Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(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. Звукові ефекти. Ігрові механіки. Гейм-дизайн. Міні-ігри
(тема)

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

Касперчук О.О.
(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. викладач 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 Касперчуку Олександру Олеговичу
(прізвище, ім’я, по батькові)

1. Тема роботи Ігровий програмний застосунок в жанрі Metroidvania з елементами карткової гри. Звукові ефекти. Ігрові механіки. Гейм-дизайн. Міні-ігри.
Затверджена наказом по університету від 20.05. 2024р. № 471 Ст
2. Термін подання студентом роботи до екзаменаційної комісії 14.05.2024
3. Вихідні дані до роботи Розробити ігровий застосунок в жанрі Metroidvania, з елементами карткової гри на мові програмування JavaScript використовуючи ігровий двигун 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.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	09.04.2024	*виконано*
2	Створення специфікації ПЗ	10.04.2024	*виконано*
3	Проектування ПЗ	13.04.2024	*виконано*
4	Розробка ПЗ	29.04.2024	*виконано*
5	Тестування ПЗ	10.05.2024	*виконано*
6	Оформлення пояснювальної записки	21.05.2024	*виконано*
7	Підготовка презентації та доповіді	29.05.2024	*виконано*
8	Попередній захист	07.06.2024	*виконано*
9	Нормоконтроль, рецензування	09.06.2024	*виконано*
10	Здача роботи у електронний архів	10.06.2024	*виконано*
11	Допуск до захисту у зав. кафедри	12.06.2024	*виконано*

Дата видачі завдання 08 квітня 2024р.

Студент (ка)_____ Касперчук О.О.
(підпис)

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

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до кваліфікаційної роботи бакалавра, 41 стор., 10 рис., 4 джерела.

METROIDVANIA, ІГРОВИЙ ПРОГРАМНИЙ ЗАСТОСУНОК, КАРТКОВА ГРА, CT.JS, JAVASCRIPT

Робота описує процес проектування та обраний спосіб розробки для ігровий програмний застосунок в жанрі metroidvania з елементами карткової гри.

Об’єкт розробки – ігровий програмний застосунок в жанрі metroidvania з елементами карткової гри.

Мета розробки – створення ігрового програмного застосунку, який поєднуватиме в собі жанр metroidvania із елементами карткової гри.

Метод рішення – середовище розробки Ct.js та власний редактор цього движка, мова програмування JavaScript.

У результаті розробки створено ігровий програмний застосунок, котрий поєднує в собі жанр metroidvania із елементами карткової гри у вигляді збору карток на рівнях та можливості грати в карткову гру.

METROIDVANIA, GAME SOFTWARE, CARD GAME, CT.JS, JAVASCRIPT

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

The purpose of the work is to create gaming software application that will combine the genre of metroidvania with the elements of card game.

Solution method – Ct.js game engine and its own code editor, JavaScript programming language.

As a result of the development, a game software application was created, that combines the genre of metroidvania with the elements of card game in a way that player collects cards on a level and may play a card game with them.

Я, Касперчук Олександр Олегович, студент гр. ПЗПІ-20-1, здобувач вищої освіти на першому (бакалаврському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Ігровий програмний застосунок в жанрі Metroidvania», 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

Темою кваліфікаційної роботи є ігровий застосунок в жанрі metroidvania із елементами карткової гри під назвою The Saga of Sigurd. Основне завдання ігрового програмного застосунку – надання гравцеві можливість поринути в світ гри із тематикою міфологічної Київської Русі та отримати насолоду від ігрового процесу.

Метою роботи є розробка програмної системи, що складається із веб-версії, застосунку для персональних комп’ютерів та мобільного додатку. Кожна із версій містить повну версію гри із певною адаптацією системи керування під конкретну ігрову платформу. Для розробки продукту використовувався ігровий движок Ct.js та мова програмування JavaScript. В якості середовища розробки використано вбудований в ігровий движок Ct.js редактор коду.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

Ігри у жанрі metroidvania стають все більше популярними у світі. Аналіз даних із відкритих джерел продемонстрував, що цей жанр ігор, разом із суміжним із ним жанром platformer, є найбільш популярним і продаваємим за весь час із моменту створення ігор.

Назва жанру metroidvania походить від двох найперший ігор цього типу: Metroid та Castlevania. Особливістю жанру полягає в наявності відносно відкритого світу, здебільшого вид зі сторони, а також отримання нових навичок головним героєм гри у процесі проходження, які дозволяють дістатися до нових локацій[1]. Для наочності наведено зображення однієї із ігор в цьому жанрі, а саме Guacamelee у мексиканській стилістиці, яка відповідає усім вищеописаним характеристикам (див. рис. 1.1).

Рисунок 1.1 – Гра у жанрі metroidvania Guacamelee (https://gameranx.com/features/id/159613/article/guacamelee-2-how-to-get-the-good-ending-5-keys-locations-guide/)

Популярність ігор даного жанру можна пояснити тим, що гравцеві дуже легко навчитися основним правилам і законам запропонованого ігрового циклу. За основними своїми механіками жанр подібний до platformer, завдяки чому

гравці із досвідом можуть легко освоїтися із новою грою. На додачу, учасник гри відчуває власний прогрес завдяки можливості покращувати головного героя та його навички, завдяки чому отримує більше позитивних емоцій, коли персонаж стає сильнішим.

Разом із тим, в цьому жанрі є прогрес складності, оскільки без отримання деяких покращень, герой не може йди далі по сюжету. Такі вимоги дозволяють вирівнювати рівень гравця та ворогів, що запобігає ситуаціям, коли персонаж гравця недостатньо підготовлений, і, таким чином, значно слабший за ворогів.

Але при цьому важливою частиною ігор у жанрі metroidvania є відносно відкритий світ із багатьма розвилками та переходами між локаціями. Завдяки можливості платформингу, тобто стрибків по платформах, що розташовані в ігровому світі, гравець може добиратися до різних локацій. Самі локації в іграх даного жанру є досить складними та побудовані багатьма рівнями меншого розміру.

Варто зазначити, що на рівнях можуть розташовуватися головоломки, які гравцеві доведеться вирішувати для подальшого проходження сценарію. Але не кожну із них можна вирішити одразу: в деяких випадках герою потрібно отримати предмет, який може знаходитися на іншому рівні чи іншій локації.

Сюжетна складова metroidvania ігор подається у вигляді заставок та діалогів. У деяких випадках, описи предметів, здібності та книжки на рівнях містять інформацію про ігровий світ, яка має доповнити та домислити в голові гравця ігровий сюжет. Гарним прикладом такої гри може стати гра Blasphemous, де частини інформації приводяться в описі предметів (див. рис. 1.2).

Рисунок 1.2 – Опис предмету у грі Blasphemous (https://blasphemous2.wiki.fextralife.com/Items)

Важливо також підкреслити особливості бойової системи, які характерні для даного виду ігор. Зазвичай, бої є динамічними і залежать від особистих навичок гравця. Основними складовими бойової системи є декілька видів атак, блокування удару ворога та ухилення. Важливим для динаміки є те, що всілякі види ударів наносять різну кількість шкоди ворогу, а деякі атаки неможливо заблокувати, в результаті чого доводиться ухилятися від них.

1.2 Виявлення та вирішення проблем

Щодня виходить дуже багато відеоігор різних жанрів, в тому числі і жанру metroidvania. Але, при цьому, ігрові видавці здебільшого випускають ігри зі старими та ставши колись у минулому успішними формулами ігрового процесу. Тобто, видавці віддають перевагу однотипним і подібними одна на одну іграм. Це призводить до того, що нові ідеї не так швидко проникають в даний жанр відеоігор, що гальмує прогрес ігрової індустрії.

Стосовно візуального оформлення та стилістики гри, важливо зазначити, що наразі не існує жодної гри в вищеназваному жанрі, яка б стосувалась якогось

періоду історії України, особливо – Київської Русі, який є недостатньо репрезентованим у відеоіграх.

Замало ігор розкривають тематику слов'янської міфології та фольклору, історичної перспективи. Більшість ігор мають певні помилки та умовності у відображенні історичних подій та контексту історії.

Говорячи про технічні проблеми metroidvania, більшість із них виходять на невеликій кількості платформ. Зазвичай це персональні комп'ютери та іноді ігрові консолі. При цьому ігнорується наявність мобільних пристроїв, хоча це дуже великий сегмент ринку[2]. Разом із мобільними версіями зарідко ігри мають веб-версію, що є значним мінусом, оскільки в такому випадку гра втрачає частину гравців, яким комфортніше грати у веб-версію відеогри.

Вищеописана проблема може бути вирішена, якщо використовувати для розробки гри такі інструменти, дозволяючи випускати гру одразу на усіх платформах, або мати інструменти, які дадуть можливість портувати її на інших популярних платформах.

У випадку якщо код гри написаний погано та коли у грі наявна сучасна графіка, можуть виникнути проблеми із оптимізацією. Це значно знизить задоволення від гри для наявних користувачів та може відштовхнути нових. Також в такій ситуації наявний ризик, що не всі користувачі зможуть запустити гру на своїй улюбленій платформі. Щоб запобігти цьому, варто писати якісний код, а також, за можливістю, перевірити ігровий движок та його рівень оптимізації фінальних версій гри.

З боку графіки, не варто створювати забагато об'єктів та необхідно своєчасно чистити пам'ять й видаляти об'єкти із ігрового світу, якщо вони більше не використовуються. Одночасно із цим, для оптимізації звукового супроводу гри, можна знизити якість звуку або, у випадку, якщо декілька доріжок звучать одночасно, звести їх до однієї, і використовувати лише єдину, щоб запобігти завеликій кількості звуків, які програються одночасно та цим навантажують систему.

Окрім того, кажучи про музику та звукове супроводження гри, великою проблемою сучасних ігор є те, що вони не приділяють достатньо уваги цій частині гри, в результаті чого музичне акомпанування не завжди буває зробленим якісно, може не достовірно і яскраво передавати атмосферу та настрій гри. Це особливо важливо у відеоіграх, що орієнтуються на сюжет, оскільки музика створює більшу частину атмосфери на цих етапах ігор. І оскільки, як було зазначено раніше, сюжетна частина нерідко буває ключовою у іграх жанру metroidvania, музика та звуки є надзвичайно важливими.

Для вирішення цієї проблеми в грі повинно бути створено якісне звукове супроводження: усі звуки мають бути достатньої якості, варто використовувати живі музичні інструменти або якісні віртуальні. З точки зору коду усі звуки мають бути технічно оптимізованими, щоб запобігти проблемам зі звуком і швидкодією у гравців.

Ще однією великою проблемою є одноманітність ігрового процесу. Так, незважаючи на прогресію персонажа, нерідко бойова система не вдосконалюється під час проходження гри та, якщо покращень для бою замало, може швидко набриднути гравцеві. Те саме стосується і невеликої кількості різних ворогів у грі. Для вирішення даної проблеми можна додати більше різноманітних ворогів та покращень для головного героя, а також ускладнити гру міні-іграми. Головною особливістю міні-ігор є те, що вони роблять ігровий процес більш різноманітним та сприяють проводженню гравцю в грі більшого часу.

Також великою проблематикою сучасних ігор є необхідність постійного підключення до інтернету. Саме по собі це не є поганим, але, при цьому, не кожен користувач має постійний доступ до швидкісного інтернету, що може погіршити ігровий досвід. Особливо ця проблема актуальна у випадках із іграми, які не використовують мережевий функціонал безпосередньо, тобто є іграми для одного гравця.

Для вирішення цієї проблеми гра має не вимагати інтернет-з’єднання та мати можливість працювати повністю автономно навіть у випадку, якщо в користувача немає з’єднання із мережею. У цьому випадку відеогра має бути

повністю адаптованою для одного гравця. Не має використовувати інтернет, якщо гравець не хоче грати у багатокористувацькому режимі разом із іншими гравцями. Окрім цього, усі медіа файли гри мають зберігатися локально, а не на віддаленому сервері, а логіка гри має зберігатися на комп'ютері гравця разом із клієнтською частиною. Такий підхід дозволить використовувати програмний продукт без жодних технічних проблем, навіть у випадку повної відсутності підключення, оскільки логіка зберігатиметься локально та не буде вимагати з'єднання.

1.3 Постановка задачі

Зазвичай користувачі грають у ігри цього жанру для отримання емоцій від сюжету, а також насолоди від цікавого і нестандартного геймплею. Для вирішення таких потреб, було прийнято рішення про розробку ігрового застосунок у жанрі metroidvania із елементами карткової гри.

Ігровий застосунок у жанрі metroidvania із елементами карткової гри матиме в собі можливість збирати карти на рівнях та використовувати їх, що є новою механікою для даного жанру.

Дана гра матиме тематику Київської Русі та фокусуватимуться на її достовірному висвітленні, відповідно до наявних історичних джерел. Важливою складовою буде опора на автентичності фольклорних даних для відтворення міфологічного світу стародавніх слов'ян у найбільш приближеному до дійсності вигляді.

Окрім цього, важливо щоб відеогра вийшла на різноманітніші платформ. Це дасть гравцям можливість спробувати гру на їхній улюбленій платформі та значно збільшить аудиторію ігрового додатку. Задля цього було прийнято рішення використовувати для розробки ігровий двигун Ct.js, який підтримує одночасно персональні комп'ютери, веб та мобільні пристрої. В якості мови програмування обрано JavaScript, оскільки ця мова є досить гнучкою та має відповідні можливості для коректної праці на декількох різних платформах.

Також гра має бути оптимізованою. Швидкодія у грі не має бути меншою за 60 кадрів у секунду на рекомендованих налаштуваннях. Код гри має бути написаним за правилами чистого коду та чистої архітектури й працювати швидко та ефективно. Для цього мають використовуватися ефективні алгоритми, а також добре оптимізовані сторонні бібліотеки.

На додачу, гра повинна мати якісне звукове супроводження. Оскільки гра багато в чому спирається на сюжет та занурення гравця у ігровий світ, музика має на меті погрузити гравця у цей світ, створити необхідну атмосферу. Також звук має змінюватися в залежності від інтенсивності рівня та бою із ворогами і направляти гравця завдяки ритму музики. Звукові ефекти у грі мають бути доброякісними, миттєво та коректно відтворюватися і відповідати загальній атмосферності гри та конкретній музичній композиції на певному рівні.

