких гравців;
      – недостатня зрозумілість деяких моментів, наприклад сюжетних завдань;

б) «Dead Cells»;
   1) плюси;
      – динамічний геймплей;
      – велика кількість різноманітної зброї та апгрейдів;
      – досить висока реіграбельність гри завдяки процедурної генерації рівнів;
   2) мінуси;
      – занадто висока складність для деяких гравців;
      – не всім може сподобатися стиль графіки;

в) «Axiom Verge»;
   1) плюси;
      – унікальний 8-бітний стиль;

- велика кількість різноманітної зброї та апгрейдів;
- цікавий, загадковий сюжет та атмосфера;

2) мінуси;

- деякі аспекти геймплею сьогодні можуть здаватися надто застарілими;
- обмежена масштабність у порівнянні з іншими іграми у цьому жанрі;

г) «Ori and the Will of the Wisps»;

1) плюси;

- вражаюча графіка та анімація;
- глибокий емоційний сюжет;
- розмаїття геймплею та велика кількість загадок;

2) мінуси;

- деякі проблеми з оптимізацією;
- бойова система іноді може здаватися занадто простою та повторюваною.

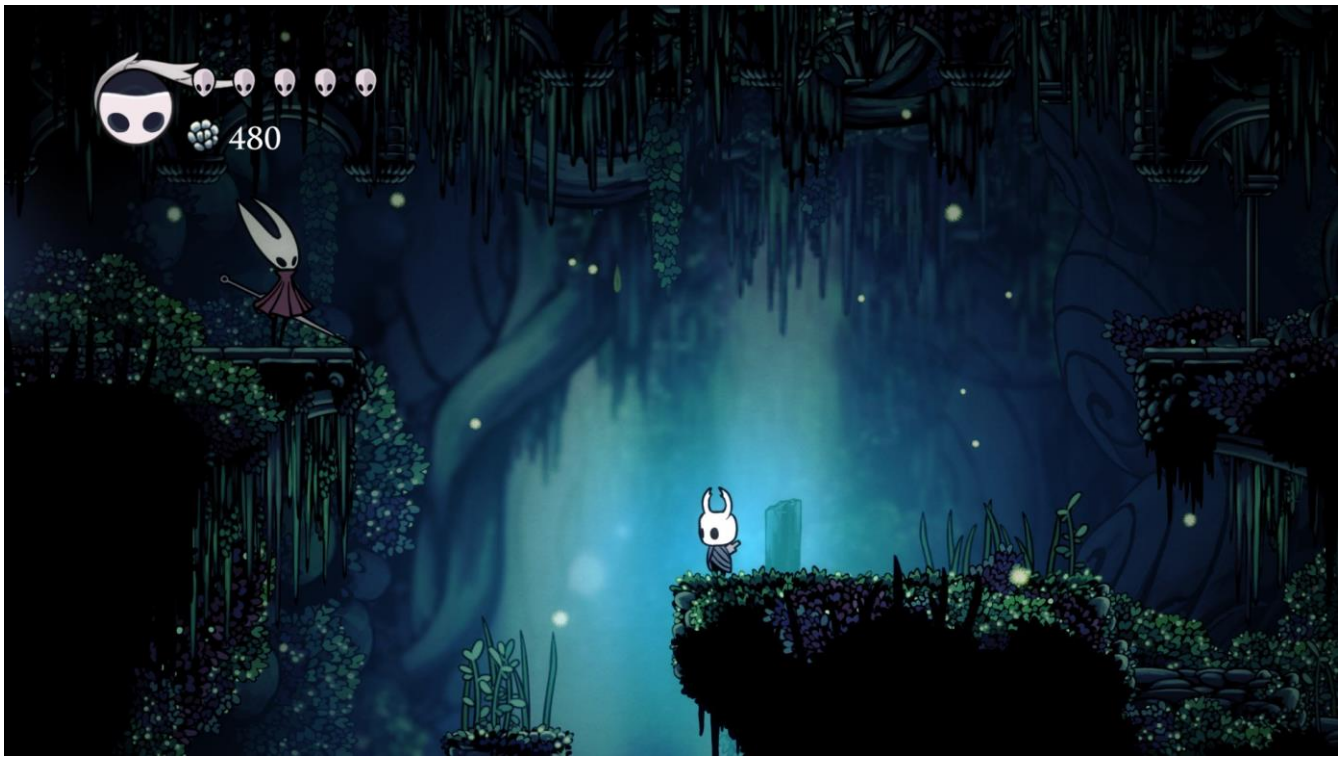


Рисунок 1.6 – Гра «Hollow Knight» 2017 року.

1.2 Виявлення та вирішення проблем

Як було з’ясовано існуючі аналоги, незважаючи на їх популярність, мають і свої недоліки. Саме тому створення нової гри у жанрі Metroidvania виправдане через декілька ключових причин, що вирішують певні проблеми, які можуть бути актуальними для сучасних гравців.

До проблем, що вирішує розробка нової гри можна віднести втому гравців від існуючих представників жанру. З’явлення нової гри у жанрі Metroidvania може принести свіжість у гейміндустрію та зацікавити гравців. Люди, які вже грали у старі ігри, можуть хотіти нових розваг та ігрового досвіду. Тож сама поява нової гри вже вирішує цю проблему.

Запити та інтереси гравців можуть змінюватися з часом. Нова гра у жанрі Metroidvania може відповісти на нові потреби гравців, які хотіли б бачити більшу складність геймплею, глибину сюжету та інноваційні механіки. Кожний представник жанру Metroidvania має свої унікальні особливості та механіки, що робить їх відмінними від інших. Отже, свіжі ідеї, інноваційні механіки і підходи до розробки можуть збагатити жанр та привернути і фанатів старих ігор, і людей не знайомих із жанром.

Важливо відмітити, що, як і будь який інший жанр, Metroidvania у деякий період може втрачати свою популярність через малу кількість нових ігор чи, навпаки, перенасиченість ринку схожими продуктами. Тому випуск нової гри може відродити інтерес до цього жанру серед гравців.

1.3 Постановка задачі

Для вирішення зазначених проблем необхідно розробити гру в жанрі Metroidvania з оригінальним і глибоким сюжетом, цікавим сеттингом, динамічною та інноваційною бойовою системою, гарним стилем, анімацією та музичним супроводом. Важливо забезпечити гравцеві легку та зручну навігацію в ігровому світі, для цього спроектувати зрозумілий інтерфейс, продумати та реалізувати діалогову систему для отримання інформації.

Він повинен мати можливість бути запущений на платформах Windows, macOS, Linux, Android та iOS, для більш широкого охоплення аудиторії. Ігровий програмний застосунок має працювати ефективно витрачати системні ресурси на різних пристроях.

1.3.1 Цільова аудиторія

Якщо описувати цільову аудиторію в загальних рисах, то цим жанром цікавляться люди обох статей, вікова група може бути досить широкою, але в переважно десь від 16 до 25 років. Саме люди такого віку найбільше цінують глибокий сюжет на складний геймплей. Жанр приваблює любителів наукової фантастики та фентезі, міфології та естетичного дизайну.

Більш детально можна розділити цільову аудиторію на такі групи:

– фанати класичних ігор жанру Metroidvania;
– любителі платформерів за їх динамічний геймплей;
– гравці, які шукають складні ігри;
– прихильники глибокого сюжету;
– любителі досліджувати ігровий світ;
– фанати мистецтва та музики в іграх.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

2.1 Формування концепції гри

Ігровий програмний застосунок у жанрі Metroidvania був названий «The Saga of Sigurd» – на честь головного героя. Гра розробляється з естетикою слов'янської міфології.

Основні події відбуваються під час правління Володимира Великого у Київській Русі. Головного героя звати Сігурд, він воїн скандинавського походження, давній знайомий князя, що бився з ним пліч-о-пліч у незліченних битвах та військових походах. Як всім відомо, та епоха відома однією з найважливіших подій в історії християнства – Хрещенням Русі. Але далеко не всіх влаштувала різка зміна культури і відмова від язичницької віри. Сігурд як послідовник старої віри різко проти діянь князя. Для того, щоб протистояти князю, він вирішує заручитися допомогою слов'янських богів.

Геймплей гри відбувається у 2D світі, наповненому вигаданими істотами слов'янської міфології. Весь ігровий світ поділений на три великі зони: Явь, Навь і Правь. Гравець має можливість за бажанням переміщатися між ними, досліджувати доступні локації, шукати ключові предмети, а також картки магії, що можна використовувати подальших боях. В рамках однієї локації герой може переміщатися по горизонталі за допомогою бігу та по вертикалі стрибаючи по платформах.

Бойова система гри представляє собою типову для ігор жанру Metroidvania бойову систему[4]. Герой має можливість ухилятися від атак ворогів завдяки стрибкам, а сам має два варіанта атаки – атака мечем та атака магією. Атака мечем проводиться у ближньому бою, і герой ризикує отримати удар від ворога. Атака магією проводиться з дистанції за допомогою знайдених магічних карток, але витрачає ману.

Інформацію від неігрових персонажів герой отримає через діалоги. Саме таким чином він отримує квести, виконання яких необхідно, щоб просунутись далі по сюжету. Також з деякими персонажами доступна окремо розроблена міні-гра для різноманітності геймплею.

2.2 Формування функціональних вимог

Щоб ігровий програмний додаток коректно працював, повноцінно та точно виконував усі задумані функції, було визначено список функціональних вимог додатку[5]. Також саме цей список вимог допоможе з вирішенням того, які аспекти системи необхідно тестувати з метою оцінки системи. Наведемо перелік основних функціональних вимог:

а) меню та інтерфейс;
   1) почати нову гру;
   2) зберегти прогрес гри;
   3) завантажити збережену раніше гру;
   4) поставити гру на паузу;
   5) відкрити вікно налаштувань;
   6) відобразити поточних здоров’я, мани та карточок магії;
   7) закрити гру;

б) переміщення персонажа;
   1) бігти по горизонталі вліво чи вправо;
   2) відстрибнути від платформи;
   3) зробити подвійний стрибок у повітрі;

в) взаємодія зі світом;
   1) перемістити між локаціями;
   2) підібрати картку на ігровому рівні;
   3) взаємодіяти з предметами;
   4) зруйнувати предмет на рівні;

г) бойова система;
   1) атакувати мечем у ближньому бою;
   2) атакувати магією у дальньому бою;
   3) ухилитися від атаки;

д) ворожі істоти;
   1) перемістити по горизонталі чи вертикалі;
   2) атакувати головного героя;

3) виконати спеціальний бойовий прийом;
4) померти при закінченні здоров’я;

е) діалогова система;
1) ініціювати діалог;
2) відобразити діалог;
3) перейти до наступної репліки;
4) завершити діалог;

ж) карткова міні-гра;
1) вибрати фракцію;
2) вибрати та встановити картку;
3) закінчитись перемогою чи програшом;
4) змінити музичне супроводження.

2.3 Формування системних вимог

Використання ігрового додатку передбачається на платформах Windows, macOS, Linux, Android та iOS. Тому було визначено мінімальні вимоги до системних ресурсів цих платформ для коректної роботи системи.

а) вимоги до операційної системи Windows;
1) Версія: Windows 7 64-bit або новіша;
2) Процесор: Intel Core i3 або аналогічний;
3) ОЗП: не менше 2 ГБ;
4) Відеокарта: сумісна з DirectX 10;
5) Місце на диску: не менше 2 ГБ доступного місця;

б) системні вимоги до macOS;
1) Операційна система: macOS 10.12 Sierra 64bit або новіша;
2) Процесор: Intel Core i3 або аналогічний;
3) ОЗП: не менше 2 ГБ;
4) Відеокарта: сумісна з OpenGL 3.3;
5) Місце на диску: не менше 2 ГБ доступного місця;

в) вимоги до операційної системи Linux;

1) Операційна система: Ubuntu 16.04 64bit або еквівалентні дистрибутиви;
2) Процесор: Intel Core i3 або аналогічний;
3) ОЗП: не менше 2 ГБ;
4) Відеокарта: сумісна з OpenGL 3.3;
5) Місце на диску: не менше 2 ГБ доступного місця;

г) вимоги до операційної системи Android для мобільного застосунку;
1) Версія: Android 5.0 або новіша;
2) Процесор: ARMv7 або аналогічний;
3) ОЗП: не менше 1 ГБ;
4) Графічний процесор: сумісний з OpenGL ES 2.0 з підтримкою WebGL;
5) Місце на диску: не менше 200 МБ доступного місця;

д) системні вимоги до iOS для використання мобільного застосунку;
1) Версія: iOS 10 або новіша;
2) Процесор: ARMv8 або аналогічний;
3) ОЗП: не менше 1 ГБ;
4) Графічний процесор: сумісний з OpenGL ES 2.0 з підтримкою WebGL;
5) Місце на диску: не менше 200 МБ доступного місця.

3 АРХІТЕКТУРА ТА ПРОЕКТУВАННЯ

3.1 UML проєктування ПЗ

Для моделювання функцій ігрового застосунку було розроблено діаграму прецедентів (див. рис. 3.1). Діаграма прецедентів – це UML-діаграма, яка використовується для того, щоб змоделювати з точки зору користувача функціональність системи. На діаграмі показуються прецеденти системи, які доступні користувачу, та те, якими способами користувач може взаємодіяти із системою.

На діаграмі можна побачити одного актора – користувача.

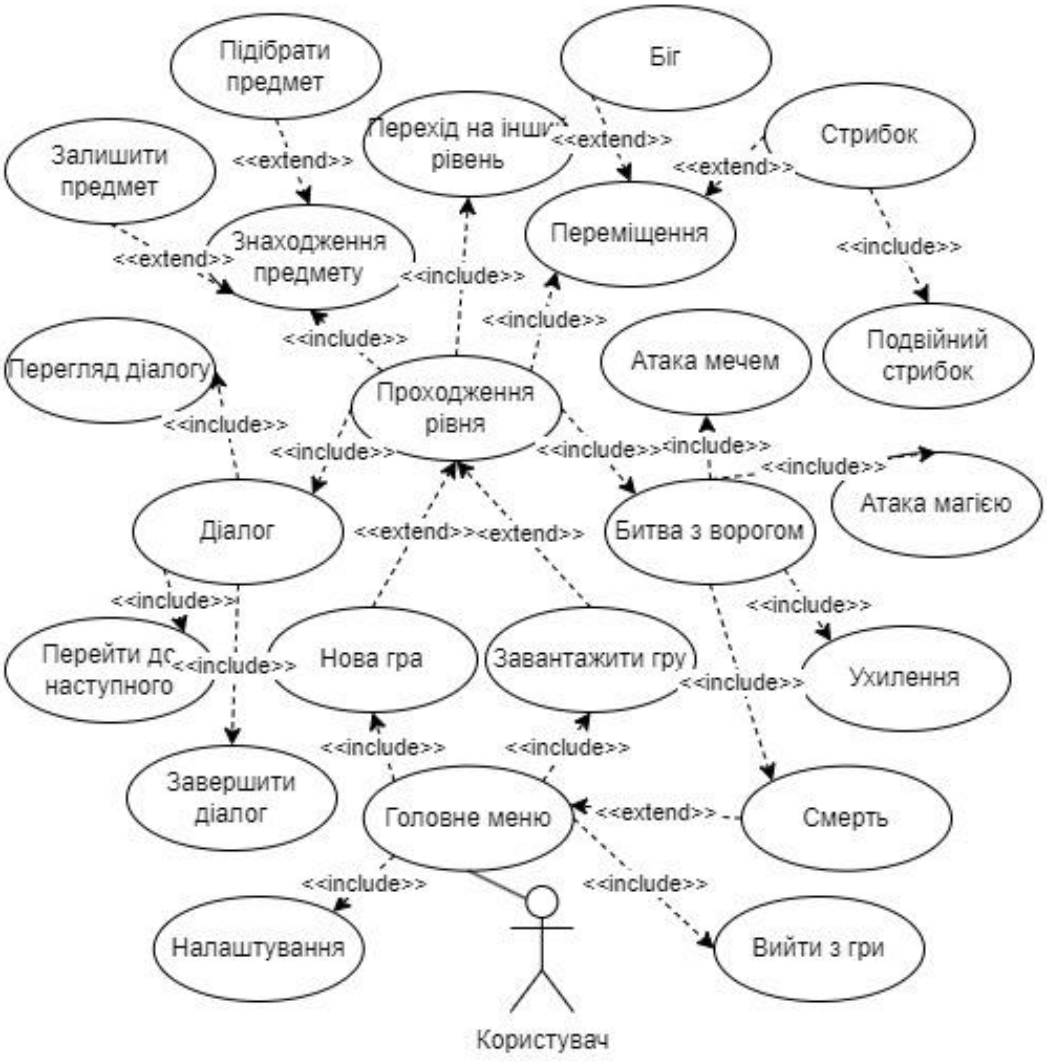


Рис. 3.1 – Діаграма прецедентів.

Серед варіантів використання:

- можливість розпочати нову гру, завантажити збережену гру, перейти до налаштувань чи вийти з гри;
- при проходженні рівню можна переміщуватись, битися з ворогом, розпочати діалог, знаходити предмети чи перейти на інший рівень;
- для переміщення можна бігти чи стрибати;
- в битві можна атакувати мечем, магією, ухилятися чи померти, та повернутися до меню;
- у діалозі можна переглядати діалог, перейти до нової репліки чи завершити діалог;
- можна знайти предмет, залишити його чи підібрати.

Було розроблено діаграму діяльності для моделювання роботи системи (див. рис. 3.2). Діаграма діяльності моделює потік роботи або процеси системи. Вона показує послідовність дій чи станів, які відбуваються протягом виконання певного завдання. Діаграма діяльності допомагає уявити послідовність операцій в системі та розуміти логіку роботи процесів.

Діаграма складається з наступних станів та процесів:

- головне меню. Серед вихідних процесів початок гри, завантаження гри, перехід до налаштувань та вихід;
- вікно налаштувань. Серед вихідних процесів збереження чи ні внесених змін та перехід до меню;
- рівень гри. Серед вихідних процесів перехід до меню, якщо закінчилося здоров’я, чи обирання дії;
- переміщення. Серед вихідних процесів повертання до рівня гри якщо не на землі, або перехід до стрибка чи бігу;
- битва. Серед вихідних процесів атака, а тоді повертання до рівню гри, або ухилення, тоді перехід до переміщення;
- діалог. Серед вихідних процесів продовження, тоді повернення до діалогу, або закриття, тоді перехід знов до рівня гри.

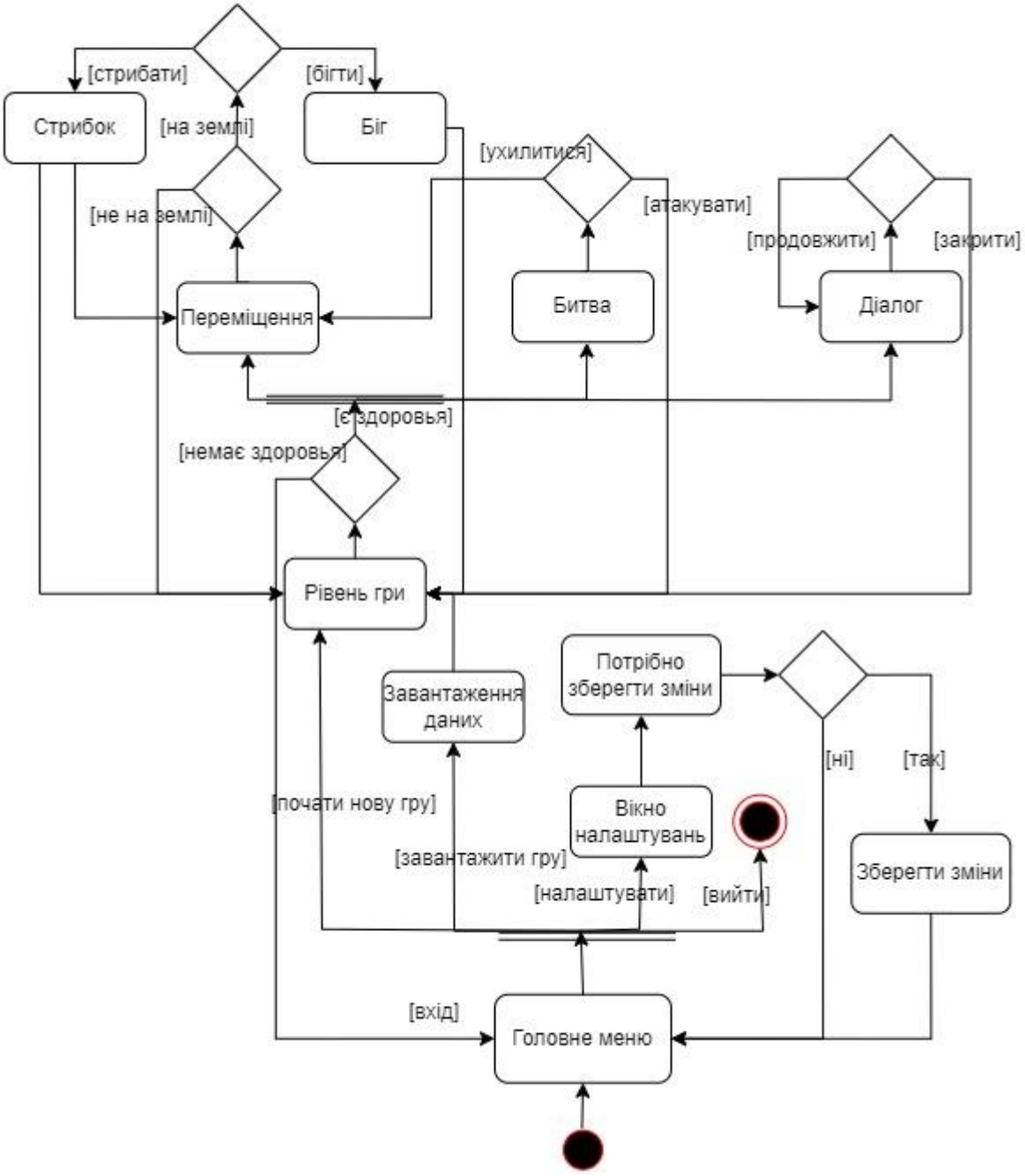


Рис. 3.2 – Діаграма діяльності.

Діаграма розгортання була розроблена для кращого розуміння фізичної структури системи (див. рис. 3.3). Діаграма розгортання вказує на фізичну архітектуру системи, демонструючи, як програмне забезпечення та апаратне засоби розміщені та взаємодіють між собою. Вона надає можливість дізнатися, як програмне забезпечення і апаратні засоби системи взаємодіють між собою, та як вони розміщені.

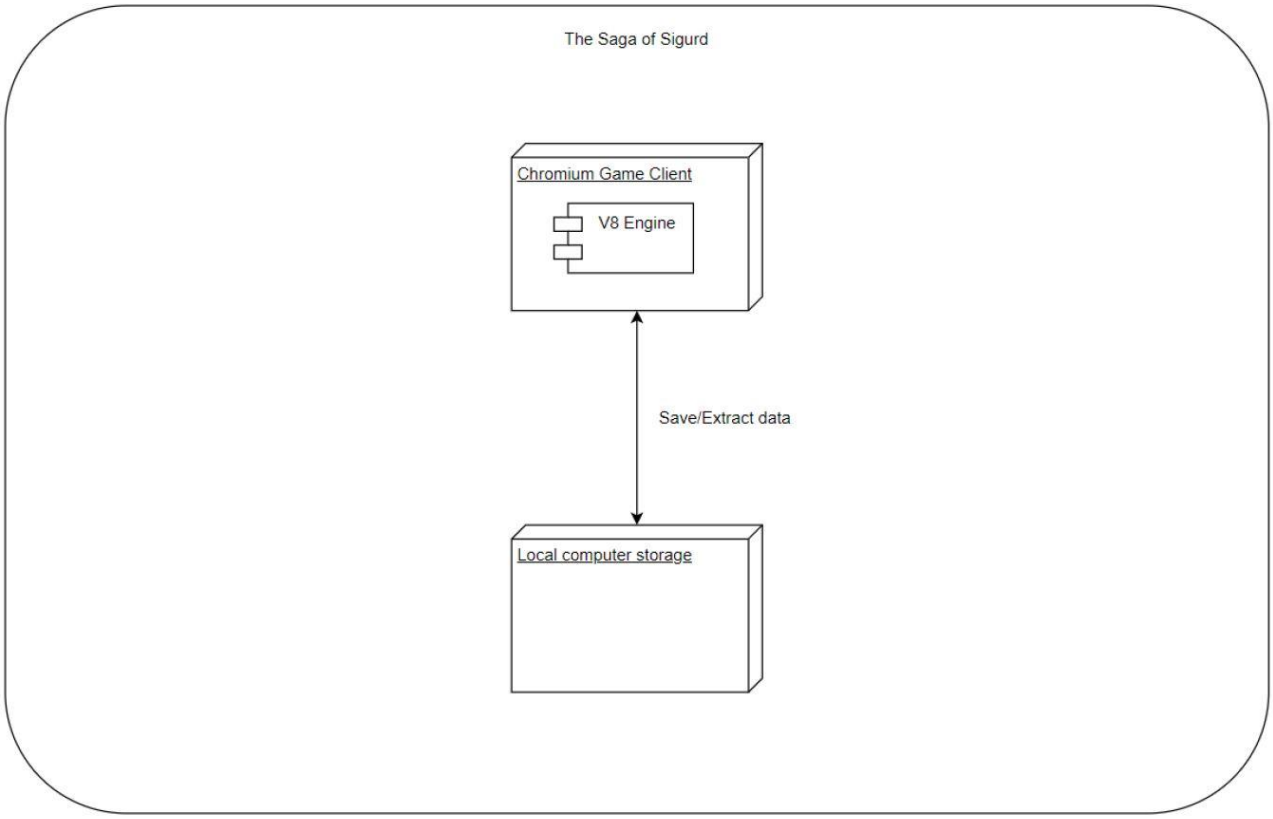


Рис. 3.3 – Діаграма розгортання.

Система складається всього з двох частин: локального сховища комп’ютера та клієнту ігрового застосунку на базі Chromium, що інкапсулює всередині движок V8 для інтерпретації JavaScript коду – логіку гри. Локальне середовище необхідно для збереження даних локально на комп’ютері користувача та отримання цих даних.

3.2 Проєктування архітектури ПЗ

Ігровий програмний застосунок на базі Chromium з движком V8. V8 є відкритим веб-движком, який використовується для виконання JavaScript коду[6]. Для оптимального та ефективного виконання коду на JavaScript він включає різні компоненти, такі як виконавчий двигун, оптимізатори та компілятори.

Також V8 використовує автоматичну систему зборки сміття, що допомагає в управлінні пам’яттю, оптимізацію використання кешу, визначенні непотрібних об'єктів та їх вивільненні для повторного використання.

V8 підтримується на різних платформах, включаючи Windows, macOS, Linux та Android, що робить його придатним для розробки мультиплатформенного ігрового застосунку[7].

Середовищем розробки гри є ct.js. Як було вже сказано він базується на движку V8 та мові програмування JavaScript. Ігровий застосунок розробляємий за допомогою має такі компоненти ct.js:

- сцени які представляють різні етапи гри. Наприклад, головне меню чи рівень гри. Сцена складається з об’єктів та логіки.
- об’єкти є основними будівельними блоками програми. Наприклад, головний герой, вороги, картки та платформи;
- події, які представляють логіку гри. Наприклад, натискання клавіши атака, дотик об’єктів і таймери;
- графіка та анімація, які відповідають за візуальну частину гри;
- звукові ефекти;
- модулі, що можуть розширити функціональність ct.js, додати нові інструменти розробки.

3.3 Проєктування структури зберігання даних

Для організації ресурсів у ігровому застосунку у ct.js, який використовується для розробки, застосовується так звані «content types» (типи даних) – спеціальні об’єкти, що допомагають з групуванням та типізацією даних. Вони дозволяють додавати та управляти такими основними ресурсами, як:

- Number (число);
- Short text (короткий текст);
- Textbox (довгий текст);
- Checkbox (булеве значення);
- Texture (текстура);
- Template (ігровий об’єкт);
- інші типи контенту.

Для програмного застосунку з метою групування інформації було спроєктовано 4 типу даних:

- тип даних Hero (головний герой). Цей тип даних має поля Health (здоров’я головного героя) тип Number, Mana (мана головного героя) тип Number, Inventory (інвентар головного героя);
- тип даних Dialogues (діалоги). Тип даних має поля Id (id діалогу) тип Number, Name (назва діалогу) тип Short text, Sprite (спрайт говорючого) тип Texture, isSpriteReversed (чи віддзеркалений спрайт) тип Checkbox, isLeft (чи спрайт зліва) тип Checkbox, Text (текст діалогу) тип Textbox, nextId (id наступного діалогу) тип Number, isOneTime (чи програється один раз) тип Checkbox, isPlayed (чи вже програвся) тип Checkbox;
- тип даних Card (картка). Має поля Id (id картки) тип Number, CardName (назва картки) тип Short text, CardHealth(здоров’я картки) тип Number, CardPower(сила картки) тип Number, CardFractionId (id фракції) тип Number, Image (зображення картки) тип Template;
- тип даних CardPlace (місце для картки). Цей тип даних має поля Place (номер місця) тип Number, isYou (місце твоє чи ворога) тип Checkbox, X (координата x) тип Number, Y (координата y) тип Number.

Типи даних Card та CardPlace не потребують збереження, тому що використовуються тільки у міні-грі. А інформація про типи даних Hero та Dialogues має десь зберігатись.

Для зберігання даних було обрано використовувати локальне сховище (Local Storage). Локальне середовище це механізм, який дозволяє зберігати дані на стороні клієнта. Цей інструмент надає можливість зберігати інформацію між сесіями, навіть після перезавантаження або закриття гри. Середовище розробки ct.js та мова JavaScrip дозволяють його використовувати[8].

Локальне середовище було обрано замість бази даних через деякі його плюси, а саме:

- його використовувати значно легше ніж підключати та працювати з базою даних;

– дані з локального середовища зазвичай завантажуються швидше, оскільки вони зберігаються на клієнтському пристрої.

Звичайно, у нього є і мінуси, але некритичні:

– локальне середовище має обмежений обсяг даних, що може бути збережено. Але у даному випадку великий обсяг не потрібен;
– дані можуть бути втрачені, якщо користувач ненавмисно виконає дії, що призведуть до видалення даних.

Отже локальне середовище ідеально підходить під зазначені цілі.

3.4 Приклади найцікавіших алгоритмів та методів

У розробці ігрового програмного застосунку у жанрі Metroidvania, застосування цікавих алгоритмів та методів може покращити якість геймплею та забезпечити захоплюючий ігровий досвід. Тому для реалізації основних механік гри було розроблено декілька оригінальних алгоритмів та методів. Наведемо найцікавіші з них:

а) Керування головним героєм. У грі має бути можливість керувати рухами головного персонажа;
   1) визначити, чи персонаж у діалозі, застряв або гра у паузі. Якщо так, то герой не зможе рухатися чи битися і йому знадобиться спочатку вийти з діалогу, паузи чи звільнитися;
   2) дізнатися, чи стоїть герой на землі, платформі, або іншій твердій поверхні;
   3) розрахувати швидкість руху героя в залежності від існуючої швидкості та прискорення. Тобто якщо герой хоче піти наліво, коли він йшов направо, він спочатку сповільниться, а тільки потім розвернеться;
   4) визначити, чи знаходиться герой у стані виконання якоїсь дії, наприклад, він стрибає чи б’є. Герой має завершити попередню дію перш ніж виконувати нову;

5) зрозуміти, чи дія вже закінчилась. Якщо так, то перейти ро стану очікування;
6) якщо виконані умови, виконати нову дію, наприклад, атакувати мечем чи магією, бігти, стрибати;
7) програти анімацію відповідно до дії;

б) Поведінка ворогів;
1) визначити, чи стоїть він на землі;
2) дізнатися, чи герой знаходиться у зоні бачення ворога;
3) розрахувати швидкість руху в залежності від існуючої швидкості та прискорення;
4) визначити, чи знаходиться герой у зоні атаки;
5) визначити, чи знаходиться ворог у стані виконання якоїсь дії, наприклад, він стрибає чи атакує. Як і герой, ворог має завершити попередню дію перш ніж виконувати нову;
6) дізнатися, чи достатньо мани для магічної атаки, якщо ні, то буде доступна тільки фізична атака;
7) якщо герой у зоні атаки, підібрати атаку. Атаки можуть відрізнятися в залежності від типу ворога;
8) програти анімації відповідно до дії;

в) Ініціалізація та проведення діалогу;
1) зупинити гру, щоб ігрові події не відволікали користувача від отримання інформації;
2) знайти потрібний діалог за його id;
3) знайти спрайт для діалогу, щоб гравець міг розуміти з ким він взаємодіє;
4) поділити текст на строки, щоб він вміщувався у межу ігрового інтерфейсу;
5) відобразити діалог для користувача;
6) зберегти інформацію про використання діалогу у локальному сховищі;

7) знайти наступний діалог за його id, та почати даний алгоритм з початку.

3.5 Створення дизайну системи

Розробка гарного та зрозумілого дизайну є важливим елементом успішної гри, так як він впливає на відчуття ігрового досвіду користувачів[9]. Правильно спроектований дизайн допомагає гравцю зорієнтуватися та зрозуміти всі функції представлені у грі, що прямо вплине на рівень їх задоволення. Крім цього дизайн дає можливість поглибитися в ігровий світ та повноцінно відчути його атмосферу. Опишемо декілька важливих макетів дизайну з точки зору функціональної наповненості та відповідності запланованим механікам.

Спочатку необхідно зробити дизайн головного меню (див. рис 3.4). Так як це буде перше, що побачить користувач, від дизайну меню буде залежати, чи перше враження про гру буде позитивним. Звісно, потрібно супроводити меню всіми важливими компонентами для реалізації його функціоналу.

На розробленому макеті меню можна побачити такі компоненти:

- назва гри;
- кнопка для початку нової гри;
- кнопка для завантаження збереженої гри;
- кнопка для переходу до налаштувань та кнопка для виходу з гри.
- фонова картинка меню на задньому плані.