Також, як було зазначено вище, гра повинна мати міні-ігри, щоб зробити ігровий процес більш різноманітним та цікавим для гравця. Оскільки у складі ігрового процесу є можливість збору карток, зібрані карти ватро використати у міні-іграх. Це створить для гравця зв’язок між ігровим світом та міні-грою, в яку гравець безпосередньо грає в даний момент. Важливим для героя є також занурення у міні-гру та відчуття прогресу при її проходженні.

1.3.1 Цільова аудиторія

Дана гра матиме досить широку цільову аудиторію. В першу чергу, це особи 16 та понад років. При обранні вікового рейтингу було враховано міжнародний стандарт PEGI, який являє собою Європейську рейтингову систему для оцінки вікового рейтингу відеоігор й іншого програмного забезпечення. Крім того, така система широко використовується в Україні. Подібний вибір обумовлено наступними чинниками:

- наявність у грі сцен насилля та бійки;
- гра може бути страшною;
- гра містить відсилки до азартних ігор;
- у грі наявні грубі жарти на бранні слова.

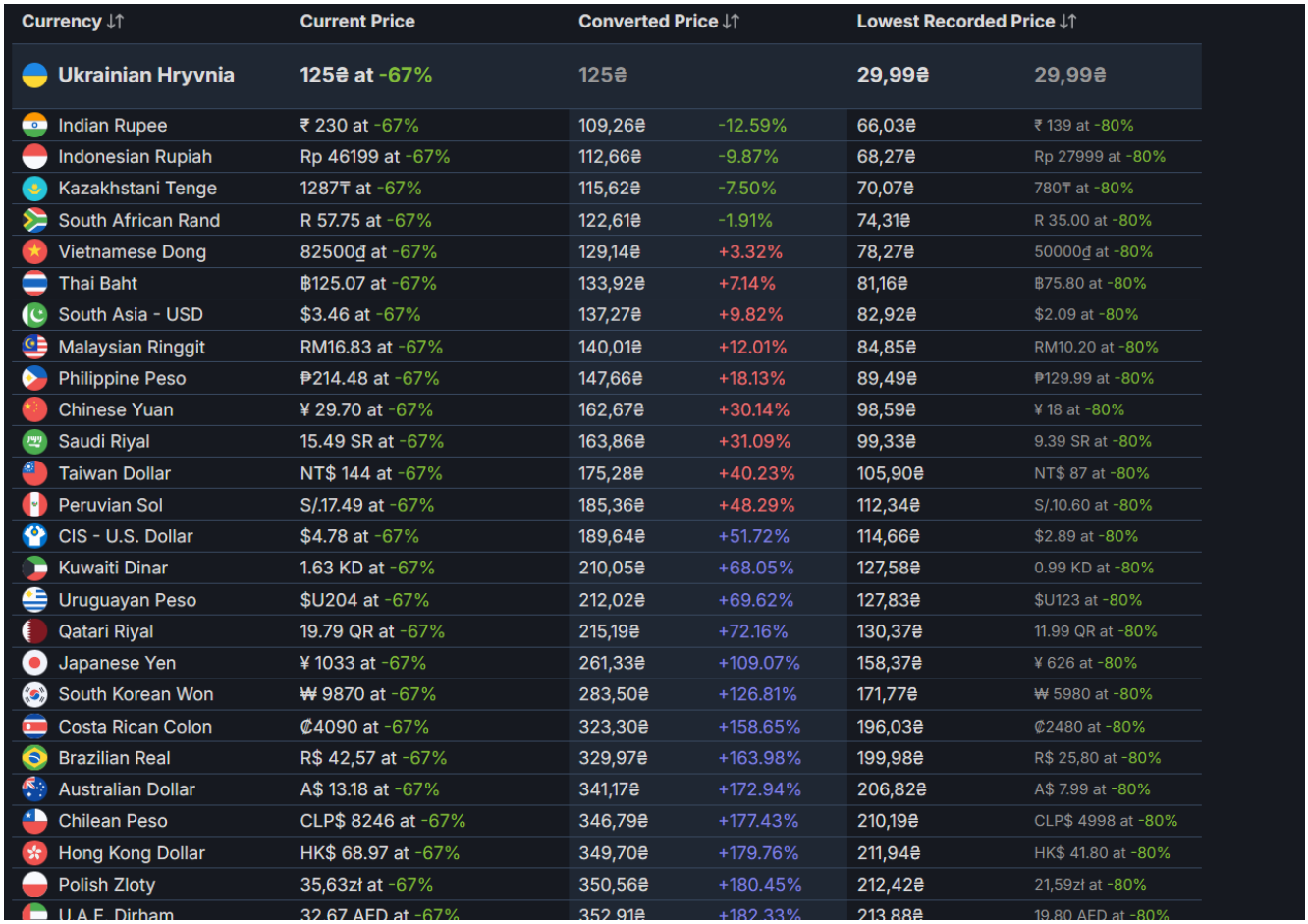
Вибірка людей від 16 років і більше, особливо чоловіків, обумовлена тим, що дана цільова аудиторія є найбільш широкою серед тих, кому дана гра може сподобатися. До того ж ця цільова аудиторія є найбільш платіжноздатною, що має позитивно сказатися на продажах та доходах від гри.

Важливим фактором у виборі такої аудиторії є зазвичай історична та середньовічна стилістика гри, яка набагато більш популярною серед чоловіків цієї вікової категорії. Але відеогра може задовільнити потреби широкої маси людейта принести задоволення для усіх типів гравців, незалежно від віку та статі.

1.3.2 Монетизація

Питання монетизації було вирішено зробити платною для гравців. Тобто наша гра має преміальну модель монетизації. Таке рішення було прийнято тому, що дуже багато ігор мають саме таку модель монетизації. Вона була однією із перших часів зародження ігрової індустрії як такої, а також дозволяє не обмежувати ігровий досвід гравця, надаючи йому повний вміст гри та повний ігровий досвід одразу після покупки. При цьому, якщо гру постійно підтримувати, інтерес гравців до неї не буде пропадати із часом [3], а також зможе залучити більше нових гравців.

Кажучи про ціни на гру, варто відмітити, що гра підтримуватиме регіональні ціни, тобто для кожного регіону будуть встановлені власні ціни у місцевій валюті. Такі платформи як Steam дозволяють встановити регіональні ціни і автоматично можуть корегуватись під різні регіони (див. рис. 1.3).



Currency	Current Price	Converted Price		Lowest Recorded Price	
Ukrainian Hryvnia	125₴ at -67%	125₴		29,99₴	29,99₴
Indian Rupee	₹ 230 at -67%	109,26₴	-12.59%	66,03₴	₹ 139 at -80%
Indonesian Rupiah	Rp 46199 at -67%	112,66₴	-9.87%	68,27₴	Rp 27999 at -80%
Kazakhstani Tenge	1287₸ at -67%	115,62₴	-7.50%	70,07₴	780₸ at -80%
South African Rand	R 57.75 at -67%	122,61₴	-1.91%	74,31₴	R 35.00 at -80%
Vietnamese Dong	82500₫ at -67%	129,14₴	+3.32%	78,27₴	50000₫ at -80%
Thai Baht	฿125.07 at -67%	133,92₴	+7.14%	81,16₴	฿75.80 at -80%
South Asia - USD	$3.46 at -67%	137,27₴	+9.82%	82,92₴	$2.09 at -80%
Malaysian Ringgit	RM16.83 at -67%	140,01₴	+12.01%	84,85₴	RM10.20 at -80%
Philippine Peso	₱214.48 at -67%	147,66₴	+18.13%	89,49₴	₱129.99 at -80%
Chinese Yuan	¥ 29.70 at -67%	162,67₴	+30.14%	98,59₴	¥ 18 at -80%
Saudi Riyal	15.49 SR at -67%	163,86₴	+31.09%	99,33₴	9.39 SR at -80%
Taiwan Dollar	NT$ 144 at -67%	175,28₴	+40.23%	105,90₴	NT$ 87 at -80%
Peruvian Sol	S/.17.49 at -67%	185,36₴	+48.29%	112,34₴	S/.10.60 at -80%
CIS - U.S. Dollar	$4.78 at -67%	189,64₴	+51.72%	114,66₴	$2.89 at -80%
Kuwaiti Dinar	1.63 KD at -67%	210,05₴	+68.05%	127,58₴	0.99 KD at -80%
Uruguayan Peso	$U204 at -67%	212,02₴	+69.62%	127,83₴	$U123 at -80%
Qatari Riyal	19.79 QR at -67%	215,19₴	+72.16%	130,37₴	11.99 QR at -80%
Japanese Yen	¥ 1033 at -67%	261,33₴	+109.07%	158,37₴	¥ 626 at -80%
South Korean Won	₩ 9870 at -67%	283,50₴	+126.81%	171,77₴	₩ 5980 at -80%
Costa Rican Colon	₡4090 at -67%	323,30₴	+158.65%	196,03₴	₡2480 at -80%
Brazilian Real	R$ 42,57 at -67%	329,97₴	+163.98%	199,98₴	R$ 25,80 at -80%
Australian Dollar	A$ 13.18 at -67%	341,17₴	+172.94%	206,82₴	A$ 7.99 at -80%
Chilean Peso	CLP$ 8246 at -67%	346,79₴	+177.43%	210,19₴	CLP$ 4998 at -80%
Hong Kong Dollar	HK$ 68.97 at -67%	349,70₴	+179.76%	211,94₴	HK$ 41.80 at -80%
Polish Zloty	35,63zł at -67%	350,56₴	+180.45%	212,42₴	21,59zł at -80%
U.A.E. Dirham	32.67 AED at -67%	352.91₴	+182.33%	213.88₴	19.80 AED at -80%

Рисунок 1.3 – Приклад регіональних цін у Steam (https://steamdb.info/app/1057090)

Подібний підхід дозволить зрівняти гравців із різних частин світу та зробити гру однаково добро проданою по всьому світу, оскільки в іншому випадку для гравців із деяких країн гра була б занадто дорогою, а для гравців із інших регіонів – занадто дешевою, що відштовхнуло би частину покупців та знизило би загальну прибутковість ігрового програмного продукту.

Окрім цього, в перспективи для підтримки гри та інтересу до неї можна випустити додатковий завантажуваний контент (DLC), який можна продавати за певну плату. Це дозволить підтримувати та піднімати інтерес гравців до програмного продукту, наповнювати відеогру контентом, розвивати ігровий світ та сюжет, а також отримувати додатковий прибуток від гри [4].

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

Основними програмними частинами ігрового програмного застосунку The Saga of Sigurd в жанрі metroidvania із елементами карткової гри є веб-застосунок, застосунок для персональних комп’ютерів та мобільний додаток. Веб-версія додатка орієнтована на браузер Chrome та працює на версіях 87 і вище. Застосунок для персональних комп’ютерів сумісний із операційними системами Windows 10, Ubuntu та Mac OS 10.10 Yosemite та вище. Мобільний додаток підтримує платформи Android та iOS. Для мінімальна версія Android – версія 5.0 Lollipop, для iOS – 10 версія.

Щодо мінімальних системних вимог до технічних характеристик ігрової платформи, вони мають наступний вигляд:

а) персональні комп’ютери:
   1) Intel Pentium 4;
   2) NVIDIA GeForce GT 640M;
   3) 512 МБ вільної оперативної пам’яті;
   4) 3 ГБ вільного місця на диску;

б) мобільні платформи;
   1) 700 МГц та більше на графічному процесорі;
   2) 3 ГБ вільного місця на диску;
   3) процесор із двома ядрами та частотою 1,8 ГГц та більше;

в) веб-застосунок;
   1) 800 МГц та більше на графічному процесорі;
   2) 512 МБ вільної оперативної пам’яті;
   3) 1,7 ГБ вільного місця на жорсткому диску;
   4) наявність DirectX 8.1 або новішої версії;
   5) процесор Intel Pentium 4 або новіше із підтримкою SSE3.

З точки зору вимог до коду, код програмного продукту має бути написаний за основними принципами чистого коду, використовуючи основні принципи

SOLID. Програмний код має працювати оптимально, використовуючи швидкі та ефективні алгоритми й структури даних там, де це необхідно.

Особливо важливим для The Saga of Sigurd, як і для будь-якої іншої відеогри, є ігровий дизайн [5]. Для даного програмного продукту основними віхами дизайну є наступні:

- тетра гри;
- USP;
- структура рівнів;
- переміщення;
- бойові механіки;
- міні-ігри.

Тетра гри має наступний вигляд:

- гра для персональних комп’ютерів, веб-сайту та мобільних пристроїв;
- жанр metroidvania;
- історія про головного героя, обраного богами;
- стилістика слов’янського фентезі;

Крім того, можна назвати наступні елементи USP:

- музика та графіка стилізовані під часи Київської Русі;
- детально пророблений світ і багато різних локацій;
- детально та коректно відтворений світ слов’янської міфології;
- оригінальна історія та цікавий головний герой.

Ігровий світ умовно відкритий для дослідження. Хоча сюжет відеогри лінійний і гравець має чітко слідувати за сюжетом, але головний герой особо необмежений у дослідженні ігрового світу і більшість локацій доступні для дослідження з самого початку гри, хоча й головна задача гри – виконати основний квест. Деякі локації, відповідно до законів жанру, можна відкрити після здобуття певних ігрових предметів та вирішення загадок.

Важливо також зазначити, що локації умовно складаються у регіони. Різниця в тому, що регіон поділений на декілька локацій, а локації всередині

регіону мають однаковий набір ворогів. Це необхідно для того, щоб зробити ігровий світ більш різноманітним, відмовитися від деяких ігрових умовностей, що дозволить більш повно та глибоко показати світ гри, погрузити гравця в атмосферу історії Київської Русі. Окрім того, рівні всередині одного регіону відрізняються, завдяки спільним елементам вони викликають у гравця відчуття знайомого, що дозволяє повільно та прогресивно будувати складність.

Структурно усі ігрові рівні можна поділити на три різні світи: Навь, Явь та Правь. Кожна з цих частин є повноцінним ігровим світом із власними рівнями, які різняться між собою, а також власними персонажами, ворогами. Для переміщення між ними є окрема локація із мостом. Але щоб головний герой міг вільно переміщатися між світами, йому потрібно спочатку отримати артефакт в одній із стартових локацій ігрового світу після проходження відповідного завдання.