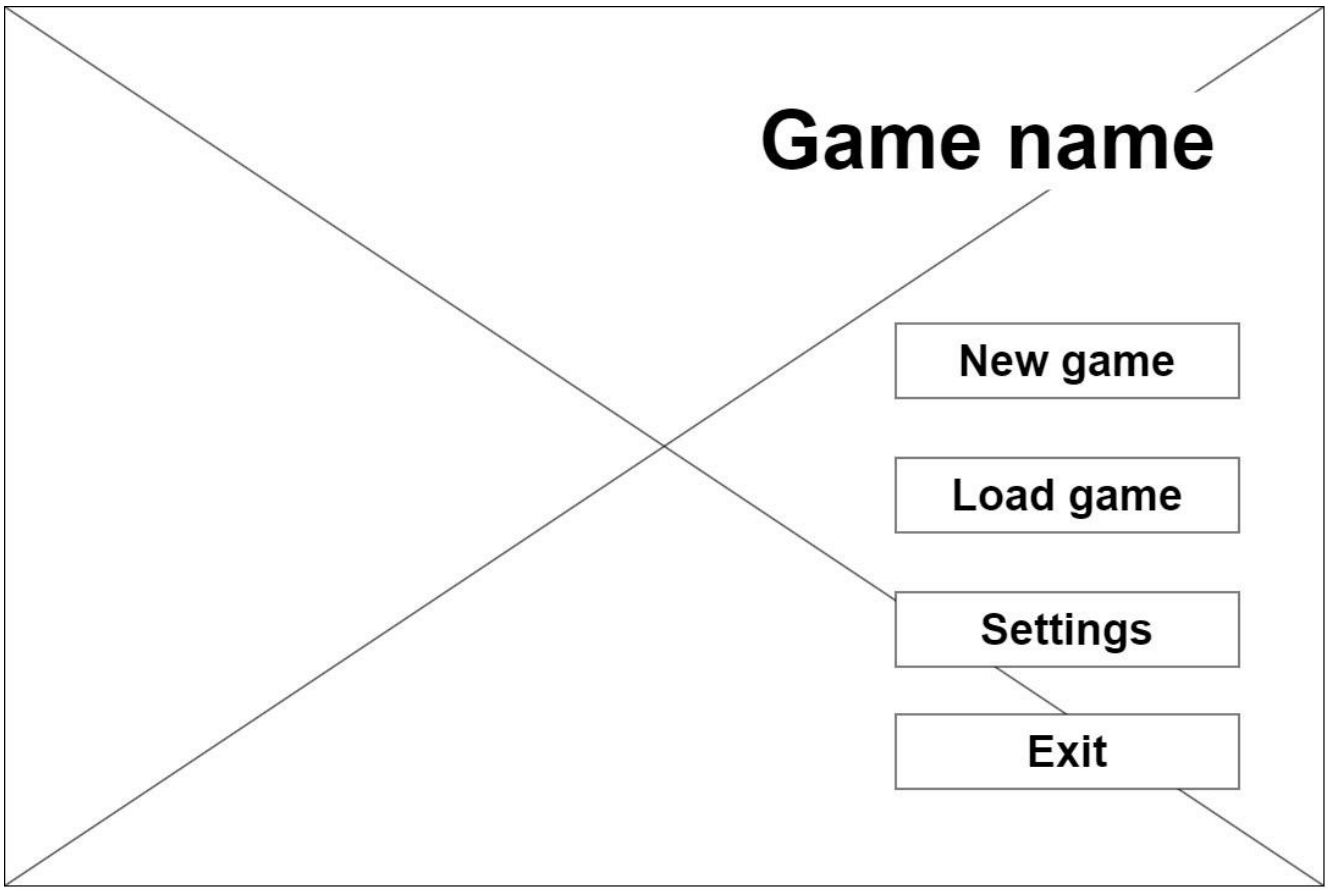


Рисунок 3.4 – Макет головного меню.

Другим важливим компонентом є дизайн основного інтерфейсу гри[10]. Саме цей інтерфейс гравець буде бачити протягом майже всієї гри, а також використовувати його для розуміння власних характеристик та вмінь, що дуже важливо під час зустрічі з ворогами.

Інтерфейс має розміщуватися поверх гри (див. рис 3.5). За допомогою нього має бути можна побачити:

- індикатор здоров’я головного героя;
- індикатор мани головного героя;
- обрані гравцем магічні карти, які він може використати у бою.

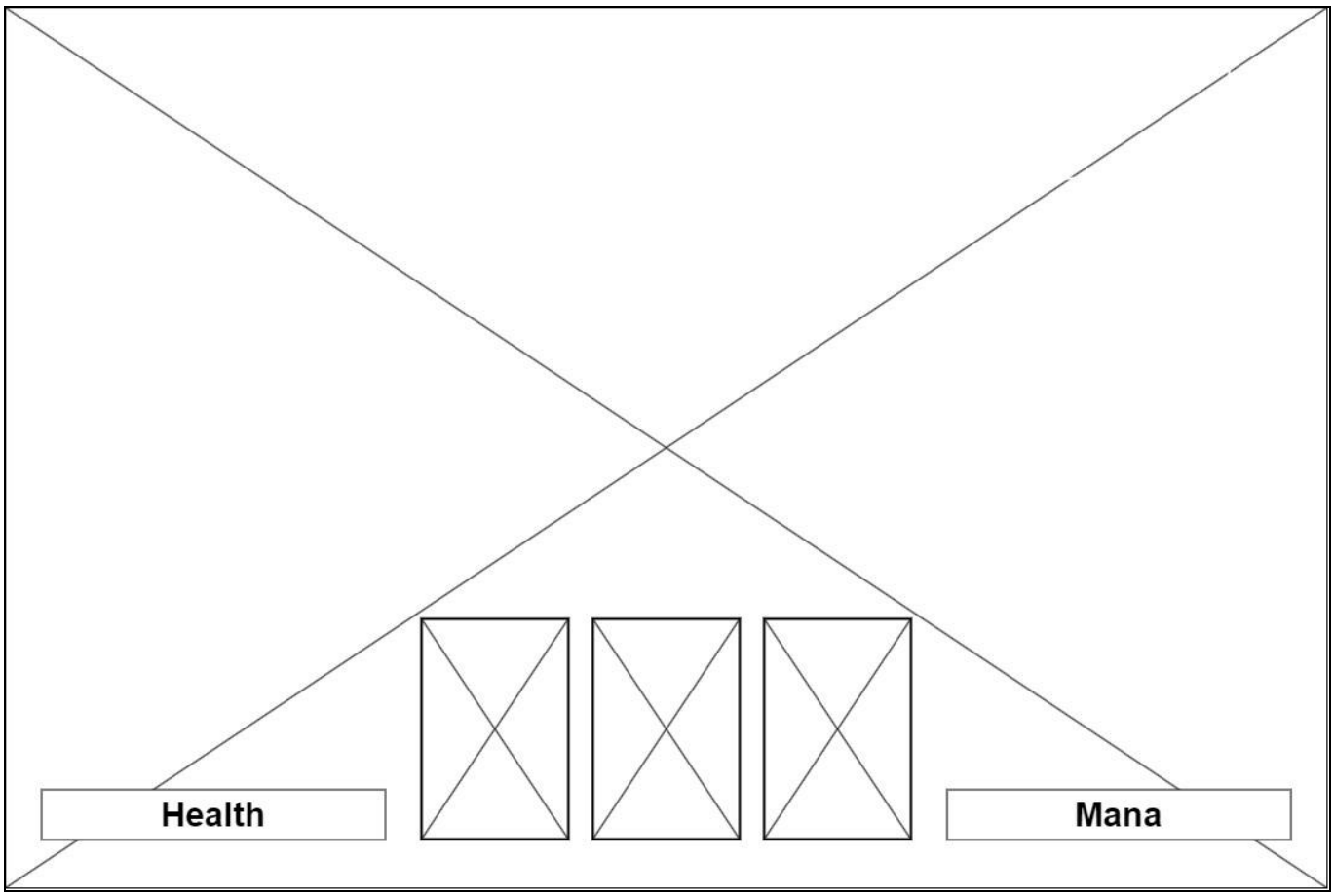


Рисунок 3.5 – Макет інтерфейсу гри.

За допомогою діалогів користувач буде отримувати важливу інформацію про сюжет, свої цілі та світ. Необхідно зробити його інформативним та зрозумілим, щоб користувач зміг поглибитися у історію та не загубитися у грі (див. рис 3.5).

Завдяки інтерфейсу можна буде побачити поле діалогу та текст на ньому. Також на цьому полі буде розміщуватись стрілка для переходу до наступної репліки. Крім цього зверху від діалогового поля буде відображатися картинка того, хто говорить.

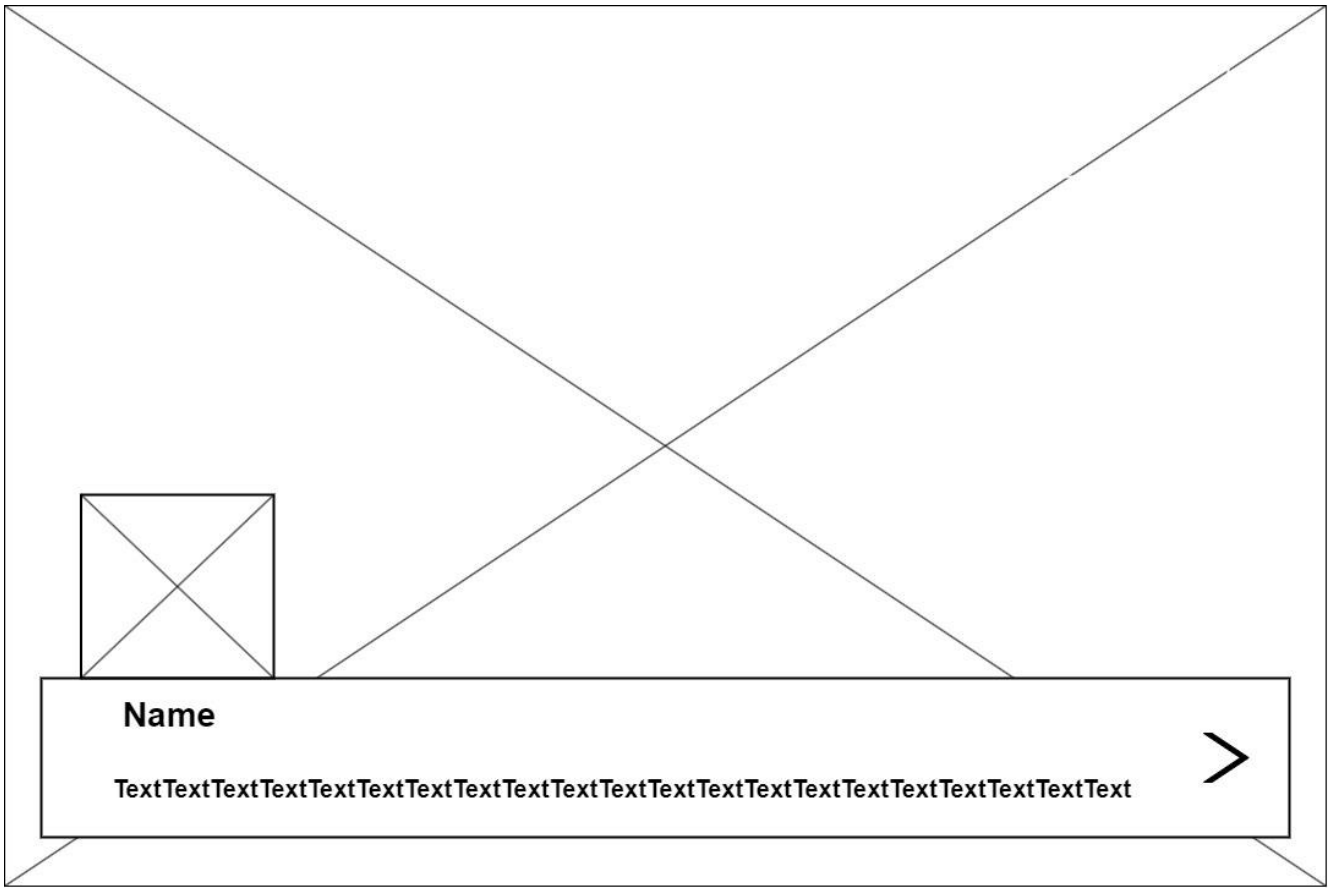


Рисунок 3.6 – Макет інтерфейсу діалогів.

Також розглянемо дизайн розроблений для міні-гри, в яку герой зможе пограти для різноманітності геймплею (див. рис 3.7).

На макеті можна побачити:

- свої розкладені картки внизу;
- картки ворога у зверху;
- посередині розташовані картки, які гравець буде обирати під час свого ходу;
- у куточку розташований олтар.

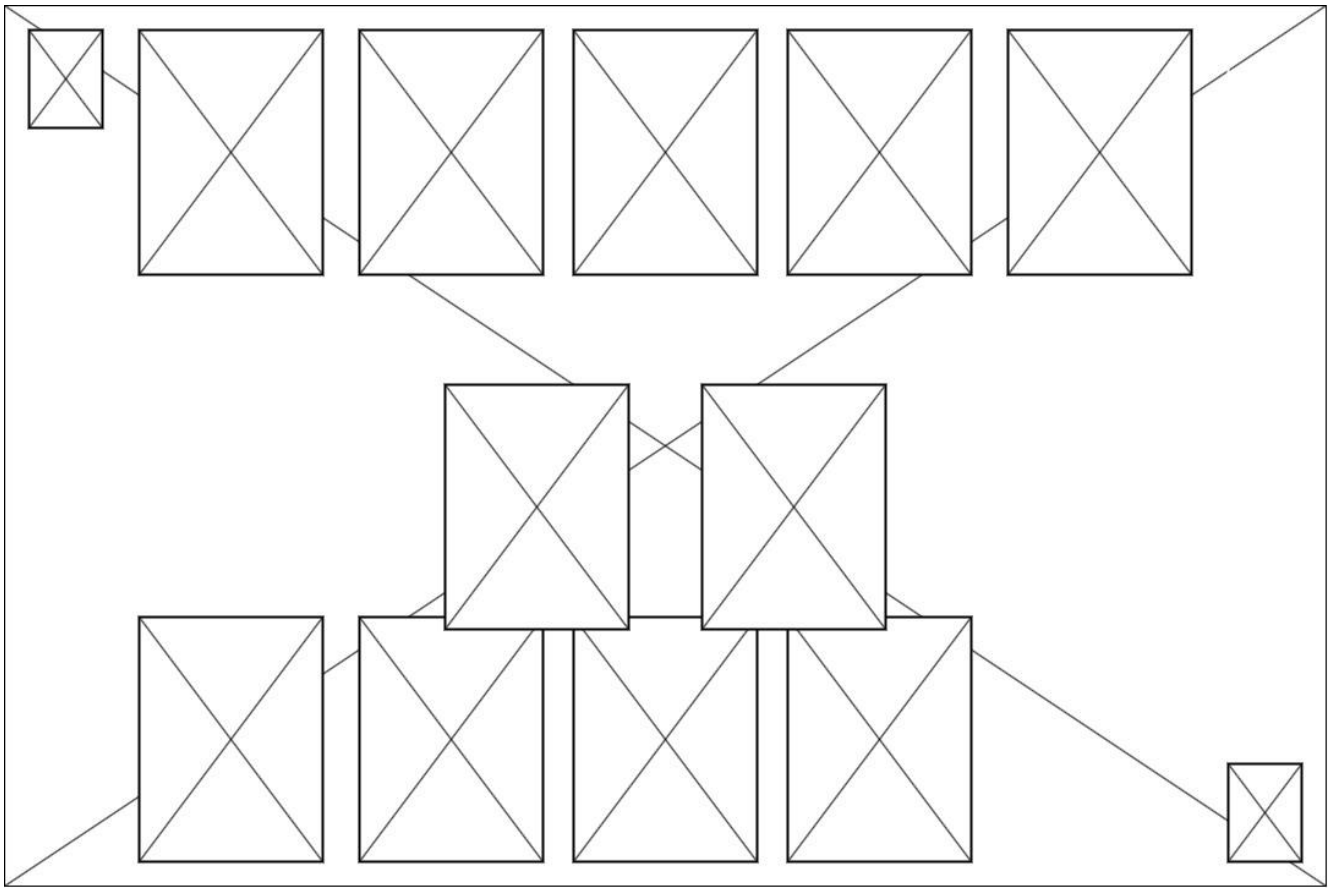


Рисунок 3.7 – Макет інтерфейсу діалогів.

Цей інтерфейс міні ігри хоч і не є основним, проте все ще дуже важливий для позитивного сприйняття гри користувачем.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Основні елементи гри

У результаті розробки було отримано ігровий програмний застосунок у естетиці слов’янської міфології у жанрі Metroidvania. Події гри відбувається у 2D світі, який складається з множини різних локацій, які можна досліджувати і переходити між ними.

Гра починається з головного меню, де у гравця є на вибір три опції (див. рис. 4.1). Кнопка виходу вимикає гру, кнопка початку нової гри завантажує перший рівень гри, а кнопка завантаження гри знаходить останній зіграний рівень у збережених даних і змінює кімнату на знайдену. Наведемо код кнопки завантаження:

```
if ('lastLevel' in localStorage) {
    console.log(localStorage.lastLevel)
    ct.rooms.switch(String(localStorage.lastLevel));
}
```

Рисунок 4.1 – Головне меню.

Кожний рівень представляє собою окрему локацію, що складається з платформ, по яких можна переміщатися, заднього фону, персонажів, ворожих істот, предметів (див. рис. 4.2).

Рисунок 4.2 – Приклад ігрового рівня.

Важливим частиною кожного рівня є невидимі об’єкти. Так сині об’єкти необхідні, щоб обмежувати переміщення персонажа, не давати впасти крізь текстури. Зелені об’єкти являють є тригерами, при торканні яких відбувається якась подія, наприклад, перехід на інший рівень, початок діалогу.

```
initializeDialogue(5);
this.kill = true
```

Основний ігровий інтерфейс складається з індикаторів здоров’я та мани, а також екіпірованих магічних карт (див. рис. 4.3).