Оскільки світ гри має 2D форму, герой може вільно переміщатися по ньому у двох напрямках: висота та ширина. Кожна з локацій містить елементи платформінгу та поєднана із іншими локаціями, що дозволяє гравцю вільно переміщуватися між ними. Також важливим елементом у системі переміщення є прогрес головного героя та відкриття нових предметів під час гри. Наприклад, коли гравець здобуде одну із карт з покращеннями, головний герой зможе стрибати двічі, що відкриє деякі нові локації, до яких раніше гравець не міг дістатися.

Стосовно бойових механік, їх умовно можна поділити на магічний бій із використанням карток та фізичний бій із використанням героєм звичайної зброї. Фізичний бій у грі виконаний за аналогією до інших подібних ігор. Гравець має зброю і може використовувати її для вбивства ворогів. І у головного героя, і у ворогів є певна кількість здоров’я та атаки віднімають його частину від атакованого. Гравець додатково може збирати покращення для зброї протягом гри.

Кажучи про магічний бій, у The Saga of Sigurd він заснований на системі карток. Під час гри гравець підбирає картки і може використати їх у бою. Кожна із карток витрачає певну частину мани, яку можна відновити у головного героя

вдома або використовуючи спеціальні зілля, які підбираються на різних рівнях. Найрізноманітніші магічні картки також дають різні ефекти. Наприклад, одна із карток дозволяє бити ворогів блискавкою, а інша підпалює землю та наносить шкоду ворогам, які стоять на землі. Картки можна покращити, якщо знайти парні їхні копії під час гри.

Також, стосовно міні-ігор, у грі наявна повноцінна карткова гра: Відьмяріца. Сутність цієї гри полягає в наступному: кожний гравець обирає колоду та на початоку гри в нього наявні п’ять різних карток. Колоди відносяться до чотирьох різних фракцій, кожна з яких має повністю унікальний набір карток: богатирі, сторожі лісу, створіння темряви та ковен Баби Яги.

Варто зазначити, що дизайн кожної з карток було повністю згенеровано штучним інтелектом. Кожен гравець має власний вівтар, який накопичує силу, коли гравець кладе на стіл картку. Коли обидва гравці поклали картки на стіл, починається стадія битви. Якщо картки обох гравців стоять в протилежних комірках, вони б’ються між собою, інакше шкода від картки наноситься вівтарю ворога.

Важливою складовою цієї гри є зміна сезонів. Кожен із сезонів має власні позитивні та негативні ефекти, наприклад, весна зцілює картки на одну одиницю здоров’я кожен хід. Гра завершується в одному із випадків: коли гравець набирає 15 очок для свого вівтаря або коли пройшло 5 ігрових років (тобто повний круг зміни чотирьох ігрових сезонів п’ять разів). У другому сценарії перемагає гравець, який набрав більше очок. У випадку, коли виконана одна із умов, але обидва гравці мають однакову кількість очок на вівтарі – наступає нічия.

Для того, щоб перейти у таку гру, потрібно почати розмову із внутрішньоігровим персонажем у селищі, де живе головний герой. Персонаж запропонує головному герою зіграти у цю гру. Якщо герой погодиться – почнеться гра.

3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 UML проєктування ПЗ

Під час проектування ігрового застосунку “The Saga of Sigurd” в жанрі metroidvania були визначено основні функції зі сторони гравця, а також на їх основі створено Use-case UML діаграму (див. рис. 3.1).

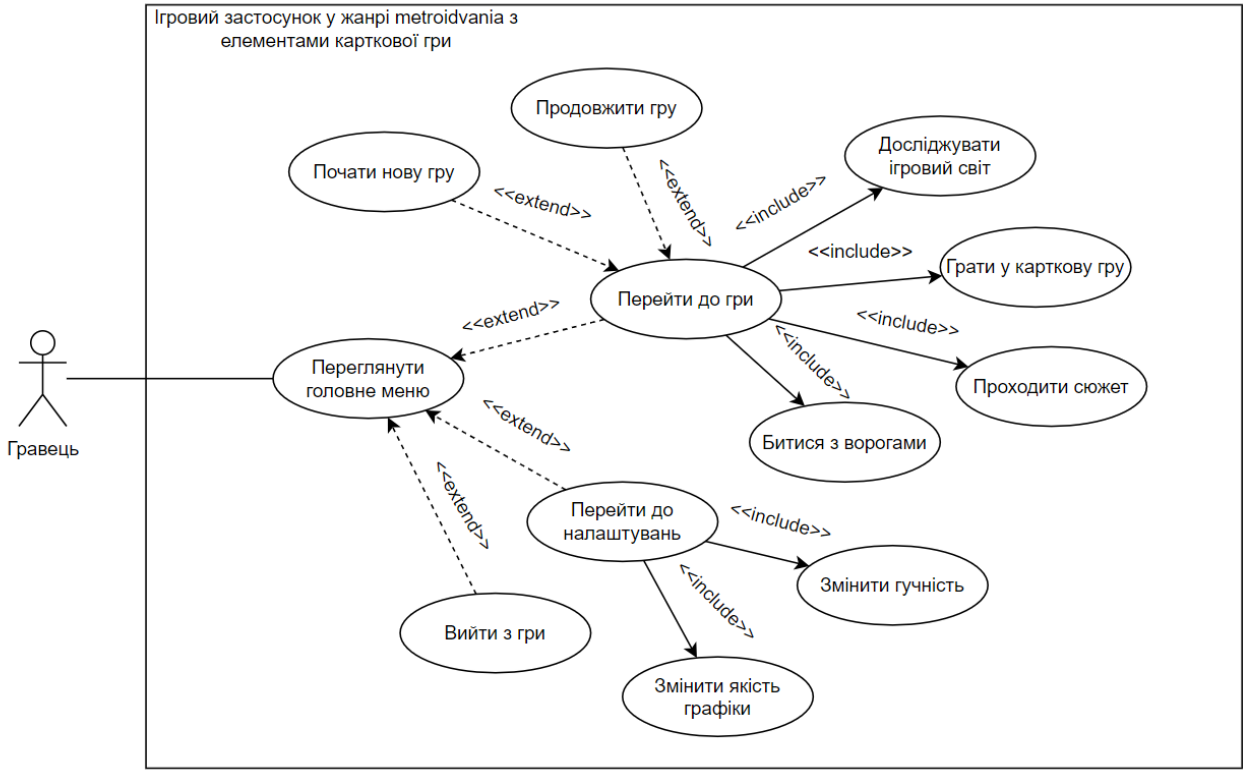


Рисунок 3.1 – Use-case діаграма ігрового застосунку (рисунок виконаний самостійно)

Як можна побачити з цієї діаграми, основним елементом, з яким найперше взаємодіє гравець, є перегляд головного меню. Виходячи з головного меню, гравець може перейти до самої гри, перейти да налаштувань та вийти з гри.

Налаштування у грі включають в себе налаштування гучності звуку та налаштування якості графіки. Саме ці елементи є найбільш важливими в налаштунках, оскільки вони в першу чергу впливають на ігровий досвід гравців.

При перехід до самої гри, є опції продовження гри на початку нової гри. Ключова різниця між ними полягає в тому, що при продовженні гри

завантажується останнє збереження. Важливо відмітити, що збереження прогресу у грі відбувається автоматично на певних етапах проходження гри, а також при переході між регіонами всередині ігрового світу. А при початку нової гри прогрес гравця стирається та відеогра починається з самого початку.

Також всередині самої гри гравець має наступні опції: дослідження ігрового світу, гра у карткову гру, проходження сюжетної лінії та боротьба з ворогами.

Дослідження ігрового світу – одна із складових частин вищеописаного програмного продукту. Гра фокусується на найбільш детальному відтворенні стилістики, і дослідження локацій надає гравцю можливості зануриться повністю в ігровий стародавній світ. Окрім того, це один із можливих способів покращити головного героя, оскільки дизайн рівнів гри спонукає шукати захованні предмети та картки, які значно поліпшують героя та роблять легшим проходження сюжетної лінії.

Як було зазначено вище, карткова міні-гра починається з діалогу з селянином, що знаходиться у селищі головного героя неподалік від його домівки. З точки зору ігрового процесу, картки не надають жодних переваг і прогрес по сюжету не залежить від того, чи грав гравець у них, чи ні. Але, при цьому, з боку дизайну гри, дана частина дозволяє гравцеві відволіктися від основної сюжетної лінії та робить ігровий процес більше різнобічним та цікавішим, одночасно із цим погружаючи гравця більш детально в атмосферу на стилістику гри.

Проходження сюжетної частини гри – основна частина даного програмного ігрового продукту. Проходячи сюжет, гравець дізнається історію головного героя та його протистояння із князем Володимиром, а також отримає багато емоцій і вражень. Виключно при проходженні сюжету гравець відкриє усі ігрові локації, зіткається з усіма ворогами та знаходить повну колоду магічних карток.

Кажучи про боротьбу із ворогами – це одна з основних частин ігрового процесу. Під час боротьби із ворогами головний герой зможе попрактикуватися із основними частинами бойової системи, а також краще вивчити бестіарій ігрового світу та навчитися добре використовувати різні види магії. Треба відмітити, що різні вороги у грі мають різні слабкості відносно декількох видів магії, і у процесі

боротьби із ними герой може експериментувати та дізнаватися на практиці по різні комбінації, які будуть більш ефективними проти ворогів. Бойова система гри сприяє такому дослідженню.

3.2 Проєктування архітектури ПЗ

Як було зазначено вище, під час проектування було прийнято рішення, що гра повинна мати чисту архітектуру. Одночасно із цим, оскільки було поставлено вимогу, що гра підходить виключно для одного гравця та повинна мати можливість працювати без підключення до мережі інтернет, це накладає свої певні обмеження на архітектуру (див. рис. 3.2).

Рисунок 3.2 – Діаграма розгортання (рисунок виконаний самостійно)

В цілому, гра має лише дві частини: клієнт із графічним інтерфейсом та сервіс ігрової логіки.

Варто зазначити, що частково таке рішення було продиктовано архітектурою та можливостями ігрового двигуна, оскільки за замовчуванням він відділяє графічну частину від ігрової логіки програмного продукту. Одночасно із цим, через необхідність працювати без підключення до мережі інтернет, уся ігрова логіка має знаходитися локально на комп’ютері чи іншій платформі користувача, а також не повинна перемежатися із інтерфейсом за правилами чистої архітектури [6].

На даний момент, UI клієнт складається із текстур для усіх наявних елементів гри, таких як головний герой, інші персонажі, вороги, оточення, картки, магія та усі ігрові анімації. Окрім цього, шар ігрового інтерфейсу містить усе музичне супроводження ігрового програмного застосунку, включаючи усі звукові ефекти та музичне супроводження гри, розбите по доріжкам.

Кажучи про анімації, зазначимо, що в ігровому двигуні усі анімації сприймаються частково як окремі текстури. Для підтримання чистої архітектури, кожна анімація разом із іншими текстурами, що відносяться до певного ігрового об’єкта, розташовані в окремій теці. Для анімації прописана кількість кадрів та розміри кожного кадру, а також розмір сфери зіткнення. Але, при цьому, всередині шару ігрової логіки було створено шаблони, які дозволяють запускати, запиняти та перемикатися між анімаціями й текстурами, залежно від ігрового процесу та логіки того, що відбувається на екрані гравця.

Аналогічним чином перемикається і звукове та музичне супроводження всередині гри. Музичні композиції, що знаходяться всередині гри, не є повністю цілісними, оскільки вони розбиті на декілька пов’язаних доріжок. Шар ігрової логіки, в свою чергу, запускає лише певну кількість доріжок, а їх кількість залежить від інтенсивності бою, наявності діалогу чи заставки всередині гри, та інших показників ігрового процесу.

Шар логіки гри складається з глобальних скриптів, які застосовуються до всієї гри цілком, а також окремих скриптів для кожного шаблону, тобто ігрового об’єкту всередині гри. Разом із цим, даний шар містить логіку для кожного окремого рівня, що включає в себе запуск музичного супроводження, створення ворогів на рівні та логіку поводження і розташування об’єктів. Окрім цього, кожен рівень тут має посилання на пов’язані із ним, що дозволяє створити локації та переходи між ними.

Щодо мов програмування треба відмітити, що логіка гри може бути написана на трьох мовах програмування: CoffeeScript, JavaScript та TypeScript. Завдяки можливостям ігрового двигуна, код, що був написаний на будь-який із цих трьох мов програмування, може безшовно та цілісно взаємодіяти із іншими

фрагментами коду, написаними на двох інших мовах програмування. Такий підхід добре підходить для великих команд та певних стилів архітектури. Враховуючи наявний в нас архітектурний стиль та цілісний підхід до розробки, було прийнято рішення писати логіку гри виключно мовою програмування JavaScript, яка є найбільш сучасною, актуальною та, одночасно із цим, найбільш гнучкою з-поміж трьох вищеназваних.

3.3 Проєктування структури зберігання даних

В якості місця зберігання даних ми використовували можливості локального сховища, які надає ігровий двигун Ct.js. Таким чином, усі ігрові дані зберігаються локально, ігровий програмний продукт не використовує базу даних для збереженні інформації. Зазначимо, що ігровий двигун має власний аналог SQLite, що має назву «Таблиці типів контенту». На даний час, гра має три основні сутності: головний герой, вороги та діалоги (див. рис. 3.3).

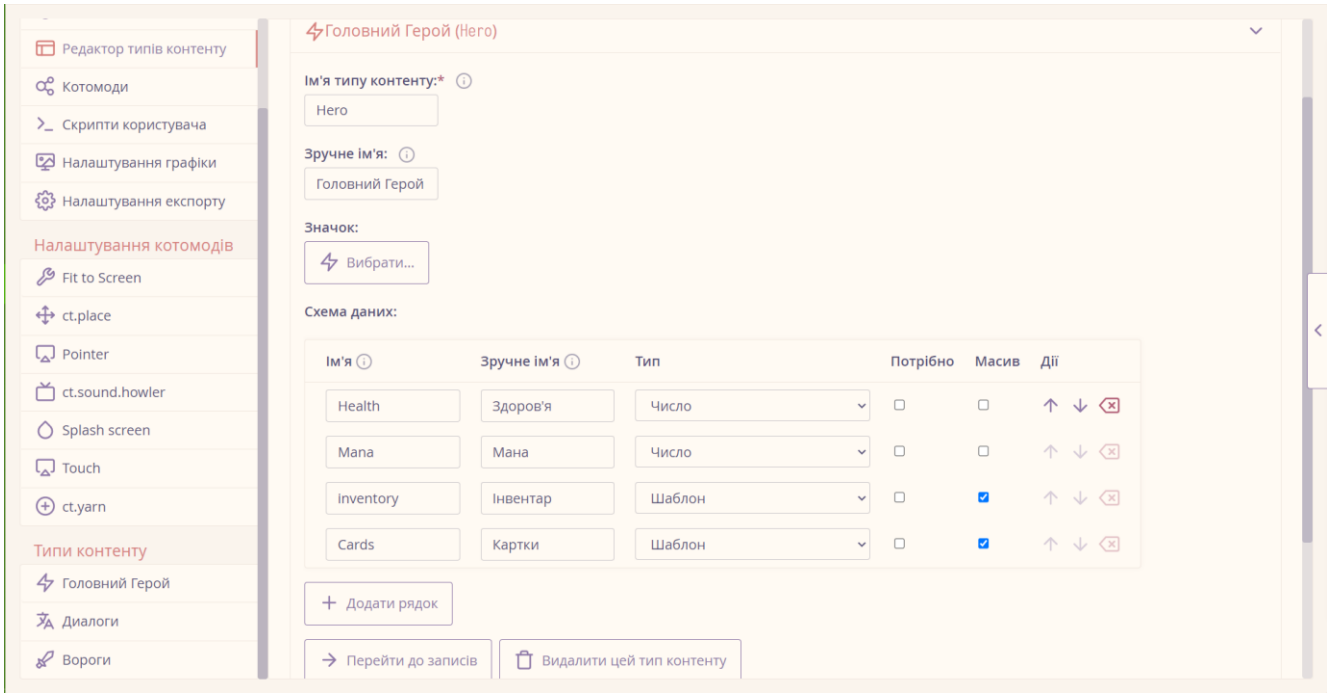


Рисунок 3.3 – Вигляд редактору типів контенту (рисунок виконаний самостійно)

В боковому меню наявні три вищеназвані типи контенту. В редакторі відкрита сутність головного героя та можна побачити її поля. Зазначимо, що оскільки ігровий двигун було написано мовою програмування JavaScript на основі

технології Electron та движку V8, завдяки цьому, дані типи контенту всередині двигуна та під час розробки використовуються у форматі JSON [7].

Таблиця контенту складається зі здоров'я, мани, інвентаря та карток. Останні дві рядки містять у собі масиви даних, що дозволяє достатньо гнучко керувати інвентарем та картками головного героя, дозволяючи їх додавати, видаляти та редагувати в будь який момент з будь-якої частини коду гри.

Окрім цього, гра має таблицю із ворогами (див. рис. 3.4), яка зберігає інформацію про стан ворогів: їхнє здоров'я за замовчуванням, ім'я та посилання на шаблон, що містить логіку дій ворога.

При проектуванні сутностей таблиці, слідуючи офіційним рекомендаціям, кожну повторювану сутність було винесено в окрему таблицю, а також потім їх було нормалізовано і приведено до третьої нормальної форми.

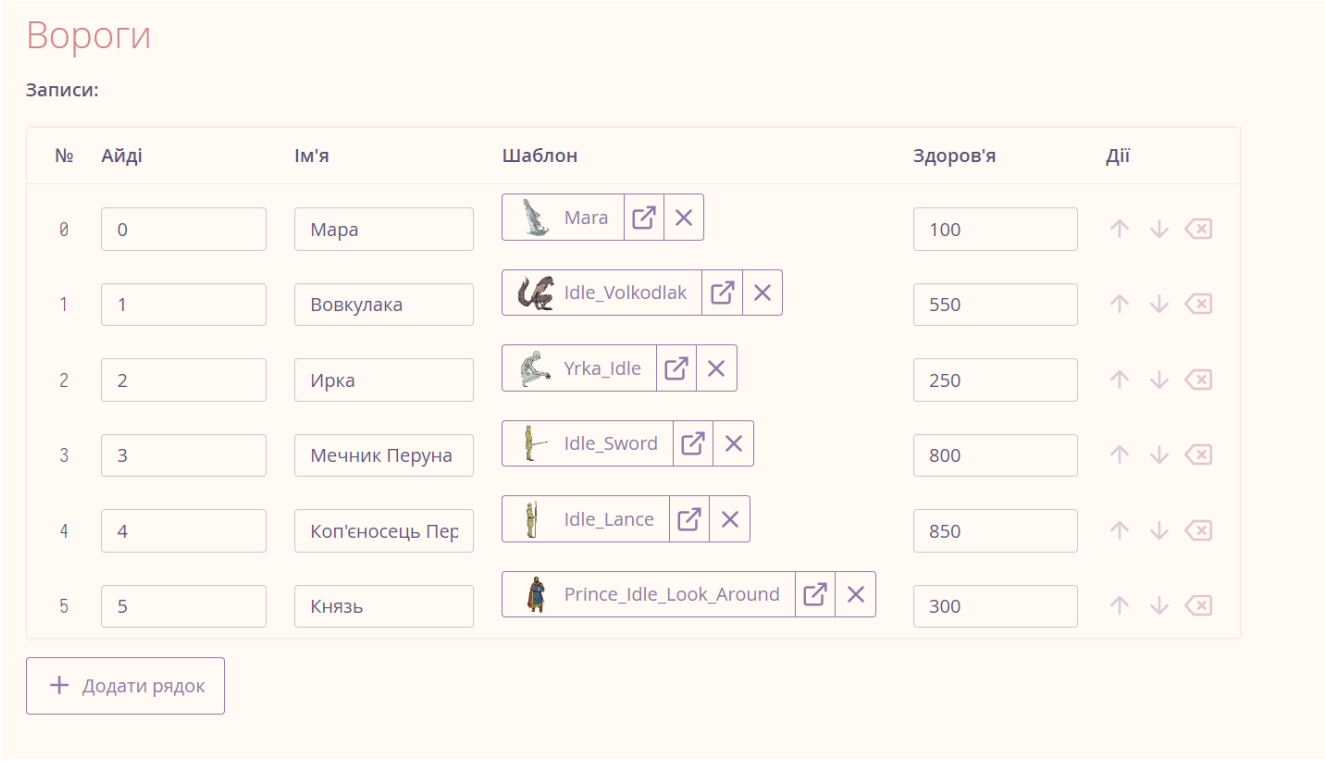


Рисунок 3.4 – Таблиця з ворогами (рисунок виконаний самостійно)

Як можна побачити, в таблиці наявні ідентифікатор для кожного типу ворога, а також задане початкове здоров'я за замовчуванням. Такий підхід, із використанням даної таблиці, дозволяє зручно керувати великою кількістю ворогів, оскільки тут зберігаються певні значення за замовчуванням, і для

кожного окремого ворога при необхідності з точки зору ігрового процесу їх можна змінювати та потім повертати до значення за замовчуванням.

3.4 Приклади найцікавіших алгоритмів та методів

Одним із цікавих та досить важливих рішень був метод для вибору музики, яка має програватися на кожному рівні. Так, в якості глобального скрипту, який може використовуватися будь-де у коді, було написано наступний метод:

```
function playMusicLevel(enemyCount, levelName) {
    if (enemyCount == 0) {
        ct.sound.spawn(`${levelName}_base`, {loop: true});
    }
    else if (enemyCount > 0 && this.enemyCount <= 3) {
        ct.sound.spawn(`${levelName}_1`, {loop: true});
    }
    else if (enemyCount > 3 && this.enemyCount <= 5) {
        ct.sound.spawn(`${levelName}_2`, {loop: true});
    }
    else if (enemyCount > 5) {
        ct.sound.spawn(`${levelName}_3`, {loop: true});
    }
}
```

Як можна побачити з коду вище, в залежності від кількості ворогів на екрані викликається необхідна музична композиція. Оскільки тут використовуються строкові шаблони, кожна з доріжок на кожен рівень має відповідати прийнятій у коді конвенції з найменування, щоб даний метод коректно виконувався та викликав саме необхідну музику.

Можна побачити, що музичні композиції розбиті на чотири частини, а саме базову та на три і більш інтенсивних доріжок. Вони інкрементно додаються в залежності від кількості ворогів на екрані.

Зазначимо, що для розрахунку ворогів на екрані на кожному рівні наявна змінна, що зберігає їхній стан. Таким чином, достатньо викликати цей метод всередині логіки рівня та передати в якості аргументів назву рівня та змінну, що відповідає за кількість ворогів, щоб на цьому рівні програвалася необхідна музика.

В свою чергу, для розрахунку першого аргументу використовується наступний код:

```
let enemyCounter = ct.content.Enemy.reduce((acc, enemy) => acc += ct.templates.list[enemy].length)
```

Як можна побачити, ми звертаємося до вищеописаного редактору контенту, отримуємо список ворогів у нього і через вбудовані методи масивів у мові програмування JavaScript, особливо метод reduce, ми рахуємо кількість даних ворогів на нинішньому рівні. Масив ct.templates.list відображає лише ті шаблони, які знаходяться на поточному рівні, не враховуючи шаблони з інших рівнів. У випадку, якщо дане значення оновлюється по певному таймеру, який для кожного рівня можна задати окремо, цю змінну можна передати в вищезазначений метод, щоб створити на рівні необхідне музичне супроводження.

Карткова міні-гра має в своїй логіці зміну сезонів. Для того щоб зміна працювала правильно, маємо розраховувати кількість ходів та в залежності від номеру нинішнього ходу оновлювати сезон.

```
function getCurrentSeason(turn) {
    const seasons = ['spring', 'summer', 'fall', 'winter'];
    const index = Math.floor((turn - 1) / 4) % seasons.length;
    return seasons[index];
}
```

Як можна побачити з коду, даний метод обчислює сезон за формулою, округлюючи завдяки вбудованій функції даної мови програмування та обчислюючи залишок від ділення . Даний код використовується на рівні з ігровим столом в режимі карткової гри:

```
        let season = getCurrentSeason(this.globalTurn);

        if (ct.rooms.list['Spring'].length == 0 && season.toLowerCase() == "spring") {
            ct.rooms.append('Spring')
        }
        else if (ct.rooms.list['Summer'].length == 0 && season.toLowerCase() == "summer") {
            ct.rooms.append('Summer')
```

```
        }
        else     if     (ct.rooms.list['Fall'].length     ==     0     &&
season.toLowerCase() == "fall") {
            ct.rooms.append('Fall')
        }
        else     if     (ct.rooms.list['Winter'].length     ==     0     &&
season.toLowerCase() == "winter") {
            ct.rooms.append('Winter')
        }
```

Задля запобігання проблем із правописом, усі сезони переводяться у нижній реєстр, а також логіка матчу для кожного рівня знаходиться в окремій кімнаті ігрового двигуна і виконується коли ми додаємо цю кімнату поверх нашої головної.

Окрім того, в кімнатах знаходиться логіка гри, кожна із кімнат з сезонами має своє графічне відображення та працює як шар інтерфейсу у самій грі. Таким чином, гравець не тільки отримує ефекти від певного сезону, але й бачить графічно який зараз сезон у середині міні-ігри. Змінна this.globalTurn, що знаходиться у коді, представляє собою змінну кімнати, та відстежує глобальний хід. Різниця між звичайним ходом і глобальним лише у тому, що глобальний хід дорівнює двом звичайним.

Таке технічне рішення необхідне для того, щоб обидва гравця зробили свій хід, створюючи тим самим баланс. Інакше це призвело б до того, що той гравець, що походив першим, мав би більше шансів на виграш, бо де-факто мав би на один хід більше.

3.5 Створення UI/UX

Користувацький інтерфейс у грі створений за основними принципами проектування інтерфейсів та користувацького досвіду та відповідає основним евристикам Нільсена [8]:

- видимість статусу системи;
- відповідність між системою і реальним світом;
- користувацький контроль і свобода;
- послідовність і стандарти;

– запобігання помилкам;
– розпізнавання замість необхідності згадувати;
– гнучкість і ефективність використання;
– естетичний і мінімалістичний дизайн;
– допомога користувачам розпізнавати, діагностувати й усувати помилки;
– довідка та документація.

Окрім цього, важливою складовою для даного програмного продукту була стилізація користувацького інтерфейсу під стилістику гри, а саме під часи Київської Русі (див. рис. 3.5).

Рисунок 3.5 – Головне меню гри (рисунок виконаний самостійно)

Як можна побачити на зображенні, гра має головне меню із відповідним зображенням та набором кнопок, при натисканні на які, гравець переходить у відповідний розділ, де зазначено в діаграмі вище. Для підтримання стилістики гри, головне меню має історичну картину із зображенням частини Києва у той історичний період, а також кожна кнопка завдяки певній комбінації шрифту із чорним фоном допомагає створити потрібну атмосферу. Важливою є і музика, а

саме оркестр народних музичних інструментів разом із звуками води, що гарно підходить до наявного екрану заставки.

Наразі в головному меню наявні 4 опції для гравця: нова гра, завантаження гри, налаштування та вихід з гри. Як було зазначено вище, при натисканні на нову гру гравець починає гру з нуля, при завантаженні гри відтворюється останнє збереження, при відкриті налаштувань гравець переходить на екран налаштувань, а кнопка виходу вимикає гру.

За основними принципами дизайну, на кожну дію гравця гра відповідає графічно, тим самим відповідаючи гравцеві та даючи зрозуміти, що він робить те, що потрібно, показуючи, що гра відповідає та працює як слід, а також захищаючи користувача від помилок. Наприклад, під час запуску гри та її завантаження, з’являється екран із зображенням, що показує гравцеві, що гра завантажується (див. рис. 3.6).

Рисунок 3.6 – Завантажувальний екран гри (рисунок виконаний самостійно)

Цей екран завантаження має власну анімацію, а саме збільшення під час процесу завантаження гри під час першого запуску. Таким чином, гравець може зрозуміти, що коли картинка заповнить весь екран – гра завантажиться та гравець перейде до головного меню, звідки зможе перейти далі за вибором.

Окрім того, важливим є екран смерті головного героя. Коли герой помирає, після невеликої затримки з’являється екран, що повідомляє гравцеві про закінчення гри, а також, що він програв (див. рис. 3.7).

Рисунок 3.7 – Екран смерті (рисунок виконаний самостійно)

На зображенні видно напис про закінчення гри, виконаний в народному стилі. Використання такого народного шрифту разом із відповідним звуковим супроводом так само одночасно повідомляє гравцеві і не збиває атмосферу гри.