Рисунок 4.3 – Інтерфейс гри.

Наведемо код призначення індикатору здоров’я даних персонажа:

```
let hero = ct.templates.list['Character1'][0];
this.health = hero.health;

let health = this.health;
if (health < 10)
    health = '00' + health;
else if (health >= 10 && health < 100)
    health = '0' + health;

this.healthBar.tex = this.healthBarNameTemplate + health;
```

Користувач має грати у гру, керуючи головним героєм (див. рис. 4.4). Як і інші живі істоти у грі, він має такі особисті характеристики як: здоров’я, мана, швидкість переміщення, висота стрибка, сила гравітації на нього, режим, в якому він знаходиться та інше.

```
this.health = 100
this.mana = 100;
this.maxSpeed = 10;
this.jumpSpeed = 30;
this.gravity = 1;
this.gravityDir = 90;
this.interactionPoints = 0;
this.isStuck = false;
this.dialogueMode = false;
this.attackMode = false;
this.magicMode = false;
this.jumpMode = false;
this.doubleJumped = false;
this.loop = true;
```

Рисунок 4.4 – Головний герой.

Перед виконанням будь яких дій, система перевіряє, чи стоїть гра на паузі, знаходиться головний герой у режимі діалогу або застряг. У цьому випадку головний герой не зможе виконувати таки дії, як переміщення та бій.

4.2 Створення системи переміщення

В рамках одного ігрового рівня герой під управлінням користувача може переміщатися вліво і вправо за допомогою бігу та по вертикалі стрибаючи по платформах.

Щоб зрозуміти, чи торкається головний герої твердої поверхні, система шукає перетинання з об’єктами типу «Solid». У цю групу входять платформи, земля, і будь-як об’єкти, на яких можна стояти.

```
var collisions = this.moveContinuousByAxes('Solid');
this.onGround = ct.place.occupied(this, this.x, this.y + 2, 'Solid');
```

Стрибати головний герой може тільки від твердих поверхонь. Якщо головний герой не робить іншу дію, система перевіряє, чи отримує він команду стрибка. Якщо він отримує команду, то перевіряє, чи стоїть герой на твердій поверхні, і виконує стрибок (див. рис. 4.5). У грі також було реалізовано механіку подвійного стрибка – персонаж може один раз стрибнути у повітрі перед приземленням.

```
if(!this.attackMode && !this.magicMode) {
    if(ct.actions.Attack.pressed) {
        …
    }
    else if (ct.actions.Jump.pressed) {
        if(this.onGround) {
            this.jumpMode = true;
        }
        else if (!this.doubleJumped) {
            this.vspeed = -this.jumpSpeed;
            this.doubleJumped = true;
        }
    }
}
```

Рисунок 4.5 – Персонаж стрибає.

Персонаж може переміщуватися по горизонталі як коли він торкається землі, так і коли він у повітрі. Це особливість жанру, яка полегшує переміщення по платформах.

Система отримує команду від гравця – рухатися вправо чи вліво, та вираховує цільову швидкість персонажа. Якщо команди руху не було або головний герой у цей час виконує іншу дію, то цільова швидкість буде дорівнювати нулю. Після знаходження цільової швидкості система, враховуючи поточну швидкість героя, знаходить нову поточну швидкість. Врахування поточної швидкості необхідно, щоб персонаж мав інерцію і рухався реалістично.

```
var targetHspeed = ct.actions.MoveX.value * this.maxSpeed;
if(this.attackMode || this.magicMode) {
    targetHspeed = 0;
}
if (this.hspeed > targetHspeed) {
    this.hspeed = Math.max(this.hspeed - this.accDec * ct.delta,
targetHspeed);
} else if (this.hspeed < targetHspeed) {
    this.hspeed = Math.min(this.hspeed + this.accDec * ct.delta,
targetHspeed);
}
```

Також в залежності від цільової швидкості додаток змінює напрям погляду персонажа – вліво чи вправо.

```
if (this.hspeed > 0.1) {
    this.scale.x = 1;
} else if (this.hspeed < -0.1) {
    this.scale.x = -1;
}
```

4.3 Створення бойової системи

Розроблена бойова система гри є досить типовою для ігор жанру Metroidvania. Крім переміщення під час битви персонаж може атакувати двома способами – мечем та магією. Обидві атаки можна проводити як на землі так і у повітрі.

Атака мечем проводиться в безпосередній близькості з ворогом, тому вона є біль ризиковою, бо можна отримати удар від ворога у відповідь. При отриманні команди атакувати мечем система перевіряє чи може герой зараз виконати цю дію. І якщо персонаж не атакує вже, то вона вмикає відповідний режим для героя – режим бою. Після цього система виконує анімацію удару і програє відповідний звук (див. рис. 4.6).

Рисунок 4.6 – Атака мечем.

```
if(!this.attackMode && !this.magicMode) {
    if(ct.actions.Attack.pressed) {
        this.attackMode = true;
        this.magicMode = false;
        this.jumpMode = false;
        ct.sound.spawn('Sword')
    }
    …
}
```

В режимі бою герої не може отримати шкоду здоров’ю, але сам торканням наносить шкоду здоров’ю ворогу. Вихід з цього режиму відбувається після завершення анімації.

```
if (this.attackMode == true) {
    if(other.timer1 <= 0) {
        other.timer1 = 0.8;
        other.health -= 40;
    }
}
```

Магічна атака проводиться з дистанції при використанні магічних карток, які знайшов гравець. Ця атака є більш безпечною, але витрачає ману персонажа. Атака починається, коли система отримує команду використати одну із трьох екіпірованих карток. Як і з атакою мечем, спочатку йде перевірка, чи може герой зараз виконати цю дію, а потім перехід до режиму магії, що вмикається та вимикається разом з відповідною анімацією.

```
if(!this.attackMode && !this.magicMode) {
    …
    else if(ct.actions.Card1.pressed || ct.actions.Card2.pressed ||
ct.actions.Card3.pressed) {
        this.attackMode = false;
        this.magicMode = true;
        this.jumpMode = false;
    }
    …
}
```

Види атаки магією можуть бути різними в залежності від картки. Наприклад, герой може створити магічну сферу, яка буде летіти вперед, поки не торкнеться об’єкта (див. рис. 4.7). Після цього вона зменшить кількість його здоров’я і зникне.

Рисунок 4.7 – Атака магією.

4.4 Створення логіки ворожих істот

Протягом проходження рівнів гравець майже постійно буде зустрічатися з різними ворожими істотами, яких знадобиться перемогти для просування далі. Кожний ворог має свої унікальні атаки, сильні та слабкі місця та інші особливості. Розглянемо декілька ворожих істот.

Перший ворог зустрічається ще на першому рівні – це мара, привид зі слов’янської міфології (див. рис. 4.8).

Рисунок 4.8 – Мара.

Вона є досить простим ворогом як з боку геймплею, так і з боку логіки – мара просто летить вперед і відіймає здоров’я героя, коли торкається його.

Коли мара торкається головного героя, йде перевірка чи він в бойовому режимі, в якому його не можна атакувати. Якщо ні, то у головного героя відіймається здоров’я та встановлюється тимчасова невразливість. Подібна схема атаки використовується і для інших ворогів.

```
if (other.attackMode == false) {
    if(other.timer1 <= 0) {
        other.timer1 = 1;
        other.health -= 15;
        ct.camera.shake = 0.5;
        ct.camera.shakeDecay = 0;
        ct.u.wait(1000).then(() => {
            ct.camera.shakeDecay = 1;
        })
    }
}
```

Інший, більш складний з точки зору логіки поведінки та геймлею ворог – волкодлак (див. рис. 4.9).

Рисунок 4.9 – Волкодлак.

Для нього, як і для головного героя, система визначає, чи торкається він землі або іншої поверхні. Також визначається відстань від нього до головного героя. Волкодлак побачить головного героя, коли той опиниться досить близько, і

тоді почне свій рух. Він переміщується виключно стрибками і під час стрибка робить удар лапами.

```
if(!this.jumpMode) {
    if(isClose || this.attackMode || this.magicMode) {
        …
    }
    else {
        this.jumpMode = true;
        this.vspeed = -this.jumpSpeed;
        this.hspeed = (-1) * distX / Math.abs(distX) * this.maxSpeed;
    }
}
```

Коли він дострибає до гравця, то почне атакувати (див. рис. 4.10).

Рисунок 4.10 – Фізична атака волкодлака.

Крім фізичної атаки волкодлак магічну – він може покликати іншого волкодлака на допомогу. Його мани вистачає на два виклика, і нові волкодлаки не можуть викликати собі допомічників. Атакувати будь якою атакою він може один раз у встановлений час. Логіка фізичної атаки дуже схожа на атаку мари, але він б’є не постійно, а тільки в певні кадри анімації удару та стрибка.

```
if(!this.jumpMode) {
    if(isClose || this.attackMode || this.magicMode) {
        this.hspeed = 0;

        if(this.timer2 <= 0 && !this.attackMode && !this.magicMode) {
```

```
                this.timer2 = 2;
                let action = Math.floor(Math.random() * (10)) + 1;
                if(this.mana > 0 && action <= 3) {
                    this.mana -= 100;
                    this.magicMode = true;
                    ct.sound.spawn('Howl')
                }
                else {
                    this.attackMode = true;
                }

                if (distX < 0) {
                    this.scale.x = 1;
                } else {
                    this.scale.x = -1;
                }
            }
        }
        …
}
```

Логіка атаки дуже схожа на атаку мари, але він б’є не постійно а тільки в певні кадри анімації удару та стрибка.

Третьою цікавою ворожою істотею є ирка. Цікавим цей ворог є через свою магічну атаку. Якщо гравець буде стояти занадто близько до ирки, то вона його повалить на землю та почне відіймати здоров’я (див. рис. 4.11).

Рикунок 4.11 – Магічна атака ирки.

Головний герой не зможе переміщатися, атакувати чи робити будь-які інші дії. Щоб звільнитися від захвату, потрібно багато разів натиснути кнопку взаємодії.

```
if (isDrinkingMode) {
    other.isStuck = true;
```

```
        …
        other.timer1 = 1;
        other.health -= 1;

        if(ct.actions.Interact.pressed) {
            other.interactionPoints += 1;
        }
        if(other.interactionPoints >= 10) {
            other.interactionPoints = 0;
            other.isStuck = false;
            this.magicMode = false;
        }
        …
}
```

4.5 Створення діалогової системи

Розроблена система діалогів має досить важливу функцію – надати можливість користувачу взаємодіяти із істотами ігрового світу та краще розуміти його історію через текстову інформацію.

Діалоги представлені у ігровому додатку як набір типів даних, які мають інформацію про номер репліки, текст репліки, ім’я того, хто говорить, його спрайт, чи повторюється діалог та інше (див. рис. 4.12).

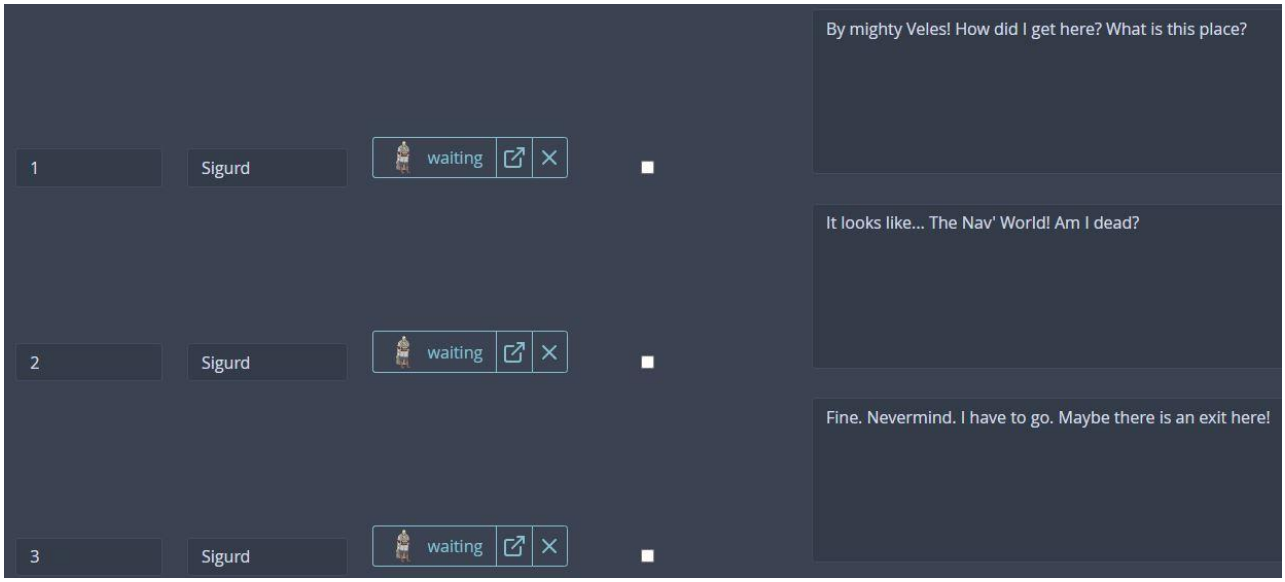


Рисунок 4.12 – Представлення діалогів у системі.

Для виклику репліки діалогу необхідно, щоб трапилась якась подія. Це може бути початок чи закінчення рівню, торкання об’єкту чи взаємодія з іншим персонажем.

При початку діалогу додаток шукає необхідну репліку за номером. У разі, коли репліка вже була вимовлена раніше і вона не може повторюватися, система закінчує діалог. Якщо ці умови не виконані, система переводить персонажа у режим діалогу і тимчасово зупиняє ігровий світ.

Далі йде відображення репліки діалогу для користувача (див. рис. 4.13). Знизу екрану з’являється поле для діалогу і виводиться текст ігровим шрифтом, також виводиться ім’я того, хто говорить, і відображається його іконка, яка в залежності від певних умов може бути розвернена чи переміщена.

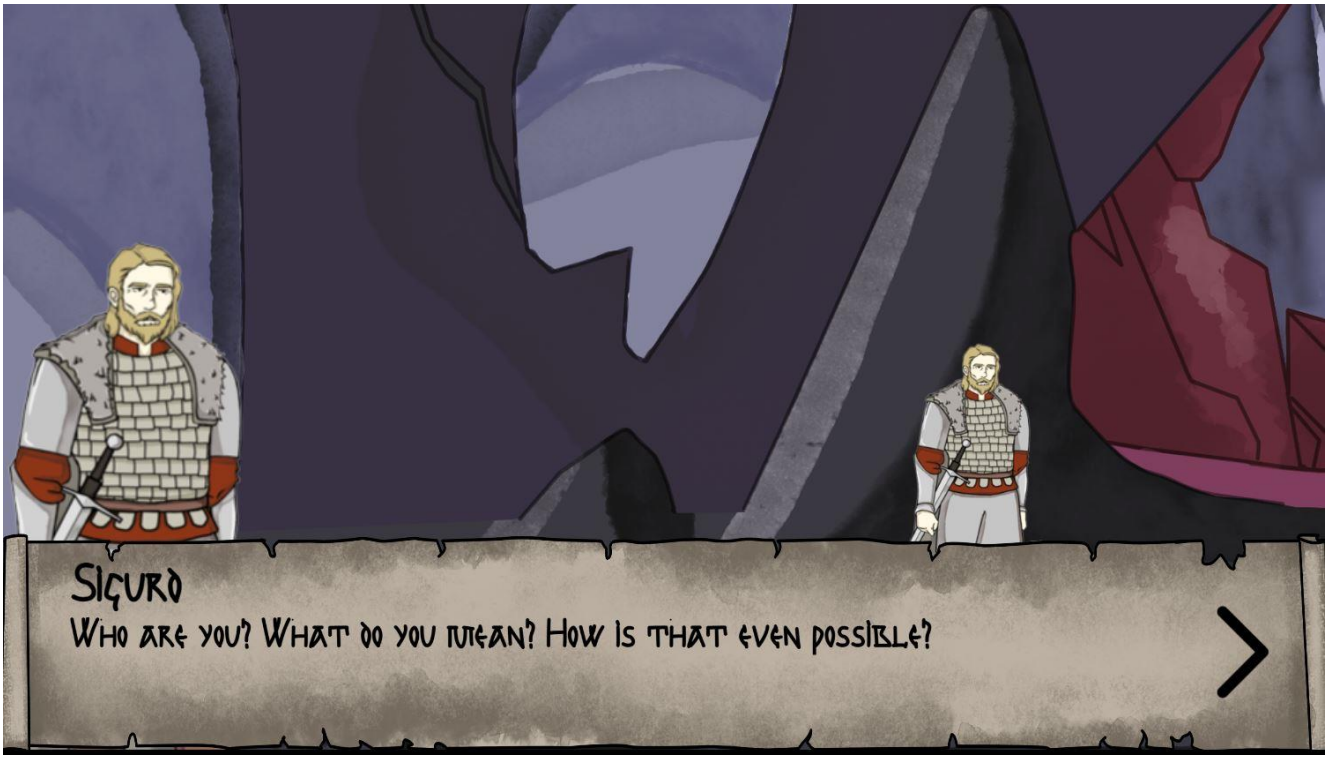


Рисунок 4.13 – Ігровий діалог.

Далі система зберігає інформацію про те, що репліка була вимовлена, у локальному сховищі.

```
var initializeDialogue = (startId) => {
    let dialogue = findDialogue(startId);
    if (dialogue.isOneTime && dialogue.isPlayed)
        return;
    let hero = ct.templates.list['Character1'][0];
    hero.dialogueMode = true;
    StopGame();

    let room = ct.rooms.append('Dialogues');
    createText(room, dialogue.Name, 'Beresta', 100, 810, 1);
    let textArr = splitString(dialogue.Text, 70);
    …
```

```
        sprite.scale.x = 2;
        sprite.scale.y = 2;
        if (dialogue.isSpriteReversed == true)
            sprite.scale.x *= -1;
        sprite.depth = -2;

        localStorage.setItem('currentDialogue', JSON.stringify(dialogue))
        dialogue.isPlayed = true
    }
```

Після закінчення репліки видаляється її візуальне відображення і йде перевірка, чи є ця репліка останньою у поточному діалозі. Якщо ні, то повторюються попередні дії але з новою реплікою, а якщо так, то головний герой виходить з режиму діалогу.

```
    var nextDialogue = () => {
        let currentDialogue =
JSON.parse(localStorage.getItem('currentDialogue'));
        currentDialogue.isPlayed = true;
        let nextDialogue = findDialogue(currentDialogue.nextId);
        ct.rooms.list['Dialogues'].forEach(r => r.destroy());
        if (nextDialogue && nextDialogue.Id != -1) {
            initializeDialogue(nextDialogue.Id);
        }
        else {
            let hero = ct.templates.list['Character1'][0];
            hero.dialogueMode = false;
            StopGame();
        }
    }
```

Перехід між репліками виконується за допомогою натискання стрілочки у правій частині поля діалогу.

5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Проєктування тестування

Головною метою тестування розробленого ігрового застосунку є перевірка повноти реалізації функціоналу, відповідності функціоналу системи до запланованого, зручність використання функцій ігрового застосунку для користувачів.

Мануальне, функціональне та нефункціональне тестування відбувається за допомогою методу «чорної скриньки». У цьому методі тестування програмного забезпечення, перевіряється робота програми без знання її внутрішньої побудови та схеми роботи.

Для тестування системи було обрано наступні види тестування:

- функціональне тестування;
- тестування інсталяції;
- тестування відмовостійкості та відновлення;
- тестування продуктивності;
- тестування інтерфейсу користувача;
- тестування конфігурації;

Тестування ігрового додатку відбувається на операційній системі Windows 10, Linux, macOS, у веб-браузері Google Chrome, а також на операційній системі Android та iOS.

Результатом тестування має бути детальний звіт, який надає розробникам та користувачам програмної системи повноцінне розуміння щодо якості функціоналу, можливості використання системи.

5.2 Проведення тестування

5.2.1 Тестування інсталяції

Метою тестування інсталяції є забезпечення того, що система успішно встановлюються, деінсталюється та працює належним чином у цільовому середовищі.

Тестування інсталяції включає наступні кроки:

- запустити інсталятор ігрового додатку і перевірити, чи є коректним запуск процесу інсталяції;
- переконатися, що існує можливість додаткового налаштування при інсталяції, наприклад, вибір місця установки;
- перевірити, що пройшов без помилок процес завантаження ресурсів гри, таких як графіка і звук;
- відкрити додаток і впевнитися, що він запускається без будь-яких проблем та інтегрується з операційною системою;
- видалити встановлений ігровий додаток та переконатися, що деінсталяція пройшла успішно;
- знов запустити інсталятор та встановити гру. Також знов перевірити роботу додатку після встановлення.

5.2.2 Функціональне тестування

Основною метою функціонального тестування є перевірка того, що програмне забезпечення виконує всі заявлені функції відповідно до зазначених вимог.

З основних функціональних галузей ігрового застосунку, які необхідно протестувати, можна виділити:

- меню та інтерфейс гри;
- переміщення персонажа;
- бойова система;
- взаємодія гравця зі світом;
- поведінка ворожих істот;
- анімації ігрових об’єктів;
- діалоги;
- міні-гра.

Так як наведені функції є основними у додатку, тому найбільша частина помилок буде виявлена під час тестування саме цих галузей. Коректна робота цих функцій є критично важливою для забезпечення якості програмної системи, що розробляється.

5.2.3 Тестування інтерфейсу користувача

Метою тестування інтерфейсу користувача є забезпечення того, що інтерфейс користувача програмного забезпечення є зручним, функціональним та інтуїтивно зрозумілим для кінцевих користувачів.

Тестування інтерфейсу користувача включає наступні кроки:

- оцінити ефективність навігації в грі, перевірити, чи всі елементи меню та інтерфейсу є доступними та працюють коректним чином;
- переконатися, що дизайн та анімації покращують досвід користувача під час проходження гри;
- переконатися, що різні способи введення, а саме клавіатура, миша, контролери, реагують на команди від користувача.

5.2.4 Тестування відмовостійкості та відновлення

Метою тестування відмовостійкості та відновлення є забезпечення того, що програмний додаток може витримувати відмови і швидко відновлюватися після них.

Тестування відмовостійкості та відновлення включає наступні кроки:

- самостійно викликати помилки системи і переконатися, що додаток їх виявляє та відновлює працю;
- забезпечити, щоб додаток правильно реагував на відмови програмних або апаратних компонентів, а саме відключення пристрою введення, зменшення ресурсів системи;
- переконатися, що додаток ефективно обробляє великі збережені;

– вимкнути ігровий додаток від час процесу гри та впевнитися, що додаток може відновити працю та завантажити останній збережений стан гравця.

5.2.5 Тестування продуктивності

Метою тестування продуктивності є визначення, оцінка та покращення ефективності роботи системи під різними навантаженнями для забезпечення стабільності геймплею.

Тестування продуктивності включає наступні кроки:

– протестувати, як швидко гра працює при різних конфігураціях системи переконатися, що фреймрейт є задовільним навіть на непотужних системах;
– визначити швидкість завантаження та вивантаження ресурсів гри, таких як рівнів, текстур, відео;
– зробити тестування продуктивності системи при великій кількості одночасних подій, ігрових об’єктів, їх взаємодій та анімацій.

5.2.6 Тестування конфігурації

Метою тестування конфігурації є забезпечення того, що програмний додаток працює належним чином у різних конфігураціях обладнання та програмного забезпечення.

Цей вид тестування має наступні пункти:

Запустити ігровий додаток на різних платформах, таких як Windows, macOS, Linux, Android та iOS, і оцінити коректність праці додатку на кожній з платформ;

– протестувати сумісність додатку з різними конфігураціями системи;
– переконатися, що додаток успішно адаптується до різних роздільних можливостей екранів і працює коректним чином у віконному та повноекранному режимах.

5.3 Результати тестування

Наведемо приклад помилок, знайдених при функціональному тестуванні (див. табл. 5.1). Помилки були виправлені після тестування.

Таблиця 5.1 – Помилки знайдені при функціональному тестуванні додатку

№	Назва	Серйозність	Пріоритет	Фактичний результат	Очікуваний результат
1	У діалозі гра видає помилку	Блокуючий	Високий	У діалозі гра може видати помилку	Діалог має проходити без помилок гри
2	Діалог виходить за межі екрану	Тривіальний	Низький	Текст діалогу іноді не вміщується у межах екрану	Весь текст діалогу завжди вміщується у межах екрану
3	При ударі втрачається здоров’я	Критичний	Високий	При ударі ворога головний герой втрачає здоров’я	При ударі ворога головний герой не втрачає здоров’я
4	Нанесення фізичної шкоди магією	Значний	Середній	Головний герой наносить фізичну шкоду при використанні магії	При використанні магії головний герой наносить шкоду тільки магією
5	Зависання анімації стрибка	Критичний	Високий	Анімація головного героя зависає після стрибка	Анімація головного героя після стрибка змінюється на анімацію очікування
6	Перекриття діалогу інтерфейсом	Блокуючий	Високий	Інтерфейс перекриває діалог	Діалог розташований поверх інших елементів інтерфейсу
7	Діалог на першому рівні не розпочинається	Незначний	Низький	Діалог на кінці рівня не розпочинається	Діалог розпочинається на кінці рівня
8	Повтор діалогу у домі	Незначний	Низький	Відтворення однакового діалогу під час кожного завантаження гри і відвідування будинку	Даний діалог програється тільки один раз після рівня з навчанням

6 ВПРОВАДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

6.1 Соціальні мережі

Створення акаунтів гри у соціальних мережах та поширення інформації про розробку гри має декілька важливих переваг:

- публікації про процес розробки, концепт-арти, скріншоти та відео з гри можуть зацікавити у потенційних гравців;
- соціальні мережі дозволяють розробникам отримувати безпосередній зворотній зв’язок від потенційних гравців. Це допомагає зрозуміти, які аспекти гри найбільше подобаються або потребують покращення;
- акаунти в соціальних мережах створюють платформу для об’єднання фанатів гри. Це дозволяє формувати лояльну спільноту, яка може підтримувати гру на різних етапах розробки;
- соціальні мережі є потужним інструментом для просування гри. Правильна маркетингова стратегія може допомогти залучити більше уваги до гри, збільшити продажі та покращити її відомість;
- соціальні мережі дозволяють швидко та ефективно інформувати спільноту про оновлення, виправлення помилок, нові функції та майбутні події.

Всього було обрано створити та вести акаунти гри у наступних соціальних мережах: Facebook, Instagram, Telegram, YouTube, Twitter. Такий вибір був зроблений через їх велику популярність.

Незважаючи на всі зусилля з популяризації гри, кількість зацікавленої аудиторії залишалася на досить низькому рівні. Але це надало можливість отримати досвід з взаємодії з аудиторією. Окремо хочеться відзначити соціальні мережі Facebook та Telegram.

Facebook опинився найменш активною соціальною мережею, тому що цільова аудиторія майже не користується нею (див. рис. 6.1). Але у майбутньому через неї про проєкт зможе дізнатися зарубіжна аудиторія. Акаунт у Telegram, з

іншого боку, став найбільш популярним, через велику популярність платформи у аудиторії (див. рис. 6.2).

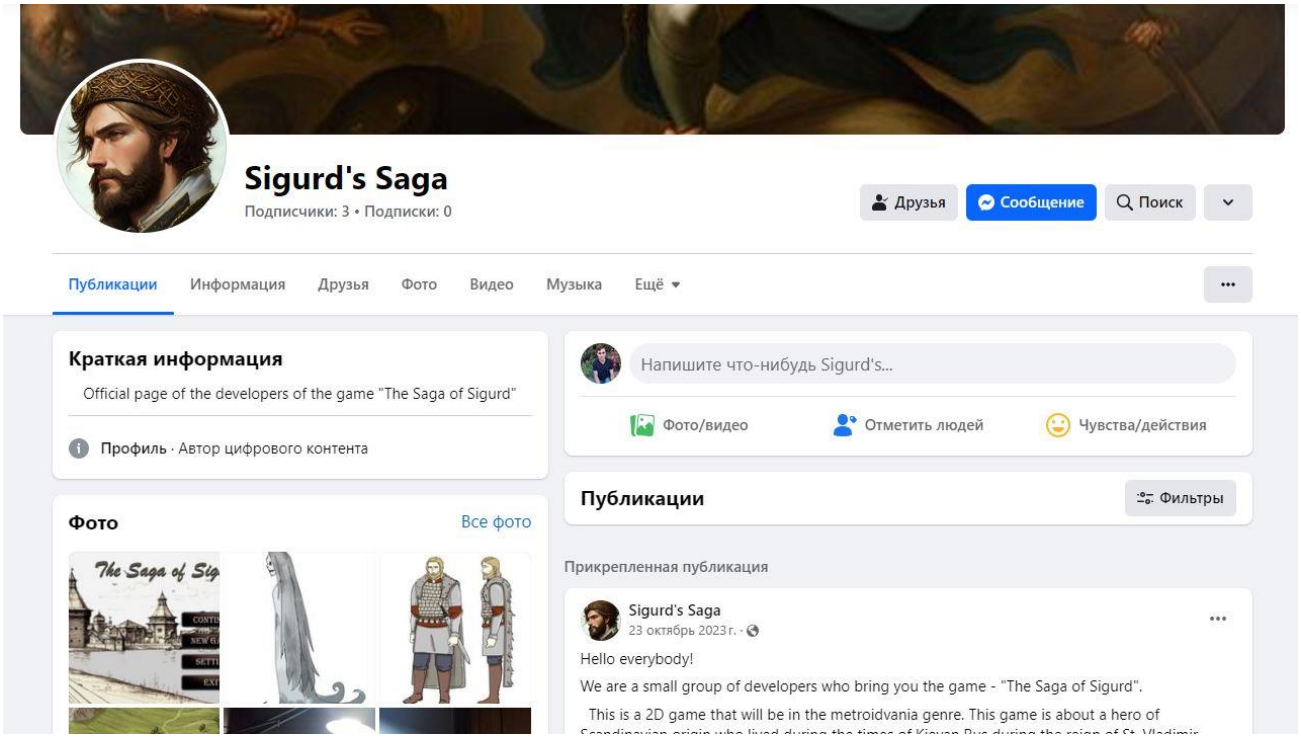


Рисунок 6.1 – Акаунт гри у Facebook.

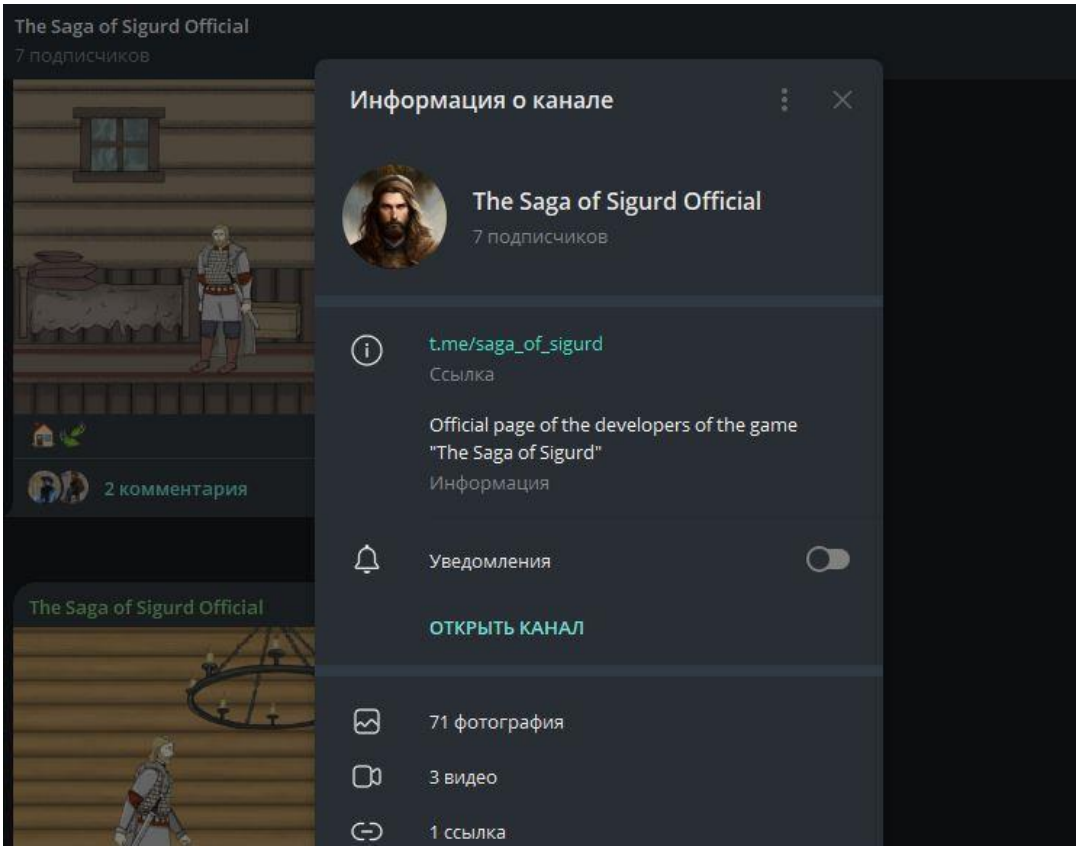


Рисунок 6.2 – Акаунт гри у Telegram.

6.2 Вивантаження на платформу

Ігрові платформи надають доступ до великої аудиторії гравців, що підвищує шанси на те, що розроблену гру помітять та зіграють. Звичайно вони мають інструменти для просування ігор, такі як виділені секції для ігор, що сприяє збільшенню популярності. І, що не менш важливо, завантаження гри на ігрову платформу дозволяє монетизувати її через продажи, рекламу чи донати від користувачів.

У якості платформи для розробленого програмного застосунку було обрано itch.io. Itch.io – це досить популярна ігрова платформа для розповсюдження інді-ігор, що надає інструменти для їх публікації, продажу та просування (див. рис. 6.3).