Важливою також є реакція гри на отримання головним героєм шкоди. Коли ворожа істота атакує гравця – екран починає трохи труситися та герой відскакує назад, що повідомляє про отриману шкоду. Це особливо важливо на початкових етапах, оскільки дає гравцеві зрозуміти які істоти та предмети на карті є ворожими до нього.

Даний спосіб отримання шкоди допомагає регулювати складність гри, оскільки герой не може в цей час нічого зробити, а скачок назад може призвести до падіння та смерті головного героя в певних місцях. Це спонукає гравця бути обережним під час гри та ретельно досліджувати місцевість перед тим, як ступити у бій, розцінювати свої сили, навички та наявність достатньої кількості предметів та магічних карток, оскільки їх нестача може призвести до смерті.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

В результаті проектування та розробки вищеописаного ігрового застосунку було прийнято багато гарних програмних рішень, що відповідають найкращим практикам написання коду для програмного забезпечення [9].

Із урахуванням можливостей ігрового двигуна було спроектовано чисту архітектуру для усього програмного застосунку. Основна логіка гри розбита на рівні та шаблони. Останні, в свою чергу, інкапсулюють текстури. Кожну категорію було окремо розбито на теки, які відповідають кожному із рівнів гри. Також додатково створено спільні теки: теку для користувацького інтерфейсу разом із окремими кімнатами для різних частин ігрового меню (див. рис. 4.1).

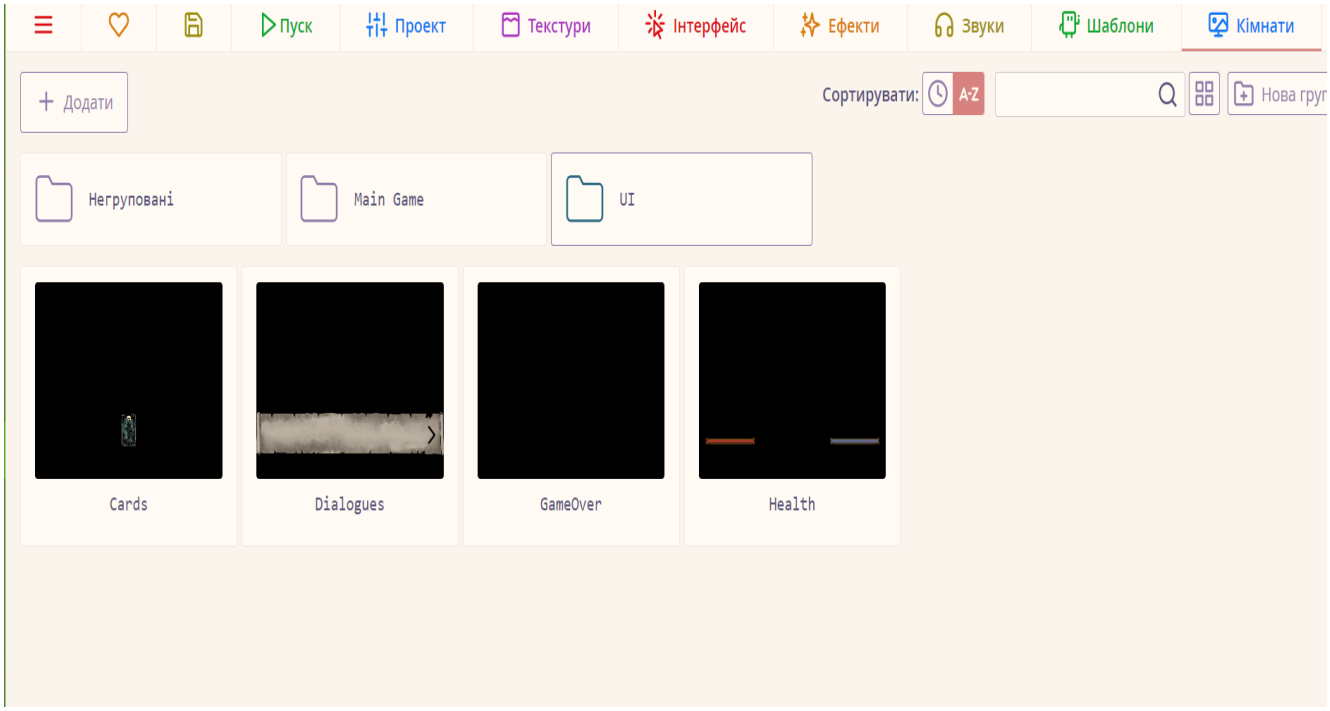


Рисунок 4.1 – Тека користувацького інтерфейсу (рисунок виконаний самостійно)

На зображенні видно меню кімнат ігрового двигуна та наявні кімнати для користувацького інтерфейсу. Як можна побачити, гра має такі кімнати інтерфейсу, як меню карток, діалоги, екран кінця гри та смуги здоров’я й мани. Таке рішення про відокремлення цих кімнат є важливим, оскільки дозволило відокремити кімнати, які використовуються для інтерфейсу від, безпосередньо ігрових рівнів, оскільки всередині самого двигуна вони є однаковими кімнатами.

Крім того, щоб ці кімнати відображалися поверх рівнів, в налаштуваннях кожної із них встановлено флаг для використання кімнати як графічного інтерфейсу (див. рис. 4.2).

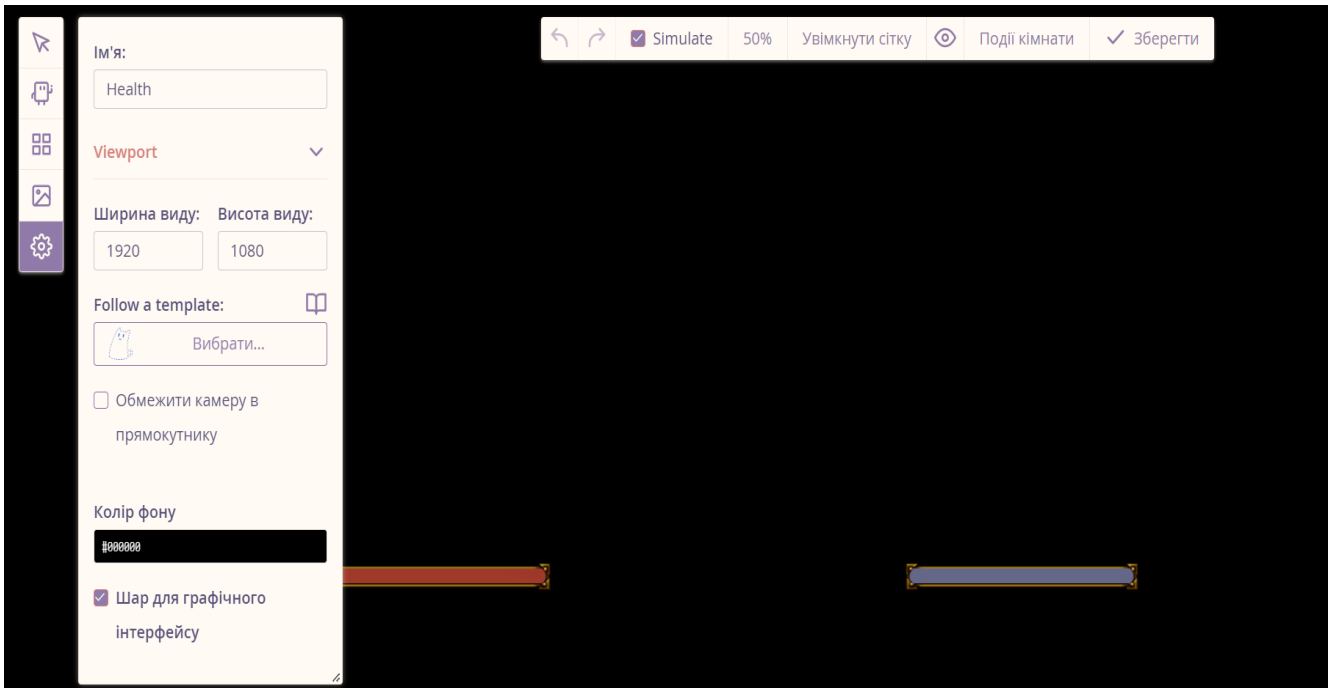


Рисунок 4.2 – Налаштування кімнати смуг здоров’я та мани (рисунок виконаний самостійно)

Як можна побачити, кімната має ввімкнений флаг для використання в якості шару для графічного інтерфейсу. Ширина та висота налаштовані на максимальний розмір екрану для цільових платформ нашого ігрового застосунку. Зазначимо, що цей розмір є відносним і для екранів із меншим розміром він стиснеться відносно самого екрану.

Саме для того, щоб це працювало відповідно, камера не обмежена в прямокутнику. Оскільки камера не слідує за якимось шаблоном, полоси здоров’я та мани є статичними та не рухаються на екрані, це створює знайомий та позитивний ігровий досвід користувача.

Стосовно архітектури та звукового супроводу гри, подібний підхід до архітектури було використано і під час розробки звукового супроводу гри (див. рис. 4.3).

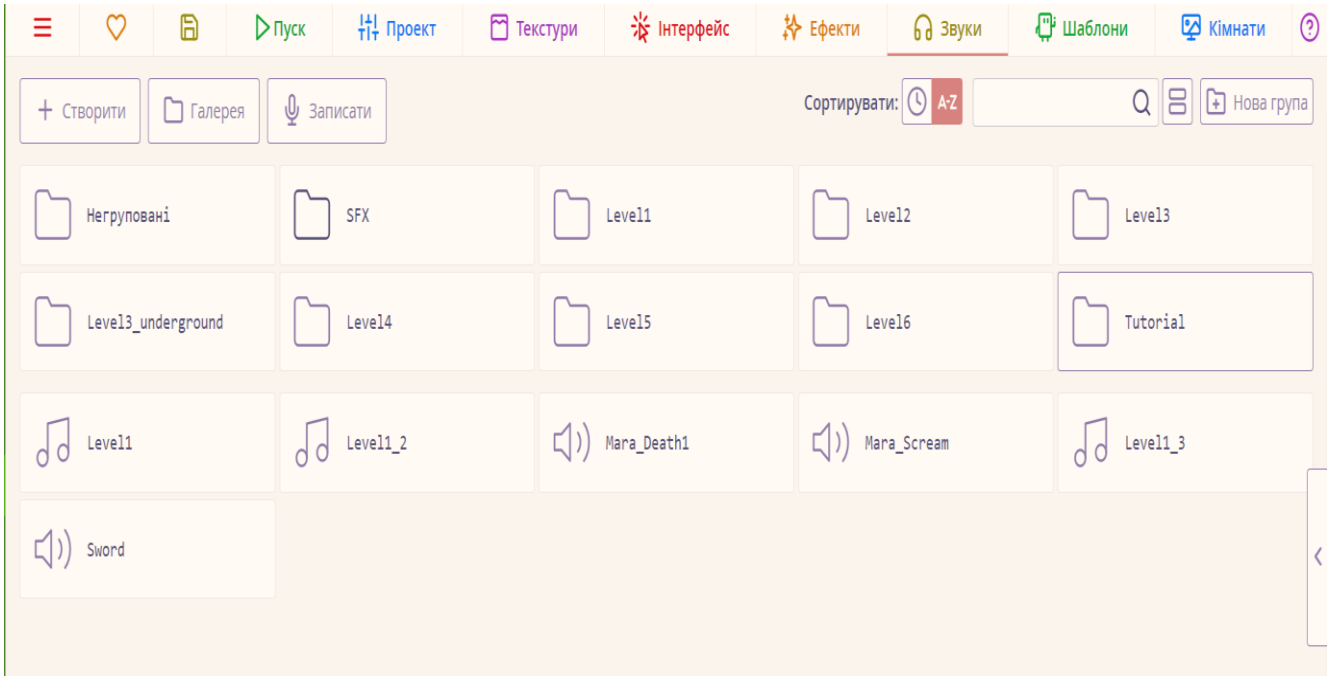


Рисунок 4.3 – Архітектура вкладки звуків гри (рисунок виконаний самостійно)

На малюнку 4.3 відкрито основні звуки для рівню із навчанням у грі. Як було описано вище, музика у грі для кожного рівня розбита на окремі доріжки, які вмикаються залежно від інтенсивності небезпеки для головного героя. Аналогічним чином наявні та розташовані звуки та музика для кожного рівня.

Таке рішення було важливим, що це дозволяє швидко знаходити та модифікувати потрібний файл, а також дозволяє легше перевіряти та відстежувати помилки у звуковому супроводженні, наявних звукових ефектах та способі їх програвання всередині ігрового двигуна, оскільки, як було показано вище, файли для кожного рівня знаходяться у відповідній теці та мають чітке найменування за основними конвенціями.

Стосовно ігрового супроводження гри, важливим є те, що ігровий двигун надає можливість керувати всіма параметрами звуку безпосередньо під час виконання. На кожному рівні в нас програється відповідна звукова доріжка із фоновою музикою, яка змінюється залежно від рівня небезпеки головному героєві, але, одночасно із цим, завдяки цим можливостям двигуна можливо керувати гучністю та скільки разів програється музика. Гучність є особливо важливою опцію у цій реалізації звукового супроводу, оскільки налаштування

гучності в головному меню впливає на відповідний коефіцієнт, який потім використовується для керування гучністю усіх звуків та музики всередині гри.

Для наочності пропонуємо навести наступний код:

```
ct.sound.spawn('Menu', {loop: true, volume: 1 * this.soundCoeff})
```

Вищенаведений рядок коду відповідає за запуск фонової музики всередині головного меню. Як можна побачити, окрім назви музикальної доріжки, в якості другого аргументу до методу, було передано об’єкт із полями, що відповідають за зациклення та гучність звуку відповідно.

Окрім цього, тут наявний параметр із коефіцієнтом гучності, який керується користувачем із меню налаштувань. Зазначимо, що налаштування гучності мають відсоткове відображення від 0 до 1, звуковий коефіцієнт також значення від 0 до 1.

Ще одним важливим елементом дизайну гри є користувацький інтерфейс, оскільки він має відгукуватися на дії гравця та допомагати йому стежити за станом головного героя та взаємодіяти із світом гри. Особливо важливими у даному контексті є смуги здоров’я та мани, які завжди присутні на екрані гри та допомагають гравцеві бачити стан магії та здоров’я героя.

Через певні лімітації ігрового двигуна, а саме неможливість створити динамічну шкалу здоров’я виключно внутрішніми можливостями через код, було прийнято рішення створити окремі текстури для відсоткового відображення кожної зі смуг (див. рис. 4.4).

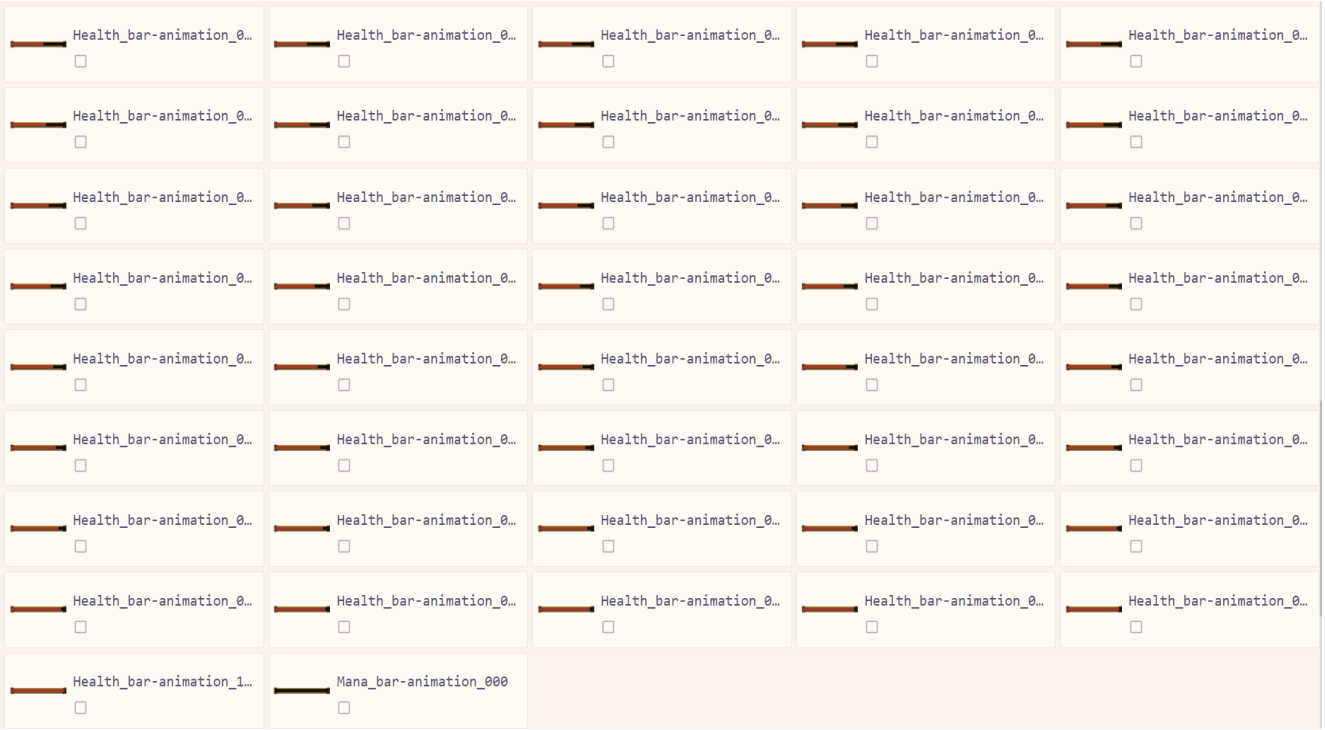


Рисунок 4.4 – Текстури смуги здоров’я (рисунок виконаний самостійно)

Зазначимо, що під час проектування було розглянуто також можливість створити шкалу, заповнюючи її червоним кольором в залежності від здоров’я гравця. Але такий підхід дуже погано впливає на швидкодію гри, оскільки замість перерендерингу однієї текстури, двигун мав би створювати багато об’єктів на екрані. Саме тому було прийнято рішення про використання підходу із використанням багатьох текстур.

Варто розуміти, що з точки зору ігрового дизайну, смуга здоров’я є важливим елементом інтерфейсу, оскільки через неї гравець може побачити скільки здоров’я лишилося в головного героя, що дозволить йому прорахувати власні зусилля перед боєм із ворогами, а також на початкових етапах гри зрозуміти, що шкодить головному герою, а що може навпаки зцілити його. З точки зору дизайну, важливо було продумати, як гравець може взаємодіяти із полосами здоров’я та яким чином ця смуга відображатиметься.

Також варто сказати, що різні вороги наносять різний відсоток шкоди головному героєві. Це було зроблено для того, щоб зробити ігровий процес більш різноманітним, інакше якби кожен ворог наносив однакову шкоду – гравець

перестав би відчувати прогрес при перемозі, а також загальна складність гри помітно би впала. Для наочності взаємодії ворогів із головним героєм пропонуємо переглянути код взаємодії із марою, що є найпершими ворогом, якого зустрічає гравець, під час навчання.

```
else {

    other.health -= 5;
    ct.camera.shake = 0.5;
    ct.camera.shakeDecay = 0;
    other.x -= 100;
    other.tex = 'damaged';
    ct.u.wait(1000).then(() => {
        ct.camera.shakeDecay = 1;
    })
}
```

Як можна побачити з коду, у випадку зіткнення із марою, коли вона атакує головного героя, здоров’я героя знижується на п’ять одиниць, а також героя відкидає назад і вмикає анімацію отримання шкоди. Такий підхід часто використовується у іграх жанру metroidvania.

З точки зору ігрових механік та дизайну – це має сенс, бо одночасно не дає марі декілька разів швидко атакувати головного героя. Показує гравцеві, що герой отримав шкоду та, одночасно із цим, збільшує складність, бо у деяких випадках, це може призвести до падіння з платформи та смерті головного героя. Це спонукає гравця бути обережним коли вороги поряд, та докладно продумувати стратегію перед боєм, особливо у випадках, коли гравець може впасти із платформи та померти.

Так само з точки зору ігрового дизайну, є спонукання гравця досліджувати ігровий світ та шукати картки, які розкидано по ігровим рівня (див. рис. 4.5).

Рисунок 4.5 – Герой поряд з карткою (рисунок виконаний самостійно)

Гравцеві часто потрібно приділяти увагу, щоб знайти картки. Під час навчання герой вчиться користуватися картками для виконання магічних здібностей. З точки зору дизайну, це спонукає гравця не лише проходити основний сюжет, а ретельно досліджувати кожну локацію в пошуку карток, оскільки кожна нова відкрита картка може полегшити проходження основної сюжетної лінії. Зазначимо також, що серед рівнів є певні локації, до яких гравець може дістатися лише після знаходження певних карток, оскільки вони можуть надавати героєві певні поліпшення, що дозволять дістатися раніше недоступних частин рівня.

Загалом, дослідження світу гри є важливим для усіх ігор metroidvania, але саме у нашій грі це зроблено через систему карток. Повертаючись до смуг здоров’я та мани, кожна картка так само використовує ману гравця. Залежно від картки та закляття, що використовується, витрати мани та здоров’я можуть бути більшими чи меншими. Це також спонукає гравця прораховувати свої дії та планувати, чи краще використати меч для битви, чи він може собі дозволити витратити ману на картки із магією.

Дуже важливою складовою нашого програмного продукту є карткова міні-гра. Основні механіки вже було описано вище, але з точки зору дизайну, варто наголосити на значному елементі реіграбельності. Досягається вона тим, що кожного разу, коли гравець обирає фракцію, йому надаються випадковим чином картки із існуючого списку для даної фракції(див рис. 4.6).

Рисунок 4.6 – Стартові картки гравця після першого ходу (рисунок виконаний самостійно)

Як можна побачити на рисунку, гравець має п’ять карток у руці та одну картку на столу. Оскільки гравець обрав картки із колоди багатирів, то усі картки в його руці відносяться саме до цієї фракції. Також зазначимо, що усі картки різні, серед них немає повторів. Це дозволяє гравцеві багато разів проходити цю міні-гру, тому що кожного разу на старті гравець матиме різні картки із різними стартовими характеристиками.

Важливою складовою міні-гри – є система зміни сезонів, що суттєво впливає на ігровий процес, оскільки навіть при однакових стартових картках, гравець може розіграти їх у різний час, а також під дією різних сезонів, може значно вплинути на фінальний рахунок між гравцем та опонентом.

Окремо зазначимо і про штучний інтелект супротивників гравця, який також включає в себе елементи випадковості, в результаті чого опонент кожного разу може по-різному реагувати на дії гравця та розігрувати різні картки. Одночасно із цим, ігрові фракції для міні-гри, хоча й збалансовані, мають різні картки із різними характеристиками, що також сильно впливає на можливість та бажання гравця грати в карткову міні-гру знову і знову. Також і ворог може кожного разу обирати нову фракцію, це робить кожну карткову партію унікальною.

Окремо зазначимо і про архітектурні рішення, які були прийняті при розробці карткової гри. Для карток було створено окремі теки для кожної із фракцій. Окремі теки є як для текстур карток, так і для їхніх шаблонів (див. рис. 4.7).

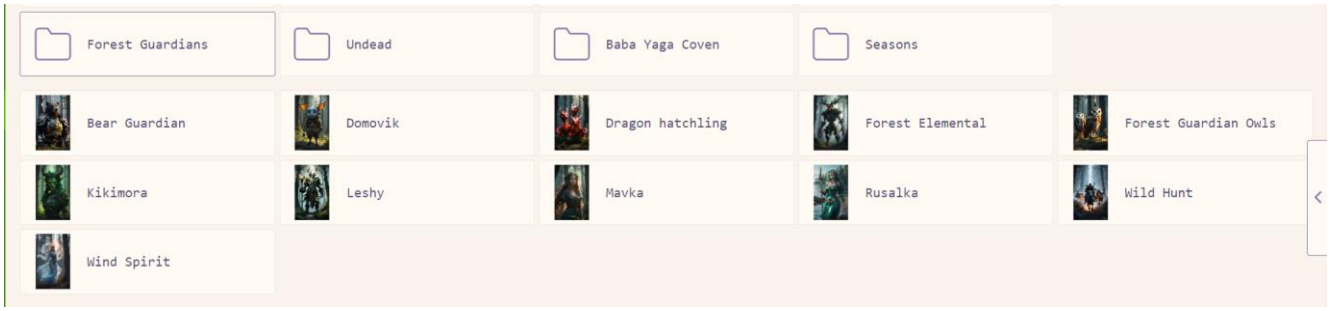


Рисунок 4.7 – Стартові картки гравця після першого ходу (рисунок виконаний самостійно)

На даному рисунку зображена фракція захисників лісу із відповідними текстурами для кожної картки. Аналогічні теки є для шаблонів і для кожної фракції.

5 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Важливою частиною процесу розробки ігрового програмного застосунку, як і будь-якого іншого програмного забезпечення, є процес тестування разом із наступним виправленням знайдених дефектів у програмному коді продукту. Для нашої гри в якості основного методу тестування було використано метод чорної скриньки, мануальне, функціональне та нефункціональне тестування.

Метод чорного ящика – це тестування програмного забезпечення, при якому перевіряється робота програмного продукту без знання її внутрішньої системи роботи. Цей метод тестування є надзвичайно ефективним для ігрових програмних застосунків, оскільки дозволяє протестувати систему з точки зору кінцевого гравця, що гратиме в гру. Таким чином, цей метод тестування дозволяє виявити дефекти у програмному коді, із якими можуть стикнутися гравці (тобто фінальні користувачі цього програмного продукту [10].

Було протестовано усі основні частини програмного застосунку: інтерфейс користувача, анімації, звукове супроводження, взаємодія користувача з персонажами та об’єктами у ігровому світу, ігровий процес, а також окремо логіку карткової міні-гри. Тестування проводилося на усіх цільових платформах та на різних розширеннях екрану, це дало змогу переконатися, що гра працюватиме однаково добре та не матиме дефектів коду для кожної платформи.

Кажучи про ігровий інтерфейс, для його тестування було перевірено основні частини інтерфейсу та усі пункти меню на кожній з цільових платформи. Тестування окремо було проведено на великих та маленьких розмірах екрану. Це було необхідним рішенням для переконання, що ігровий інтерфейс на усіх платформах залишається однаковим, цілісним та послідовним, не містить дефектів, а також інтерфейс та пункти меню відображаються однаково незалежно від розміру екрану та ігрової платформи, на якій було запущено гру.

Тестування анімацій було необхідним, так як це є важливою складовою нашого програмного продукту. Оскільки гра проходить у двовимірному просторі, анімації дозволяють гравцеві зрозуміти, що відбувається зараз на екрані та яку

дію виконує головний герой і його супротивники. Таким чином, анімації є ключовими, щоб впевнитися у коректному відображенні, плавній та правильній графічній репрезентації того, що відбувається на екрані гравця.

Для тестування було проведено ряд дій всередині гри, під час яких програються різні анімації. Завдяки обраному методу можна впевнитися, що перехід між ними є коректним, а також те, що усі анімації є достатньо плавними, при кожній дії програється відповідна анімація, яку гравець очікуватиме побачити. Наприклад, при натисканні на кнопку атаки, гравець очікує побачити анімацію як головний герой атакує ворога, і тестування дозволяє впевнитися, що анімація програється та відображається правильно, та відповідає очікуванню кінцевого користувача програмного продукту.

Тестування звукового супроводу гри є важливим з точки зору атмосфери та налаштувань звуку всередині гри для кожного користувача. Як зазначалося вище, за гучність композицій всередині гри відповідає відповідний параметр, що задається в налаштуваннях гри. Саме тому при тестуванні звукового супроводження важливим було переконатися у тому, що відповідні параметри із звукового налаштування правильно впливають на гучність гри загалом та кожної музикальної композиції окремо.

Окрім цього, для тестування звукового супроводження, як і для тестування анімацій, важливим є перевірка того, чи правильно програється звук та чи програється він у відповідний час під час ігрового процесу. Наприклад, кожна фонова тема має програватися на відповідному рівні та переставати програватися після його завершення. Гравець очікуватиме зміни у музиці під час битви із супротивником, а також залежно від інтенсивності битви. Це теж є об’єктом тестування звукового супроводу гри.

Стосовно взаємодії гравця із навколишнім середовищем, це є важливою механікою ігрового процесу та робить ігровий світ більш інтерактивним, вслід чого одночасно є більш правдоподібним для гравця з точки зору ігрового дизайну. Як було зазначено вище, на етапі навчання, гравця вчать основним ігровим механікам, в тому числі й вмінню взаємодіяти з навколишнім світом. Тому для

проведення тестування необхідно пересвідчитись, що усі взаємодії гравця з ігровим світом працюють як це очікується. Наприклад, коли гравець підбирає картку в ігровому світі, вона має з’явитися в інвентарі героя та він може її використати під час ігрового процесу. Це необхідно протестувати для кожної карти, що знаходиться в ігровому просторі, що є складним, але можливим, з точки зору тестувальника.