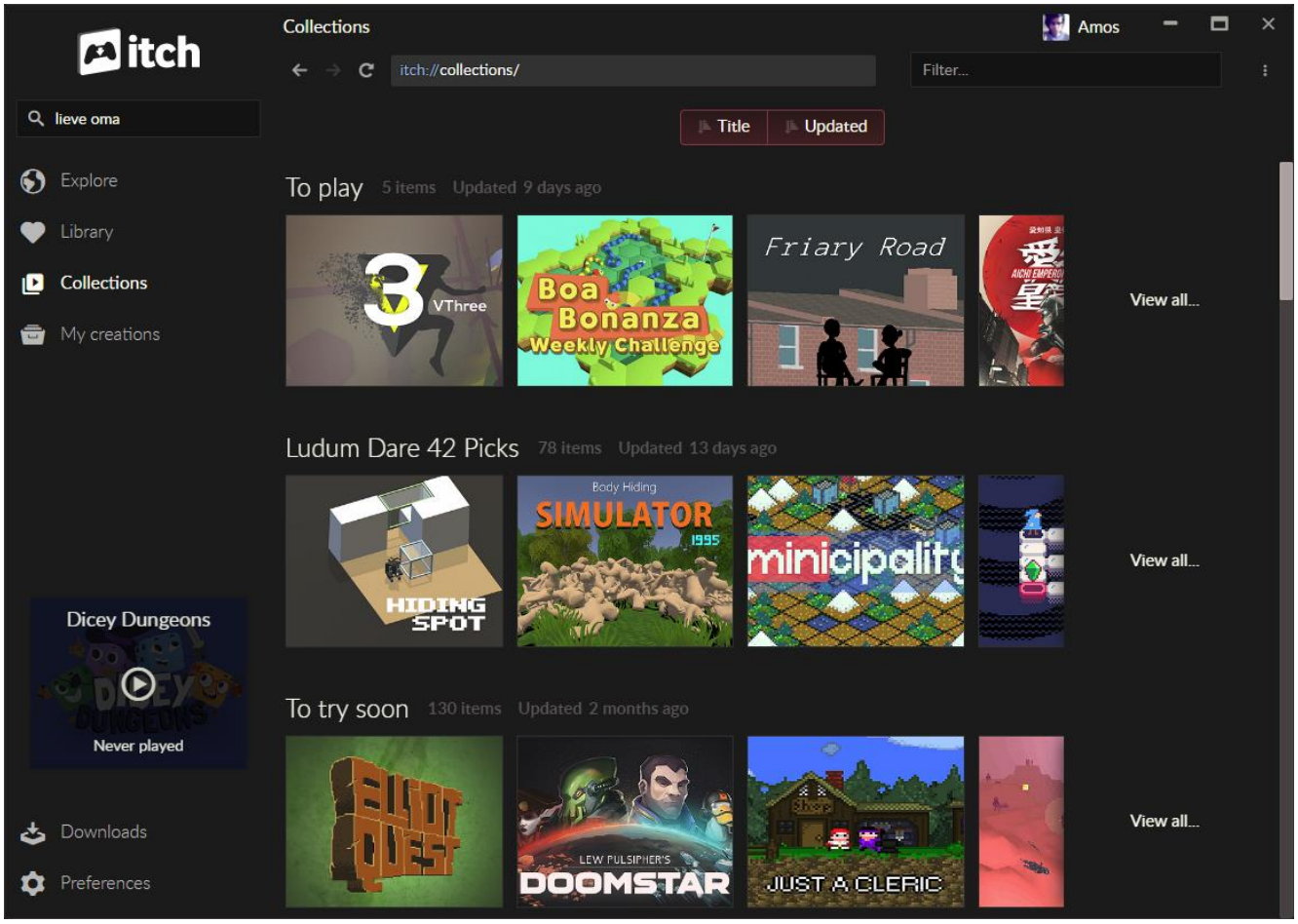


Рисунок 6.3 – Платформа itch.io.

Обрана ігрова платформа підтримує як безкоштовні, так і платні ігри. І надає розробнику повний контроль над ціноутворенням, знижками і розподілом

прибутку. Розмір комісії платформи зазвичай становить 10%, але вона може бути налаштована за бажанням.

Ігри на itch.io можуть бути розроблені для різних операційних систем, таких як Windows, macOS, Linux, а також для веб-браузерів. Це дуже важливо, тому що саме для цих платформ розроблений ігровий додаток.

Також платформа пропонує різні інструменти для управління завантаженнями і інтеграції з більш відомими сервісами, а крім цього інструменти комунікації з гравцями – stch.io підтримує форуми, коментарі, рейтинги та огляди. Ця ігрова платформа дозволяє зібрати активну спільноту гравців за допомогою низького бар’єру для входу, що сприяє підтримці проекту і обміну досвідом.

З її недоліків можна виділити те, що вона має значно меншу аудиторію у порівнянні з великими ігровими платформами, що може негативно сказатися на популяризації гри.

ВИСНОВКИ

У результаті дослідницької роботи було спроектовано та розроблено ігрові механіки та систему діалогів для ігрового програмного застосунку в жанрі Metroidvania з елементами карткової гри. Отримана система представлена у вигляді програмного застосунку на операційній системі Windows 10, у веб-браузері Chrome та на мобільному пристрої на платформі Android.

В процесі аналізу предметної галузі було виявлено основні риси та поширені проблеми жанру ігор Metroidvania, що дозволило зрозуміти основні вимоги до розроблюваного ігрового застосунку та задачі розробки. Враховуючи обмеженість ресурсів та часу розробки, було сформульовано функціональні вимоги до ігрових механік та діалогової системи програмного застосунку. З урахуванням обмеженості системних ресурсів кожної з платформ, на яких має працювати застосунок, було підібрано відповідні вимоги до системних ресурсів. Було проведено UML-проєктування ігрового застосунку, розроблено архітектуру системи та структуру зберігання даних, а також наведено найцікавіші алгоритми та методи для вирішення поставлених завдань.

Розроблені ігрові механіки були спроектовані, щоб дати гравцеві можливість переміщатися і досконально досліджувати ігровий світ, що є набором з різних ігрових локацій, для переміщення з однієї в іншу іноді необхідно виконати ряд умов гри, наприклад пошук певного предмета або перемога над противником. Були реалізовані класичні механіки переміщення для ігр жанру Metroidvania, такі як біг та стрибки, а також механіки бою, такі як атака мечем та використання карток магії. Діалогова система була реалізована з метою надання користувачеві можливості отримувати інформацію про ігровий світ, сюжет та персонажах.

Після реалізації цілей розробки було проведено досконале тестування всіх механік для знаходження та усунення дефектів. Як подальші цілі можна виділити необхідність подальшої розробки нових і вдосконалення вже реалізованих механік у разі виявлення недоліків.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Смит Й., Эгенфельдт-Нильсен С., Тоска С. Understanding Video Games: The Essential Introduction. Taylor & Francis, 2019, 400 с.

2. Крісс А. The Gaming Mind. A New Psychology of Videogames and the Power of Play. Hachette, 2021, 511 с.

3. Харрис Д., Алекс Вілтшир А. Making Videogames. The Art of Creating Digital Worlds. Thames and Hudson, 2022, 352 с.

4. Шелл Дж. The Art of Game Design: A Book of Lenses, Third Edition. A K Peters/CRC Press, 2020, 652 с.

5. Харченко О. Г., Яцишин В. В., Райчев І.Е. Інструментальний засіб розробки та комунікації вимог якості до програмних систем. Науковий журнал «Інженерія програмного забезпечення». 2010. №2. С. 29-34.

6. Османі Е. Learning JavaScript Design Patterns: A JavaScript and React Developer's Guide 2nd Edition. O'Reilly Media, 2023, 296 с.

7. Фланаган Д. JavaScript. The Definitive Guide. Master the World's Most-Used Programming Language 7th Editio. O'Reilly, 2020, 706 с.

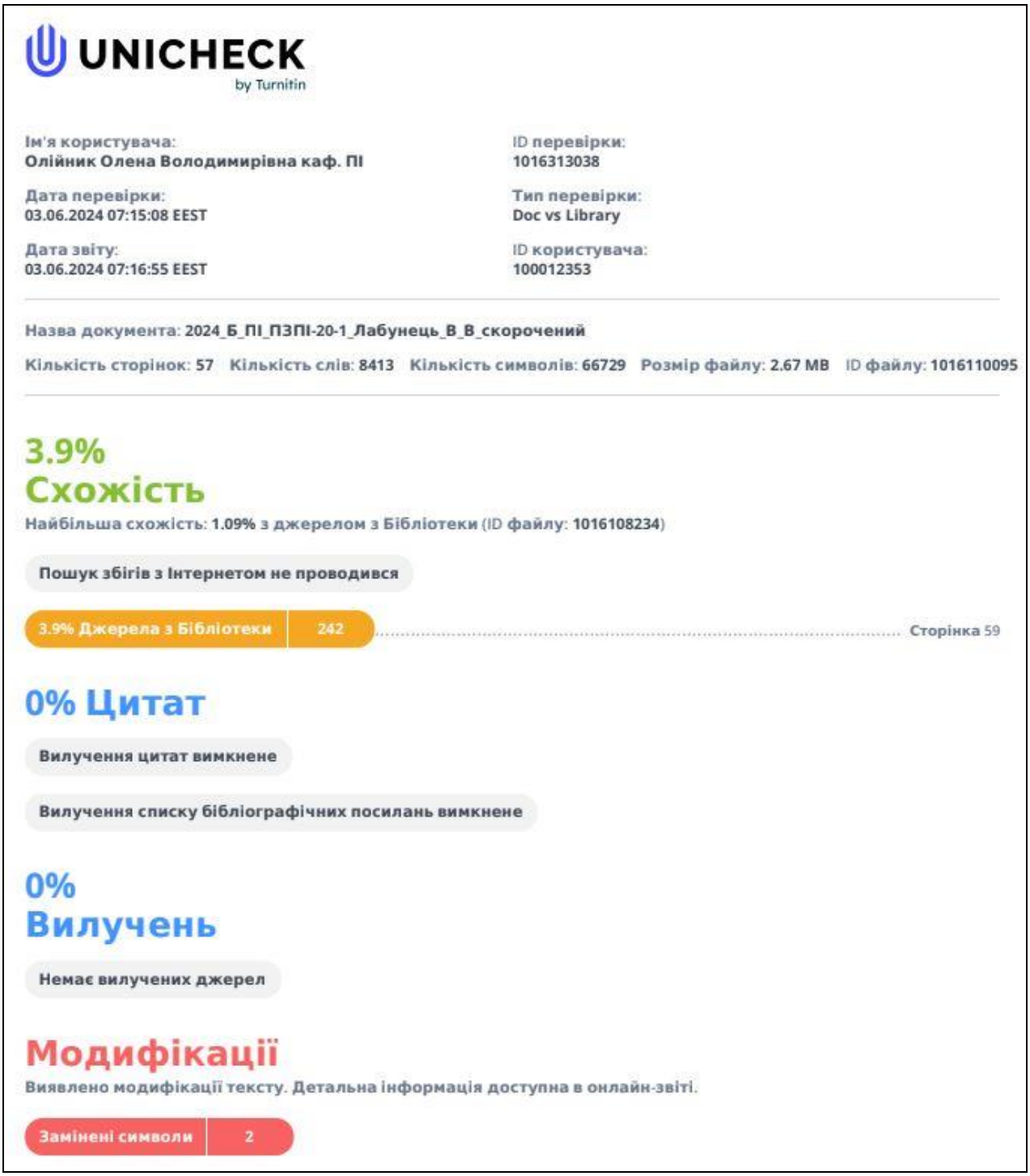
8. Офіційна сторінка середовища розробки ct.js [Електронний ресурс] - URL: https://ctjs.rocks/

9. Сон К., Павлеас Дж. Build Your Own 2D Game Engine and Create Great Web Games. 2nd Ed. Apress, 2022, 742 с.

10. Адамс Т. Procedural Storytelling in Game Design. Taylor & Francis, 2019, 584 с.

ДОДАТОК А

Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ



UNICHECK
by Turnitin

Ім'я користувача:
Олійник Олена Володимирівна каф. ПІ

ID перевірки:
1016313038

Дата перевірки:
03.06.2024 07:15:08 EEST

Тип перевірки:
Doc vs Library

Дата звіту:
03.06.2024 07:16:55 EEST

ID користувача:
100012353

Назва документа: 2024_Б_ПІ_ПЗПІ-20-1_Лабунець_В_В_скорочений

Кількість сторінок: 57 Кількість слів: 8413 Кількість символів: 66729 Розмір файлу: 2.67 MB ID файлу: 1016110095

3.9%
Схожість

Найбільша схожість: 1.09% з джерелом з Бібліотеки (ID файлу: 1016108234)

Пошук збігів з Інтернетом не проводився

3.9% Джерела з Бібліотеки 242 ..... Сторінка 59

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

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

Замінені символи 2

ДОДАТОК Б

Слайди презентації

Ігровий програмний застосунок у жанрі Metroidvania з елементами карткової гри. Ігрові механіки. Кодування. Діалогова система

Виконав: ст. гр. ПЗПІ-20-1 Лабунець Владислав
Керівник: ст.викл. кафедри ПІ Саманцов О.О.

Що таке Metroidvania?

- Вільна структура світу
- Системи прогресії
- Дослідження та збирання
- Бойова система
- Стиль та атмосфера

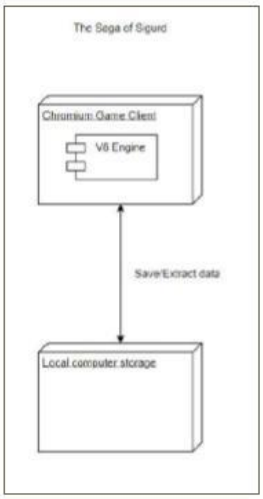
Середа розробки

Платформинг та переміщення

Збереження даних

```
var saveHero = () => {
    let heroTemplate = ct.templates.list['Character1'][0];
    let heroData = ct.content.Hero[0]
    heroData.health = heroTemplate.health;
    heroData.mana = heroTemplate.mana;
    localStorage.setItem('heroData', JSON.stringify(heroData));
}

var loadHero = () => {
    let heroTemplate = ct.templates.list['Character1'][0];
    try {
        let heroDataJson = JSON.parse(localStorage.getItem('heroData'))
        let heroData = ct.content.Hero[0];
        heroData.health = heroDataJson.health;
        heroData.mana = heroDataJson.mana;
        heroTemplate.health = heroData.health;
        heroTemplate.mana = heroData.mana;
```



Механіки бою. Ближній бій

Механіки бою. Магічні картки

Картки

Перша картка

Ворожі істоти

Мара

Волкудлак

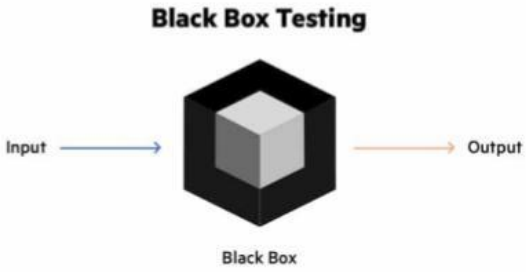
Ирка

Діалогова система

- Інформація про світ
- Завдання
- Сюжет
- Міні-ігри

Про тестування

- Функціональне тестування
- Тестування інсталяції
- Тестування відмовостійкості та відновлення
- Тестування продуктивності
- Тестування інтерфейсу користувача
- Тестування конфігурації



Дякую за увагу!

ДОДАТОК В

Специфікація ПЗ

1. ВСТУП

1. Огляд продукту

Продукт представляє з себе відеогру в жанрі Metroidvania, яка ґрунтується на слов’янській міфології і естетиці. Ігрові події відбуваються у Київській Русі за часів Володимира Великого. Головний герой – воїн, який був добре знайомий із князем.

Геймплей гри відбувається в 2D світі, розділеному на окремі локації. Гравець може вільно пересуватися світом, шукати магічні картки, щоб використовувати їх у бою. Для переміщення у окремій локації гравець може використовувати платформи, щоб можна спускатися вниз чи підніматися на них вгору.

Бойова система є класичною для жанру Metroidvania: герой може використовувати меч у ближньому бою або магічні карти, витрачаючи ману. В грі є діалоги, які просувають сюжет та інформують гравця про ігровий світ.

2. Мета

Говорячи про мету створення продукту необхідно відзначити, що існує досить багато ігор жанру Metroidvania, але існуючі аналоги, попри їх популярність, мають свої недоліки. Тому створення нової гри в цьому жанрі виправдане через кілька ключових причин. По-перше, нова гра може оживити жанр і зацікавити гравців, які шукають нових вражень. Це вирішує проблему втоми від старих ігор. По-друге, нові потреби гравців, як-от більша складність і інноваційні механіки, можуть бути враховані в новій грі. Свіжі ідеї й унікальні підходи можуть привернути увагу як фанатів жанру, так і нових гравців. Випуск нової гри також може відродити інтерес до жанру, який може втрачати популярність через перенасиченість або брак нових проектів.

3. Межі

Гра розробляється як автономний продукт без необхідності підключення до серверів для основного геймплею, крім випадків завантаження оновлень. Всі збереження гри зберігаються локально на пристрої гравця. Немає підтримки хмарних збережень.

Пз розробляється тільки під наступні операційні системи та платформи: Windows, macOS, Linux, Android, iOS. Кожна з платформ має відповідати власним мінімальним системним властивостям.

Команда розробників складається з 3 осіб, що відповідають за проєктування, дизайн, написання коду та тестування ПЗ.

Проект має бути завершений до 12.06.2024.