Також важливим з точки зору взаємодії гравця є можливість руйнувати певні предмети ігрового світу, наприклад рубити дерево, що стає на шляху. Ця механіка є важливою, оскільки від її коректності залежить можливість гравця пройти на інший рівень на певних етапах ігрового процесу. На додачу до цього, останньою важливою складовою, що входить у тестування взаємодії, є можливість переходити між локаціями у грі. Так, для переходу між домом головного героя та селищем, де він живе, гравцеві необхідно натиснути відповідну кнопку. Під час тестування важливо переконатися, що натиснення оброблюється як слід ігровим двигуном та працює однаково на різних ігрових платформах.

Останньою складовою тестування ігрового програмного застосунку є тестування карткової міні-гри «Відьмориця». Ця міні-гра, як було описано вище, є важливою складовою ігрового дизайну, що робить ігровий процес більш різноманітним. Через великий об’єм цієї міні-гри було прийнято рішення тестувати її окремо від інших складових ігрового застосунку. Було проведено тестування основних ігрових механік, таких як вибір карток, бойова система міні-гри, а також ігровий інтерфейс та звукове супроводження. Проводилося таким самим методом чорної скриньки для виявлення проблем та дефектів. Було також окремо проаналізовано логіку на її відповідність заявленим правилам гри та очікуванням гравців.

6 ВПРОВАДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Процес впровадження нашого ігрового застосунку умовно можна поділити на дві основні частини: ведення соціальних мереж та взаємодію із потенційними гравцями під час процесу розробки гри, а також подальшу публікацію фінальної версії гри на ігрових майданчиках. Тим самим надаючи можливість гравцям придбати та завантажити наш фінальний ігровий продукт.

Кажучи про працю в соціальних мережах та взаємодію із аудиторією, зазначимо, що у процесі розробки гри було створено сторінки для гри у основних соціальних мережах: Instagram, Telegram, YouTube тощо. Через ці соціальні мережі проходила основна взаємодія із потенційною аудиторію гравців.

У процесі розробки ігрового застосунку у соціальних мережах активно публікувалися щоденники розробки програмного продукту. Такий формат є поширеним в ігровій індустрії, він дозволяє аудиторії бачити над чим йде праця у конкретний період часу та дозволяє підтримувати інтерес до програмного продукту серед потенційних покупців. Наприклад, у соціальній мережі Telegram було створено канал, де публікувалися пости із процесом розробки.

Важливим було створення каналу у мережі YouTube, де публікувалися короткі ролики із ігровим процесом та сцени з гри (див. рис. 6.1).

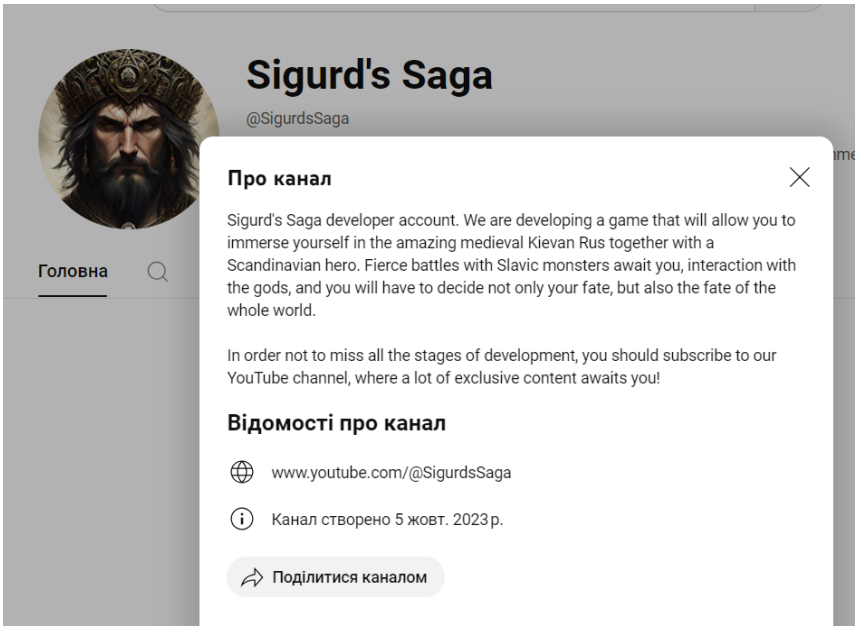


Рисунок 6.1 – Опис каналу у мережі YouTube (рисунок виконаний самостійно)

Такий підхід дозволив працювати з аудиторією та реагувати на думки потенційних гравців, в тому числі виправляючи критику та відповідаючи на запитання гравців.

Другим важливим етапом впровадження гри була її публікація на популярних майданчиках. Для початку було обрано найпопулярнішу платформу, а саме Steam. Такий вибір було зроблено оскільки саме цей майданчик має найбільшу аудиторію гравців у світі і є найпопулярнішою на даний момент платформою.

Зазначимо, хоча основна платформа для ігор на цій платформі – це персональні комп'ютері під управлінням різних операційних систем, але, при цьому, Steam має власну ігрову консоль, що потенційно може дати можливість вийти на консольний ринок. Ця платформа має гарні інструменти для збору статистичних даних, які допомагають вдосконалити програмний продукт та зробити його привабливішим для гравців.

Окрім цього, потенційно у майбутньому планується випуск гри на усіх популярних ігровий майданчиках, в тому числі для мобільних пристроїв. Основними такими на даний момент є Play Market для платформи Android та App Store для платформи IOS. Але рішення про відтермінування публікації гри на цих платформах було прийнято із метою того, щоб мати можливість допрацювати ігровий програмний продукт.

Планується збирати статистику та відгуки гравців на ігровий програмний продукт на найпопулярнішій платформі Steam, щоб виправити недоліки та випустити оновлену версію гри із урахуванням критики та аналітики майданчику на усіх платформах. Ц зробить наш програмний продукт значно привабливішим для фінального користувача.

ВИСНОВКИ

Внаслідок аналізу предметної галузі було спроєктовано дизайн та основний функціонал програмного застосунку, діаграми майбутньої програмної системи, а також необхідний зміст програмного продукту, який розроблено з урахуванням головних закономірностей жанру, але, при цьому, позбавлений основних проблем та недоліків, що характерні для ігор у цьому жанрі.

Важливим було проаналізувати економічну складову цього жанру, зазначаючи основні характеристики продаж ігор та певні спільні риси для ігор у цьому жанрі, що позитивно або негативно можуть вплинути на успіхи продажу ігор.

Спроєктовано та розроблено основні алгоритми і механіки, які можуть якісно відрізняти підготовлений ігровий застосунок від інших ігор у цьому жанрі, а також алгоритми, які можуть використовуватися у коді програмного застосунку під час його розробки та використання. Було проведено роботу над оптимізацією та покращенням коду програмного продукту, який має відповідати основним принципам чистого коду та чистої архітектури.

Проаналізовано основні закономірності, жанрові формули і проблеми ігор у вищезазначеному жанрі. В результаті, спроєктовано та розроблено ігровий програмний застосунок із елементами карткової гри у жанрі метроїдванії, який описано в роботі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Марчук Г.В. Механіки та дизайн рівнів в жанрах «metroidvania» та «souls-like». – Житомир: Державний університет «Житомирська політехніка», 2023. – 92 с.

2. Global Gaming Market Size, Share & Trends Analysis Report [Електронний ресурс] – URL: https://www.linkedin.com/pulse/global-gaming-market-size-share-trends-analysis-report-ashley-hancock-dbi3f/ (дата звернення: 06.05.2024)

3. Answers HQ [Електронний ресурс] – URL: https://answers.ea.com/t5/Drugie-igry-serii-FIFA/Eta-igra-Pay-to-Win-zaplati-chtoby-pobedit/td-p/6544552 (дата звернення: 20.05.2024)

4. Xiaofang Zhong. Measuring the effect of game updates on player engagement. – Нанкін: Нанкінський Університет науки та технології , 2022. – 17 с.

5. LeBlanc M. Mechanics, Dynamics, Aesthetics: A Formal Approach to Game Design. – Іллінойс: Північно-Західний університет, 2004. – 70 с.

6. Chickensoft games [Електронний ресурс] – URL: https://chickensoft.games/blog/game-architecture/ (дата звернення: 27.05.2024)

7. Ct.js documentation [Електронний ресурс] – URL: https://docs.ctjs.rocks/content-subsystem.html (дата звернення: 27.05.2024)

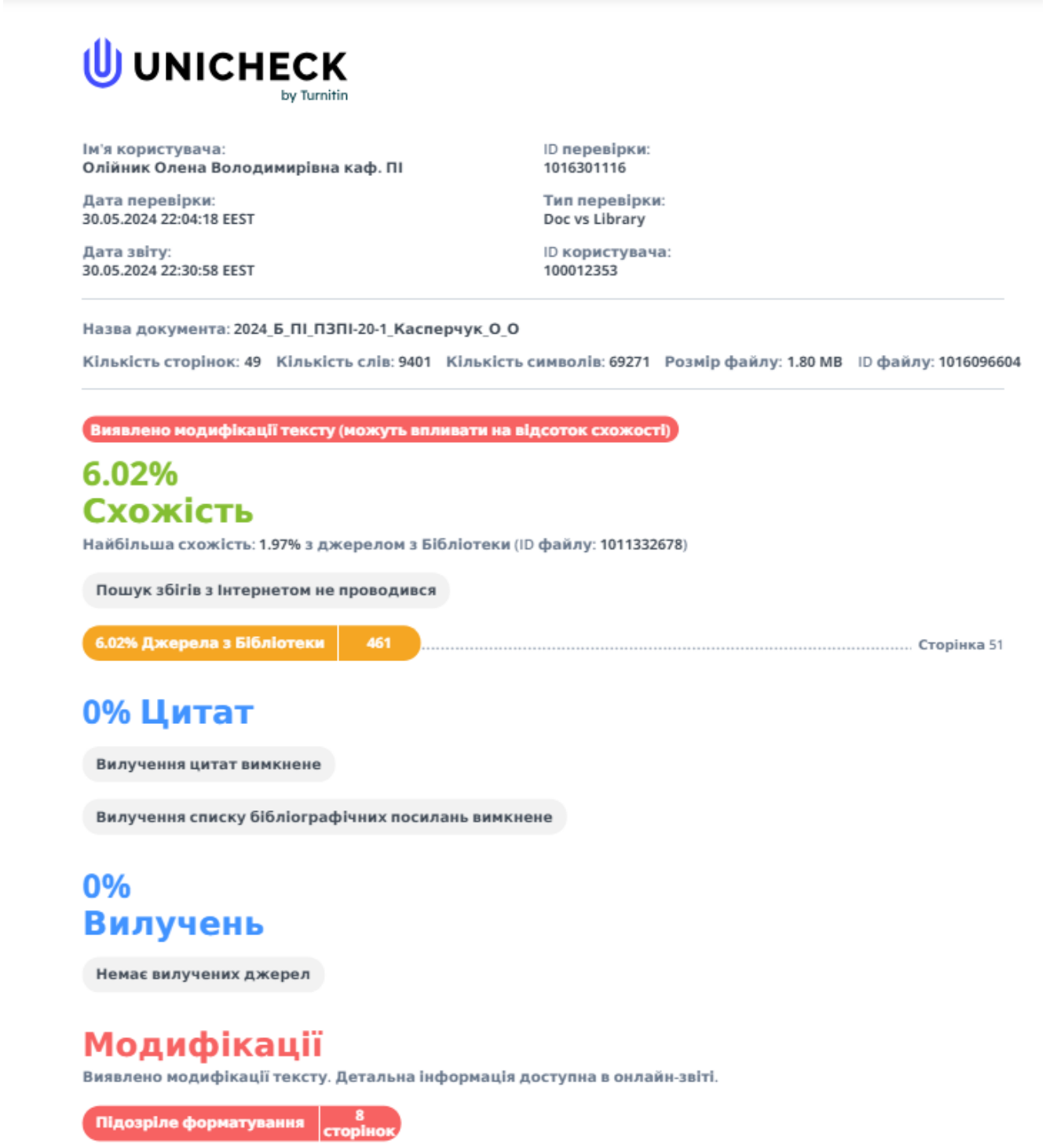
8. Nielsen Norman Group [Електронний ресурс] – URL: https://www.nngroup.com/articles/ten-usability-heuristics (дата звернення: 27.05.2024)

9. Мартін Роберт. Чистий код. Створення, аналіз та рефакторинг. – Київ: Фабула, 2019. – 416 с

10. Крепич, С. Я. Якість програмного забезпечення та тестування : базовий курс : навч. посіб. / С. Я. Крепич, І. Я. Співак. - Тернопіль : ФОП Паляниця В. А., 2020. - 479 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:
1016301116

Дата перевірки:
30.05.2024 22:04:18 EEST

Тип перевірки:
Doc vs Library

Дата звіту:
30.05.2024 22:30:58 EEST

ID користувача:
100012353

Назва документа: 2024_Б_ПІ_ПЗПІ-20-1_Касперчук_О_О

Кількість сторінок: 49 Кількість слів: 9401 Кількість символів: 69271 Розмір файлу: 1.80 MB ID файлу: 1016096604

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

6.02%
Схожість

Найбільша схожість: 1.97% з джерелом з Бібліотеки (ID файлу: 1011332678)

Пошук збігів з Інтернетом не проводився

6.02% Джерела з Бібліотеки 461 ..... Сторінка 51

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.