4. Посилання

https://docs.ctjs.rocks/ct-concepts.html – офіційна документація до ігрового двигуна ct.js.

https://en.m.wikipedia.org/wiki/Metroidvania – історія та опис жанру Metroidvania.

https://uk.esotericsoftware.com/ - офіційна сторінка програмного застосунку для роботи з анімацією “Spine”

https://krita.org/ - офіційна сторінка програмного застосунку для створення графіки «Krita»

5. Означення та абревіатури

JS – JavaScript

TS – TypeScript

HTML – HyperText Markup Language

CSS – Cascading Style Sheets

CG – Computer Graphics

2. ЗАГАЛЬНИЙ ОПИС

1. Перспективи продукту

В першу чергу варто сказати, що хоча даний програмний продукт є закінченим, його розвиток та підтримка мають бути навіть після публікації та випуску першої версії гри. Як вже було зазначено, з точки зору ринкових перспектив програмного продукту, основними можна назвати такі як вихід на міжнародний ринок та публікацію на усіх основний та найпопулярніших ігрових майданчиках, таких як Steam, Epic Games Store та інші. Окім цього, слід зазначити, що в майбутньому, при умовах успіху гри на основних цільових платформах, у перспективі можливо створити окрему версію гри для найпопулярніших ігрових консолей, таких як PlayStation та Xbox.

2. Функції продукту

При ввімкненні гри головними опціями для гравця є налаштування ігрового процесу, продовження гри у тому місці де він закінчив минулого разу, початок нової гри та вихід із гри.

В свою чергу, при переході в режим самої гри, користувач має досить багато різного функціоналу на вибір. Так, гравець може досліджувати світ гри, дізнаватися нові деталі ігрового сюжету та спілкуватися із неігровими персонажами. Окрім цього, гравець може шукати картки та артефакти, які можуть полегшити ігровий процес та покращити здібності головного героя.

Також важливою складовою є бойова система. Так, гравець також може або тренуватися на ворогах та досліджувати їх слабкі сторони, або проходити сюжет гри, яким вимагає зіткнення із ворогами. Важливою частиною бойової системи є те, що вона дозволяє герою комбінувати фізичні атаки із магічною системою.

3. Характеристики користувачів

Як вже було зазначено, основною цільовою аудиторією гри є чоловіки віком від 18 років. Така категорія була обрана з урахуванням того, що така цільова аудиторія є найбільш зацікавленою у подібних іграх, а також, окрім цього, є

найбільш платіжоздатною, що має позитивно вплинути на монетизацію та комерційний успіх нашого програмного продукту.

4. Загальні обмеження

– Для розробки гри має використовуватися ігровий двигун Ct.js
– Гра має працювати на Windows, Unix, MacOS, Android, IOS та Web
– Гра повинна мати мінімум 30 кадрів на секунду
– Гра має бути локалізована англійською мовою

5. Припущення і залежності

– Користувачі заздалегідь зацікавлені у програмному продукту
– Ігрові платформи користувачів оновлено до останньої версії
– Гра має визначений набір функціоналу, який має бути реалізовано для першої версії
– Команда розробників має ліцензії на програмне забезпечення, що використовується

3. КОНКРЕТНІ ВИМОГИ

1. Вимоги до зовнішніх інтерфейсів

1. Інтерфейс користувача

UI у ігровому програмному застосунку поділяється на 2D рівні та UIX складові. User Experience виключно представлений у головному меню, де гравець за допомогою кнопок може виконувати навігацію у грі, а саме почати нову гру, завантажити поточку та вийти на робочий екран (див. рис. 3.1).

Рисунок 3.1 - Головне меню гри

Уся інша навігація у ігровому програмному застосунку виконується за допомогою клавіатури та комп’ютерної миші.

Застосунок має декілька рівнів, де гравець,керуючи головним героєм, може взаємодіяти з об’єктами, ворожими та нейтральними персонажами.

Перший рівень представлений у вигляді нижнього світу, де користувач проходить навчання та веде внутрішній діалог із самим собою (див. рис. 3.2).

Рисунок 3.2 - Діалог у грі

Головний герой має показники здоров’я та магічних здібностей, які супроводжують його протягом усієї гри. Також на першому рівні він отримує колекційну карту, яку може використовувати під час бою з супротивником(див. рис. 3.3).

Рисунок 3.3 - Навчальна локація гри

У ігровому програмному застосунку наявна локація внутрішнього приміщення будинку головного героя, де розташовані побутові речі та його дружина (див. рис. 3.4).

Рисунок 3.4 - Будинок головного героя

Наступною локацією для переходу є селище, де розташовані будинки NPC (див. рис. 3.5).

Рисунок 3.5 - Локація селища

Користувач може відвідати рівень з мостом над річкою, яка є способом переміщення героя до верхнього або нижнього світу (див. рис. 3.6).

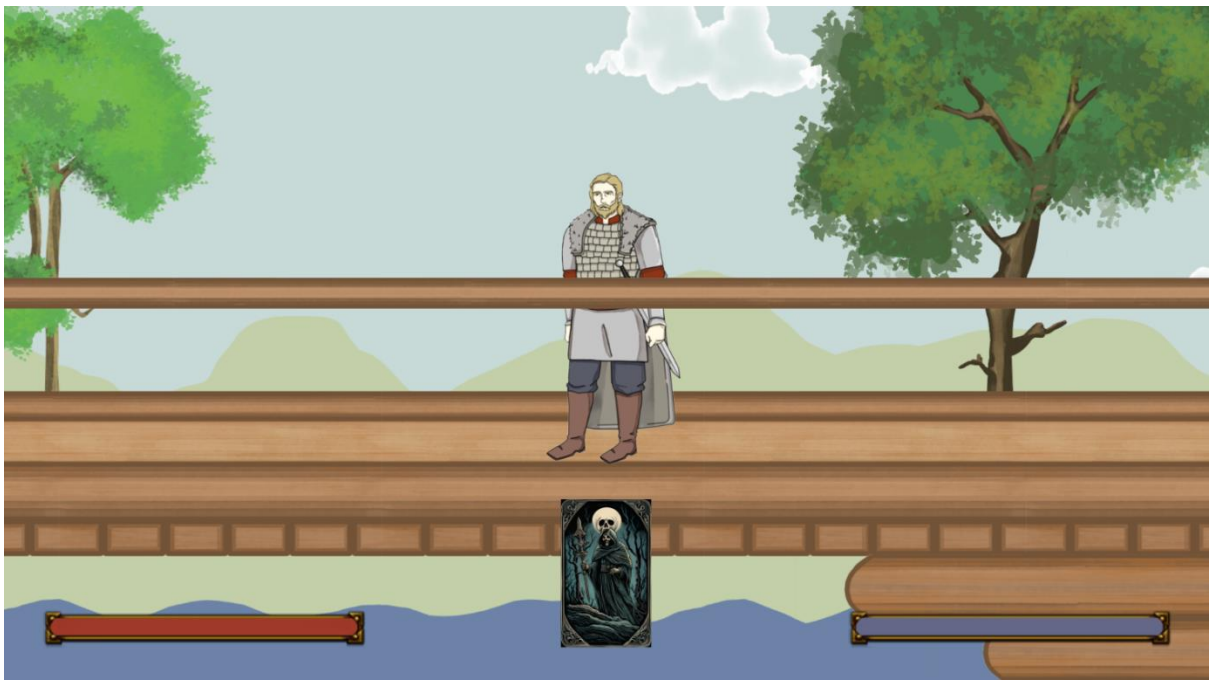


Рисунок 3.6 - Локація з мостом

У локації з помешканням князя Володимира, гравець може почати взаємодію з ним (див. рис. 3.7).

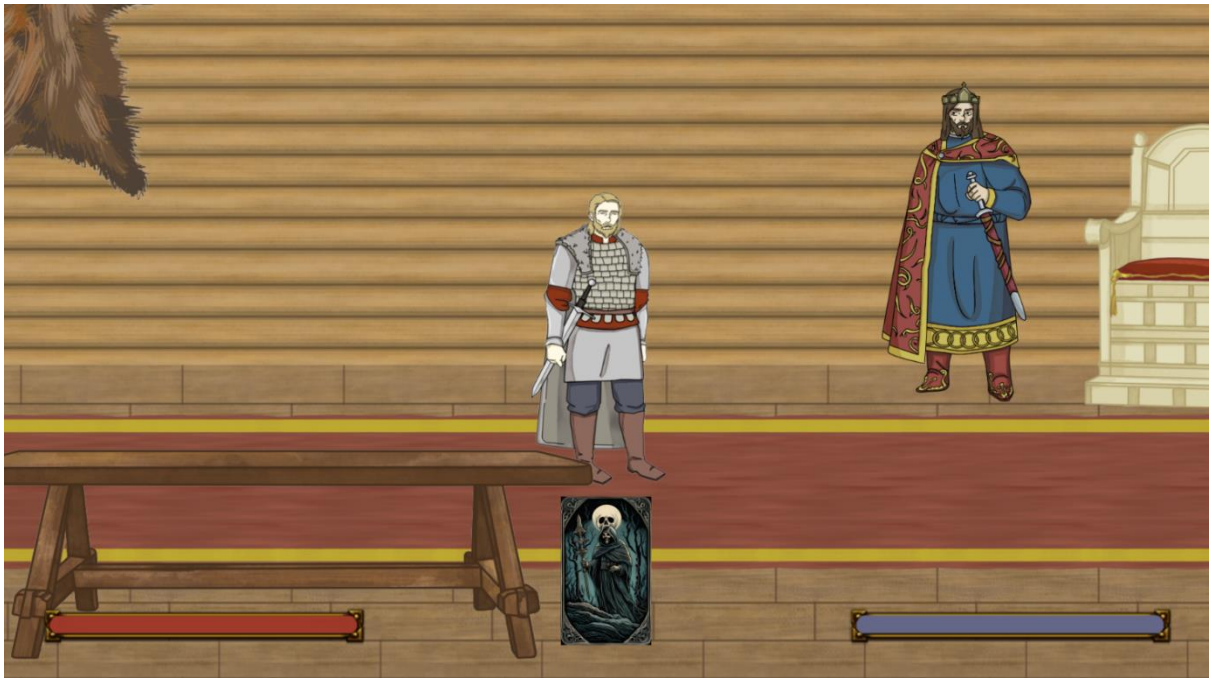


Рисунок 3.7 - Локація з помешканням князя

З лівого боку від селища, користувач може відвідати локацію з капищем та зустріти волхва (див. рис. 3.8).

Рисунок 3.8 - Локація з капищем

Для того щоб відвідати князя Володимира, гравцю необхідно пройти рівень з лісом, де його очікують такі ворожі моби, як Волкодлак та Ирка (див. рис. 3.9).

Рисунок 3.9 - Локація з лісом

З великим шансом, користувачу ігрового програмного застосунка необхідно буде вступити у бій, щоб здолати супротивників зі своїми унікальними механіками. Ця локація найкраще демонструє бойову систему гри.

2. Комунікаційний протокол

Компоненти ігрового програмного застосунку певним чином взаємодіють один з одним, що демонструє їх комунікацію. Програму можна поділити на два основних середовища: сховище ігрових файлів та файл програми ct.js (див. рис. 3.10).

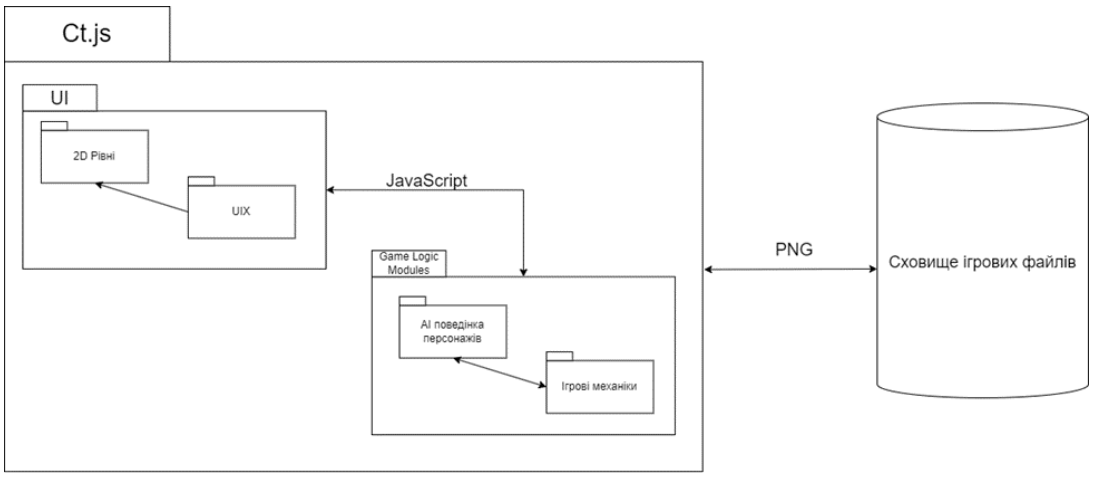


Рисунок 3.10 - Діаграма, яка демонструє комунікаційний протокол

Між собою вони взаємодіють за допомогою файлу з розширенням .png. Можна також розділити основну програму на два підрозділи: UI та Модуль з ігровою логікою. Вони мають окремий метод комунікації, а саме за допомогою мови програмування JavaScript.

3. Обмеження пам’яті

Були визначені мінімальні обмеження пам’яті під час завантаження та запуску гри:

– Оперативна пам’ять: 512 MB;
– Місце на диску: 1 GB.

4. Операції

Операції у програмному ігровому застосунку можна поділити на три види: взаємодія з UIX, введення даних з апаратного пристрою, взаємодія з оточенням. Під час взаємодії з User Experience гравець може натискати на кнопки представлені у головному меню. Коли користувач вводить певні клавіші на клавіатуру та на комп’ютерній мишці, він отримує певні зміни у інтерфейсі. Головний герой може взаємодіяти з ігровим світом, на що програма буде відповідати.

4.1 Початок нової гри

4.1.1 Представлення

Ігровий програмний застосунок повинен надавати можливість почати проходити нову гру.

4.1.2 Вхідні дані

Користувач натискає кнопку “New Game” у головному меню.

4.1.3 Обробка

Пошук рівня, з якого починається нова гра.

4.1.4 Результат

Відображення першого рівня гри.

4.2 Завантаження збереження

4.1.1 Представлення

Ігровий програмний застосунок повинен надавати можливість завантажувати рівень, на якому востаннє був гравець.

4.1.2 Вхідні дані

Користувач натискає кнопку “Load Game” у головному меню.

4.1.3 Обробка

Пошук рівня, на якому користувач був останній раз.

4.1.4 Результат

Відображення рівня та усіх даних, які мав гравець в останній раз під час проходження.

4.3 Вихід з ігрового застосунку

4.1.1 Представлення

Ігровий програмний застосунок повинен надавати можливість вийти з нього до робочого екрану.

4.1.2 Вхідні дані

Користувач натискає кнопку “Exit” у головному меню.

4.1.3 Обробка

Завершення роботи програми.

4.1.4 Результат

Користувач вийшов до власного робочого стола пристрою з повним закриттям ігрового програмного застосунку.

4.4 Рух персонажем вліво

4.1.1 Представлення

Ігровий програмний застосунок повинен надавати можливість керувати головним героєм переміщуючи його вліво.

4.1.2 Вхідні дані

Користувач натискає клавішу“A” на клавіатурі.

4.1.3 Обробка

Перевірка введених даних з клавіатури.

4.1.4 Результат

Головний герой переміщується вліво до повного відпускання певної клавіші.

4.5 Рух персонажем направо

4.1.1 Представлення

Ігровий програмний застосунок повинен надавати можливість керувати головним героєм переміщуючи його вправо.

4.1.2 Вхідні дані

Користувач натискає клавішу“D” на клавіатурі.

4.1.3 Обробка

Перевірка введених даних з клавіатури.

4.1.4 Результат

Головний герой переміщується вправо до повного відпускання певної клавіші.

4.6 Стрибок персонажа

4.1.1 Представлення

Ігровий програмний застосунок повинен надавати можливість керувати головним героєм переміщуючи його догори.

4.1.2 Вхідні дані

Користувач натискає клавішу“W” або пробіл на клавіатурі.

4.1.3 Обробка

Перевірка введених даних з клавіатури.

4.1.4 Результат

Головний герой переміщується вліво до завершення анімації стрибка.

4.7 Удар зброєю

4.1.1 Представлення

Ігровий програмний застосунок повинен надавати можливість керувати головним героєм завдаючи удар зброєю.

4.1.2 Вхідні дані

Користувач натискає ЛКМ.

4.1.3 Обробка

Перевірка введених даних з комп’ютерної миші.

4.1.4 Результат

Головний герой наносить удар противнику.

4.8 Активація магії

4.1.1 Представлення

Ігровий програмний застосунок повинен надавати можливість керувати головним героєм використовуючи магічні здібності з колекційних карток.

4.1.2 Вхідні дані

Користувач натискає клавішу“1” на клавіатурі.

4.1.3 Обробка

Перевірка введених даних з клавіатури.

4.1.4 Результат

Головний герой відтворює магію з корекційної картки відповідно до її механік.

4.9 Закінчення гри

4.1.1 Представлення

Ігровий програмний застосунок повинен надавати можливість завершувати гру шляхом смерті головного героя.

4.1.2 Вхідні дані

Головний герой отримує урон, що змушує показник його здоров’я дорівнювати нулю.

4.1.3 Обробка

Перевірка показника здоров’я головного героя.

4.1.4 Результат

Перенесення гравця до головного меню.

4.10 Взаємодія з NPC або об’єктами

4.1.1 Представлення

Ігровий програмний застосунок повинен надавати можливість взаємодіяти з NPC або об’єктами за допомогою системи діалогу або натиснення клавіші на клавіатурі.

4.1.2 Вхідні дані

Користувач натискає клавішу“E” на клавіатурі.

4.1.3 Обробка

Перевірка чи відповідає об’єкт у грі взаємодії із застосуванням відповідної клавіші.

4.1.4 Результат

Відтворення діалогу або переміщення між локаціями.

5. Функції продукту

Ігровий програмний застосунок містить основних функцій для коректної та правильної роботи додатка:

5.1 Функція відтворення діалогу

Дана функція відтворюється під час активації діалогу з певними NPC або на початку рівнів. Вона відтворює репліки до поки не спрацьовує тригер отримання квесту (див. рис. 3. 10).

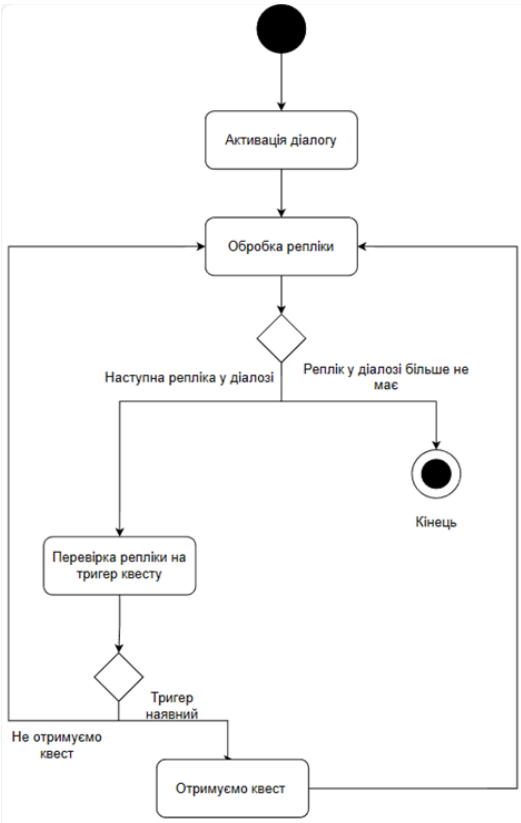


Рисунок 3.10 - Діаграма, яка демонструє діалогову функцію

5.2 Функція переміщення головного персонажа

Вона реагує на введення даних з клавіатури та комп’ютерної миші. Відповідно до дії користувача система перевіряє даний запит та реагує на нього певними змінами у інтерфейсі користувача.

5.3 Функція побудови бойової системи

Дана функція містить у собі логіку, за допомогою якою відбуваються дії персонажів у бою. Для кожного окремого NPC механіки можуть бути різними в залежності від його характеристик.

5.4 Функція взаємодії з об’єктами

У певні моменти часу ігровий програмний застосунок пропонує гравцю певну взаємодію з нею, після чого процедура завершується. Одним з можливих

варіантів завершення є збереження гри та повернення до головного меню (див. рис. 3.11).

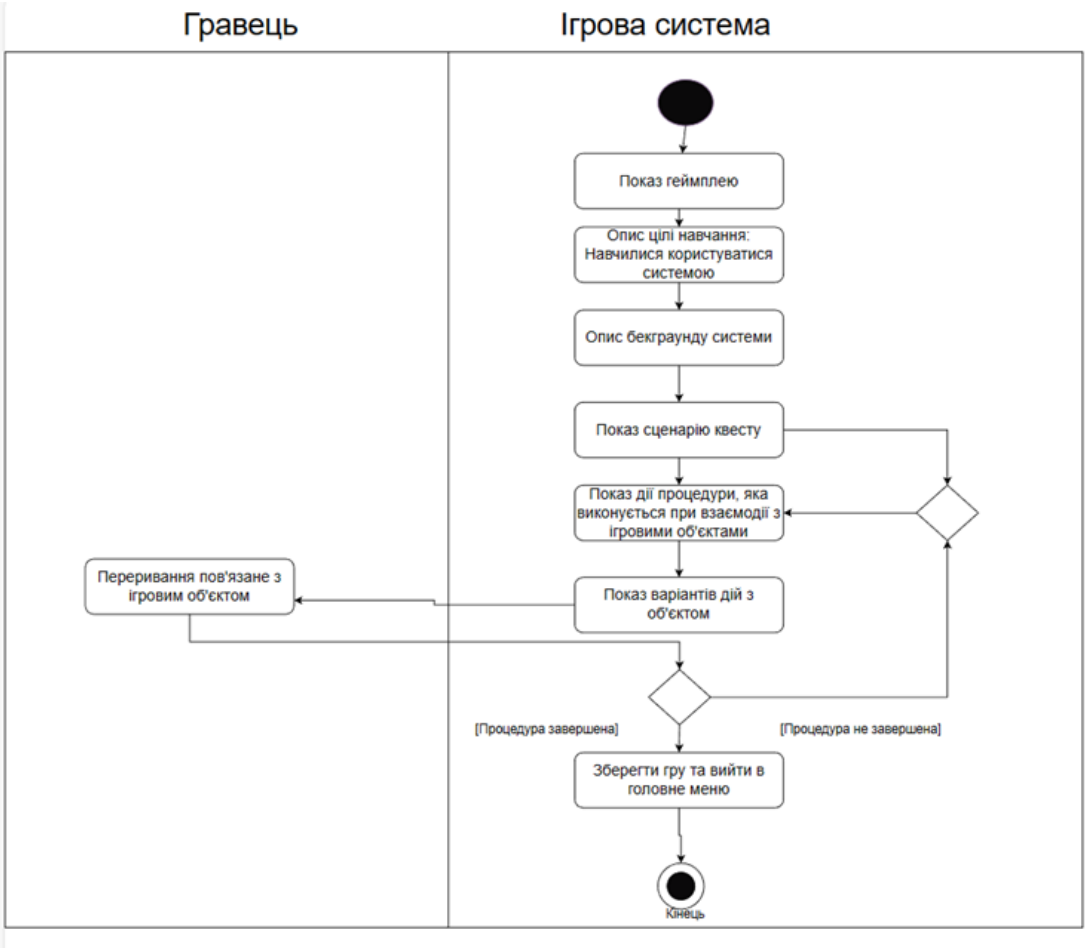


Рисунок 3.11 - Діаграма, яка демонструє взаємодію гравця із системою

6. Припущення й залежності

До припущень та залежностей можна віднести такі ситуації:

Користувачі будуть використовувати гру на персональному комп’ютері, мобільному пристрої або у Web браузері.

Користувачі будуть керувати головним героєм шляхом натискання відповідних клавіш на клавіатурі.

У певний момент часу користувачі захочуть вийти з гри та перейти до свого робочого екрану.

Користувачам буде необхідне графічне відображення кожної дії, яку вони зробили.

Користувачам буде необхідно завантажити збереження проходження гри після виходу з неї або смерті головного героя.

2. Властивості програмного продукту

Розробляємий продукт є ігровим застосунком в жанрі Metroidvania з глибоким оригінальним сюжетом, гарним стилем, захоплюючою бойовою системою та атмосферним музичним супроводом.

У грі герой може переміщатися по горизонталі за допомогою бігу та по вертикалі стрибаючи по платформах. Також є можливість переміщатись між локаціями, взаємодіяти з предметами та персонажами. Взаємодія з персонажами представляє собою діалог, що надасть гравцю інформація про світ гри та вкажіть на сюжетні завдання. У бою з ворожими істотами є можливість атакувати у ближньому бою мечем чи у використовувати магію знайдених карток.

Застосунок має гарний та зрозумілий дизайн, що впливає на відчуття ігрового досвіду користувачів. Дизайн застосунку допомагає гравцю зорієнтуватися та зрозуміти всі функції представлені у грі, що прямо вплине на рівень їх задоволення. Крім цього дизайн дає можливість поглибитися в ігровий світ та повноцінно відчути його атмосферу.

Продукт повинен маэ можливість бути запущений на платформах Windows, macOS, Linux, Android та iOS. Ігровий програмний застосунок ефективно витрачає системні ресурси на різних пристроях.

3. Атрибути програмного продукту

3.1. Надійність

ПЗ, що розробляється, повинна мати автоматичне збереження ігрового прогресу, щоб мінімізувати втрати даних при збої, при переході на новий рівень. Після перезапуску гри користувач повинен мати можливість продовжити з останньої точки збереження.

Гра повинна фіксувати всі критичні помилки в лог-файлі для подальшого аналізу та інформувати гравців про помилки у інтерфейсі. Також додаток повинен підтримувати можливість моніторингу використання ресурсів у реальному часі.

3.2 Доступність

Кажучи про доступність, як було зазначено раніше, даний ігровий програмний продукт відповідає основним принципам еврістики, що робить його доступним для нових користувачів завдяки зрозумілому та простому програмному інтерфейсу.

Так, інтерфейс нашого додатку чітко дозволяє зрозуміти що він очікує від гравця та дозволяє йому легко взаємодіяти з програмною системою. Окрім цього, як було зазначено вище, в перспективі гра буде локалізована на більшість найпоширеніших мов, що зробить гру доступнішою для кінцевих користувачів із різних країн.

Також дуже важливим елементом доступності є навчання. Так, коли гравець починає нову гру, його зустрічає навчальний рівень, який проводить гравця та навчає його основних ігрових механік, а також показує як керувати персонажем та взаємодіяти з навколишнім середовищем ігрового світу. Це значно знижує криву навчання і дозволяє гравцеві швидше включитися у ігровий процес.

3.3 Безпека

Стосовно безпеки, у нашій грі імплементовано основні принципи, які захищають кінцевих користувачів. Так, наша гра не збирає персональну інформацію про користувачів та не оперує нею, що значно поліпшує загальну безпеку та у найгіршому випадку може значно знизити ризики компрементації даних гравців.

Окрім цього, наша гра розташована на популярних ігрових майданчиках та покладається на захист, який надають ці системи. Стосовно самої гри, оскільки значна її частина працює на мові програмування JavaScript та платформі Electron, то нашу гру неможливо зворотно зібрати, оскільки гра не запускається нативно, а працює в окремому захищеному середовищі, а також не взаємодіє із файловою

системою та операційною системою платформи, на якій гру було запущено. Такий підхід дозволяє захистити гру, в тому числі, від модифікацій і чит кодів, які могли би зруйнувати ігровий процес для гравця.

3.4 Супроводжуваність

У розробляємому додатку має бути система відстеження помилок і пропозицій користувачів, а також організовано службу технічної підтримки з відповідними інструкціями та процедурами. Повинно бути наведеною детальна технічна документація з описом всіх модулів та інструкції для користувачів з прикладами використання. Система має регулярно оновлюватися нові версії мають проходити юніт-тести для всіх ключових модулів.

3.5 Переносимість

Переносимість ПЗ - це важливий аспект, оскільки забезпечує можливість використання розробляємого ПЗ на різних платформах, пристроях, операційних системах тощо. Ігровий застосунок має коректно працювати на операційних системах Windows, macOS, Linux, Android та iOS. А також підтримувати пристрої управління цих платформ.

3.6 Продуктивність

При використанні додатку має бути стабільний фреймрейт та висока якість графіки на різних конфігураціях ПК, прийнятний час завантаження рівнів та текстур, стабільний геймплей без великих затримок, стійка продуктивність гри під високими навантаженнями.

ДОДАТОК Г

Приклади коду програми

Збереження та завантаження даних:

```
var saveHero = () => {
    let heroTemplate = ct.templates.list['Character1'][0];
    let heroData = ct.content.Hero[0]
    heroData.health = heroTemplate.health;
    heroData.mana = heroTemplate.mana;
    localStorage.setItem('heroData', JSON.stringify(heroData));
}

var loadHero = () => {
    let heroTemplate = ct.templates.list['Character1'][0];
    try {
        let heroDataJson =
JSON.parse(localStorage.getItem('heroData'))
        let heroData = ct.content.Hero[0];
        heroData.health = heroDataJson.health;
        heroData.mana = heroDataJson.mana;
        heroTemplate.health = heroData.health;
        heroTemplate.mana = heroData.mana;
    }
    catch {
        heroTemplate.health = 100;
        heroTemplate.mana = 100;
    }
}
```

Функції діалогів:

```
var initializeDialogue = (startId) => {
    let dialogue = findDialogue(startId);
    if (dialogue.isOneTime && dialogue.isPlayed)
        return;
    let hero = ct.templates.list['Character1'][0];
    hero.dialogueMode = true;
    StopGame();

    let room = ct.rooms.append('Dialogues');
    createText(room, dialogue.Name, 'Beresta', 100, 810, 1);
    let textArr = splitString(dialogue.Text, 70);
    for(let i = 0; i < textArr.length; i++) {
        createText(room, textArr[i], 'Beresta', 100, 890 + i*52,
0.75);
    }

    let x = 1710;
    if (dialogue.isLeft)
        x = 210;
    let sprite = ct.templates.copyIntoRoom("DialogueHero", x, 900,
room);

    sprite.tex = dialogue.Sprite;

    sprite.scale.x = 2;
```

```
            sprite.scale.y = 2;
            if (dialogue.isSpriteReversed == true)
                sprite.scale.x *= -1;
            sprite.depth = -2;

            localStorage.setItem('currentDialogue', JSON.stringify(dialogue))
            dialogue.isPlayed = true
        }

        var findDialogue = (dialogueId) => {
            let dialogue = ct.content.Dialogues.find(dialogue => dialogue.Id
== dialogueId);
            return dialogue;
        }

        var nextDialogue = () => {
            let currentDialogue =
JSON.parse(localStorage.getItem('currentDialogue'));
            currentDialogue.isPlayed = true;
            let nextDialogue = findDialogue(currentDialogue.nextId);
            ct.rooms.list['Dialogues'].forEach(r => r.destroy());
            if (nextDialogue && nextDialogue.Id != -1) {
                initializeDialogue(nextDialogue.Id);
            }
            else {
                let hero = ct.templates.list['Character1'][0];
                hero.dialogueMode = false;
                StopGame();
            }
        }

        var clearPlayedDialogues = () => {
            let dialogues = ct.content.Dialogues;
            dialogues.forEach(d => d.isPlayed = false)
            console.log(dialogues);
        }

        var createText = (room, text, style, x, y, scale) => {
                let gameText = new PIXI.Text(text, ct.styles.get(style));
                gameText.x = x;
                gameText.y = y;
                gameText.scale.x = scale;
                gameText.scale.y = scale;
                gameText.zIndex = 1000;
                room.addChild(gameText);
        }

        var splitString = (str, len) => {
            var result = [];
            var words = str.split(" ");
            var currentLine = "";

            words.forEach(function(word) {
                if ((currentLine + word).length <= len) {
                    currentLine += (currentLine === "" ? "" : " ") + word;
                } else {
                    result.push(currentLine);
                    currentLine = word;
```

```
        }
    });

    if (currentLine !== "") {
        result.push(currentLine);
    }

    return result;
}
```

Зупинення гри:

```
var StopGame = () => {
    if (!ct.room.paused) {
        ct.room.paused = true;
        ct.pixiApp.ticker.speed = 0;
        let hero = ct.templates.list['Character1'][0];
        hero.stop();
    } else {
        ct.room.paused = false;
        ct.pixiApp.ticker.speed = 1;
        let hero = ct.templates.list['Character1'][0];
        hero.play()
    }
}
```

Властивості при створенні головного героя:

```
this.health = 100
this.mana = 100;
this.maxSpeed = 10;
this.jumpSpeed = 30;
this.gravity = 1;
this.gravityDir = 90;
this.accDec = 1.5;
this.interactionPoints = 0;

this.isStuck = false;
this.dialogueMode = false;
this.attackMode = false;
this.magicMode = false;
this.jumpMode = false;
this.doubleJumped = false;
this.isAnimationReversed = false;
this.loop = true;

// Assume that the starting coordinates are a safe spot to respawn
this.checkpointX = this.x;
this.checkpointY = this.y;
ct.camera.follow = this;
```

Дії головного героя:

```
// Pause
if (ct.actions.Pause.pressed) {
    StopGame();
    return;
}
```

```
    //Dialogue or stuck
    if(this.dialogueMode || this.isStuck) {
        return;
    }

    // Get the surrounding medium
    var collisions = this.moveContinuousByAxes('Solid');
    this.onGround = ct.place.occupied(this, this.x, this.y + 2, 'Solid');

    //Character movement
    var targetHspeed = ct.actions.MoveX.value * this.maxSpeed;
    if(this.attackMode || this.magicMode) {
        targetHspeed = 0;
    }
    if (this.hspeed > targetHspeed) {
        this.hspeed = Math.max(this.hspeed - this.accDec * ct.delta,
targetHspeed);
    } else if (this.hspeed < targetHspeed) {
        this.hspeed = Math.min(this.hspeed + this.accDec * ct.delta,
targetHspeed);
    }

    //Character direction
    if (this.hspeed > 0.1) {
        this.scale.x = 1;
    } else if (this.hspeed < -0.1) {
        this.scale.x = -1;
    }

    //Attack, magic, jump actions
    if(!this.attackMode && !this.magicMode) {
        if(ct.actions.Attack.pressed) {
            this.attackMode = true;
            this.magicMode = false;
            this.jumpMode = false;
            ct.sound.spawn('Sword')
        }
        else if(ct.actions.Card1.pressed || ct.actions.Card2.pressed ||
ct.actions.Card3.pressed) {
            this.attackMode = false;
            this.magicMode = true;
            this.jumpMode = false;
        }
        else if (ct.actions.Jump.pressed) {
            if(this.onGround) {
                this.jumpMode = true;
            }
            else if (!this.doubleJumped) {
                this.vspeed = -this.jumpSpeed;
                this.doubleJumped = true;
            }
        }
    }
```