Підозріле форматування 8 сторінок

ДОДАТОК Б

Слайди презентації

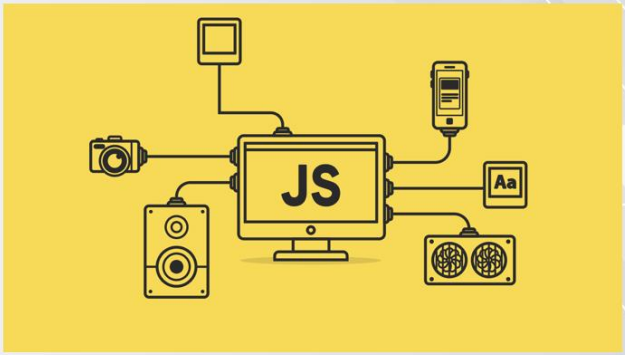
The Saga of Sigurd

Виконав:

студент 4 курсу, групи ПЗПІ-20-1

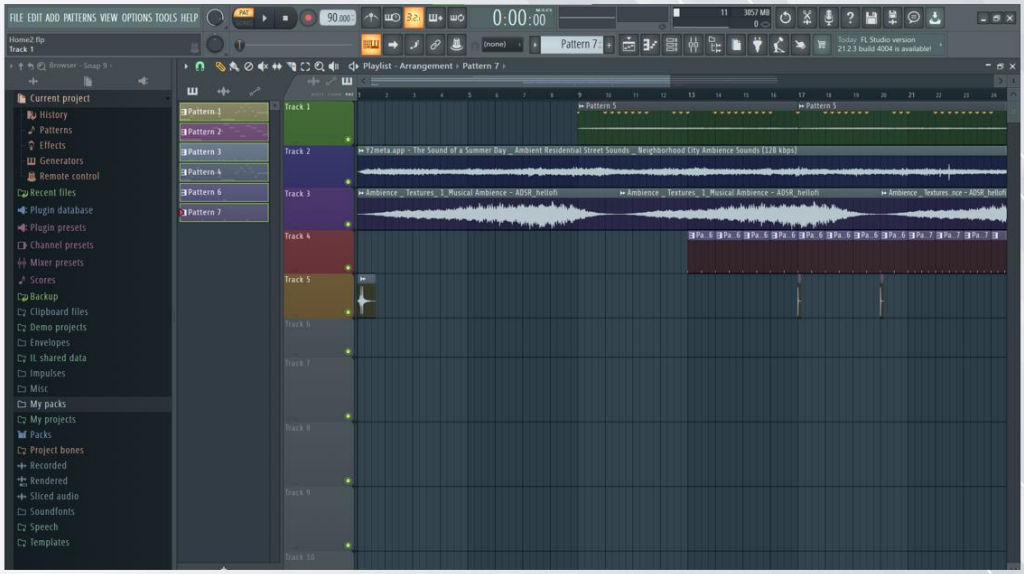
Касперчук О.О.

Вибір мови програмування



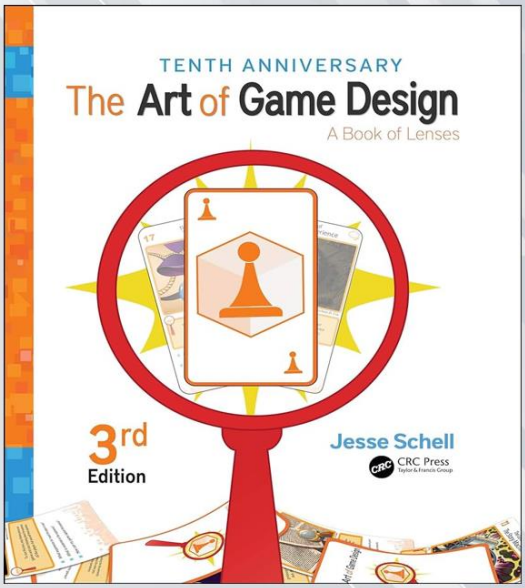
Вибір ігрового двигуна

Процес написання звукового супроводу



Гейм-дизайн

- Використали «лінзи» Джессі Шелла
- Детально проаналізували інші ігри жанру
- Використали власний підхід до дизайну
- Основна мета – зробити гру цікавою

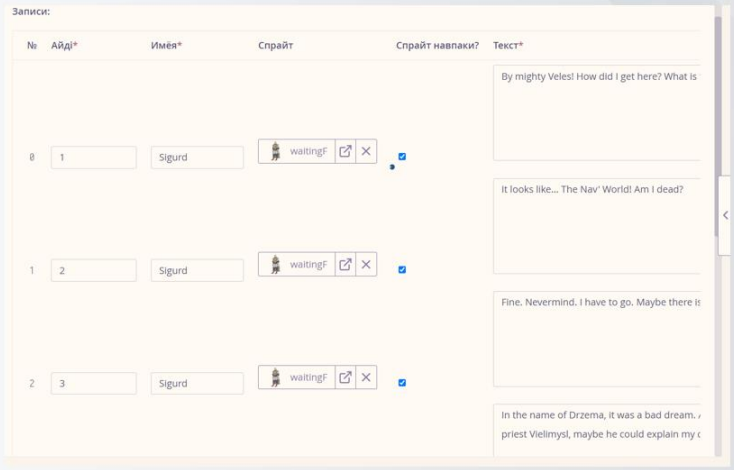


Міні-ігри

Під час проектування ми надихалися такими проектами, як Gwent та Heartstone

Але наша карткова гра унікальна та має власні ігрові механіки

Бази даних та структура зберігання об'єктів



{.json}

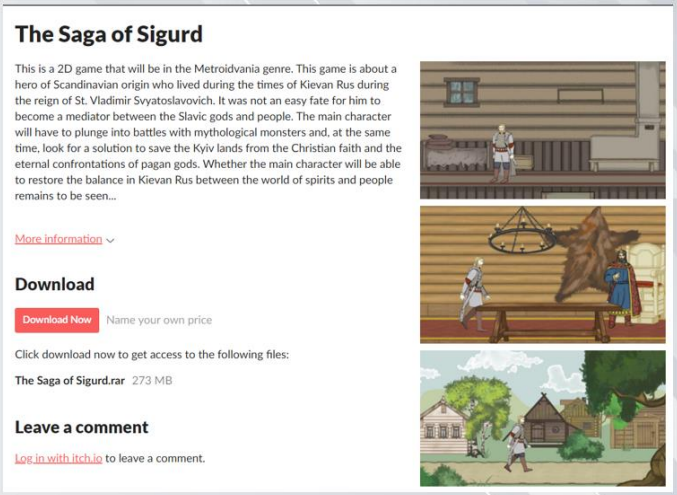
Соціальні мережі

Акаунти в

- **Instagram**
- **Telegram**
- **Twitter**
- **Youtube**

Публікація гри

- Гра вже опублікована на Itch.io
- В планах Steam, EGS
- Мобільні Play Market і App Store



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 найбльш 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).

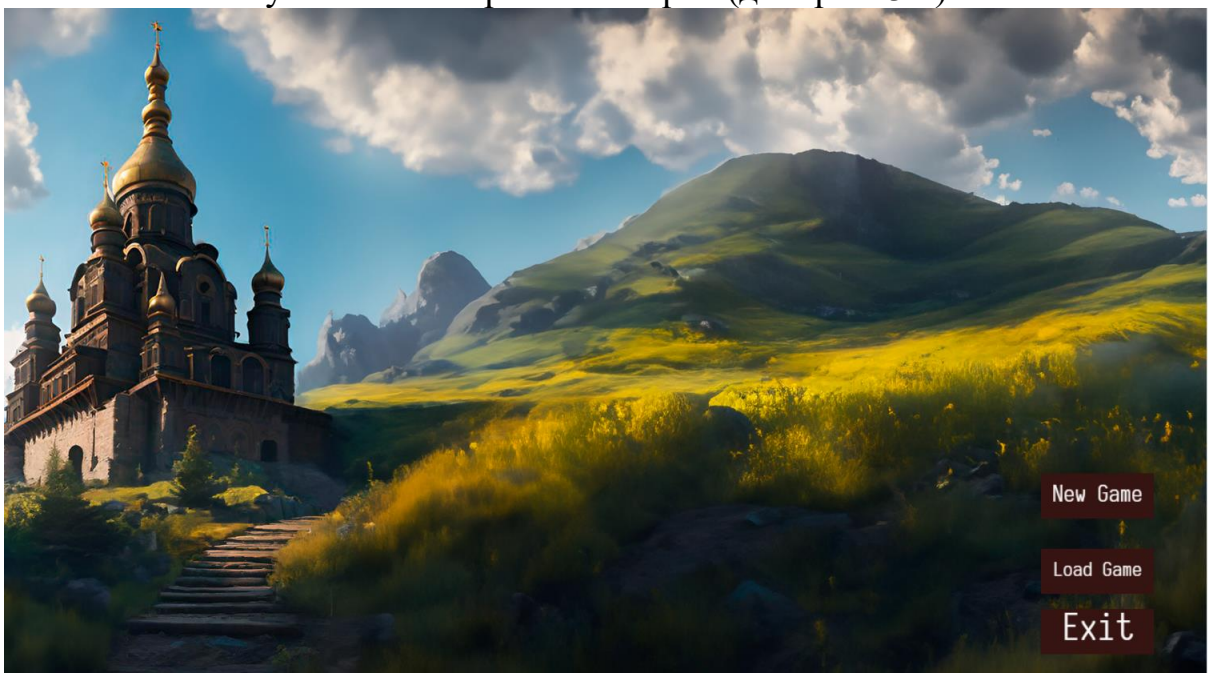


Рисунок 3.1 - Головне меню гри

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).

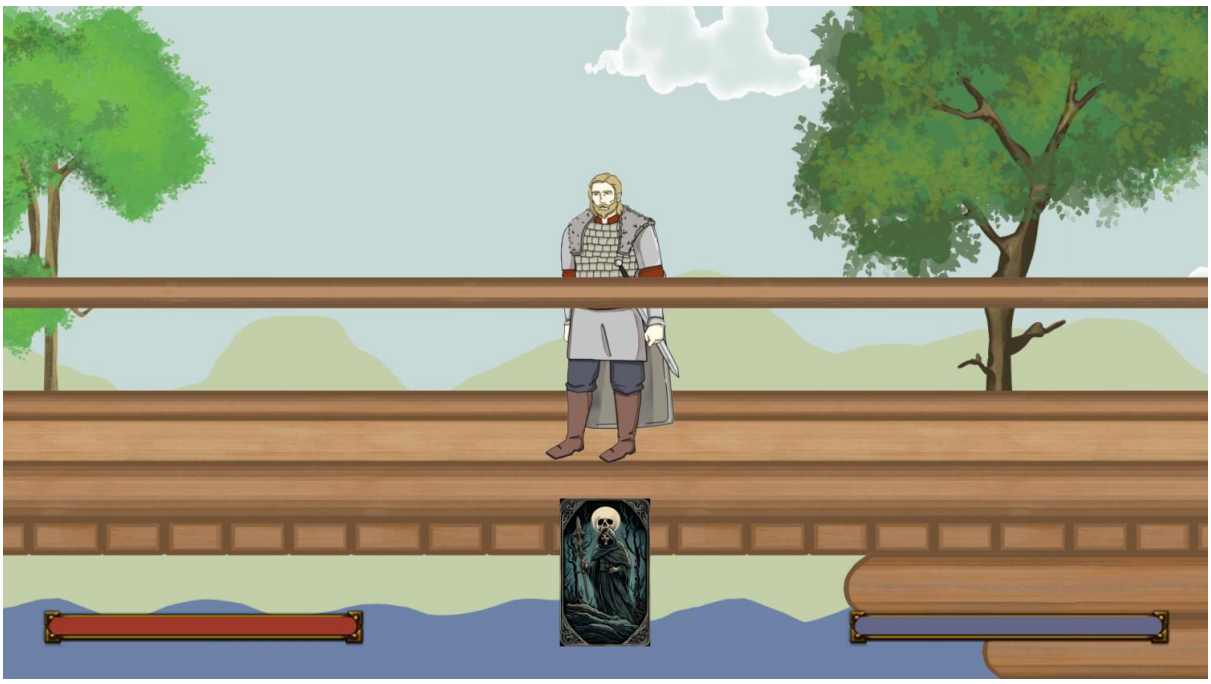


Рисунок 3.6 - Локація з мостом

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

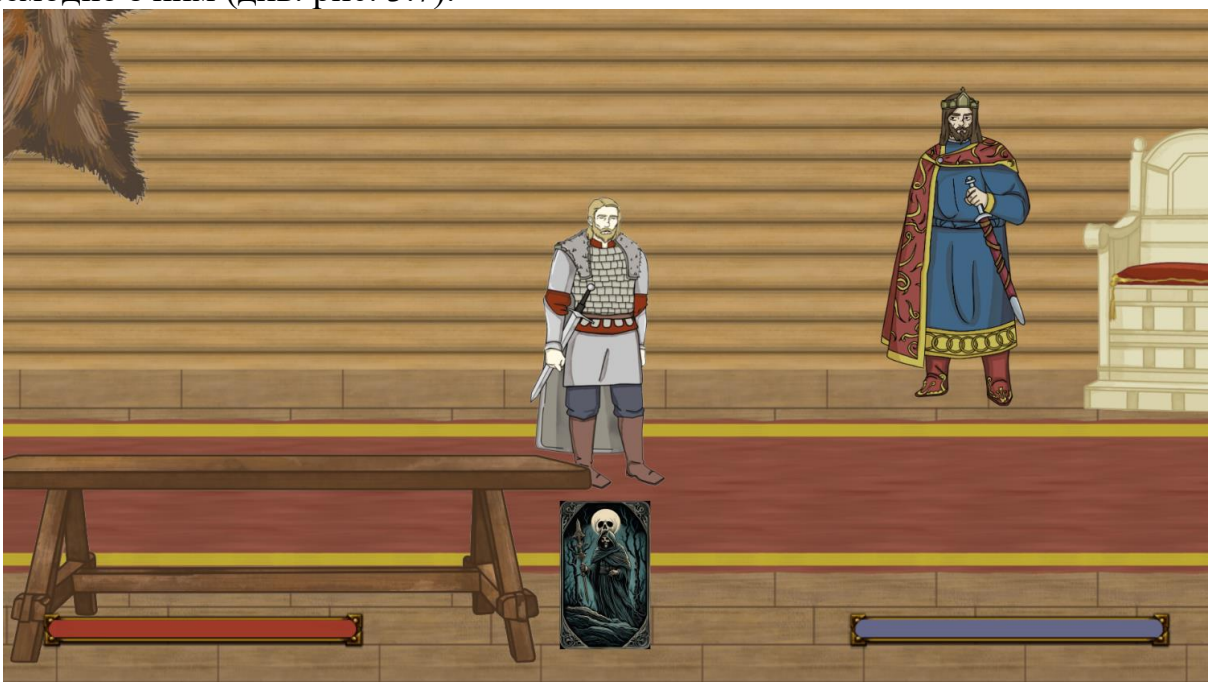


Рисунок 3.7 - Локація з помешканням князя

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).

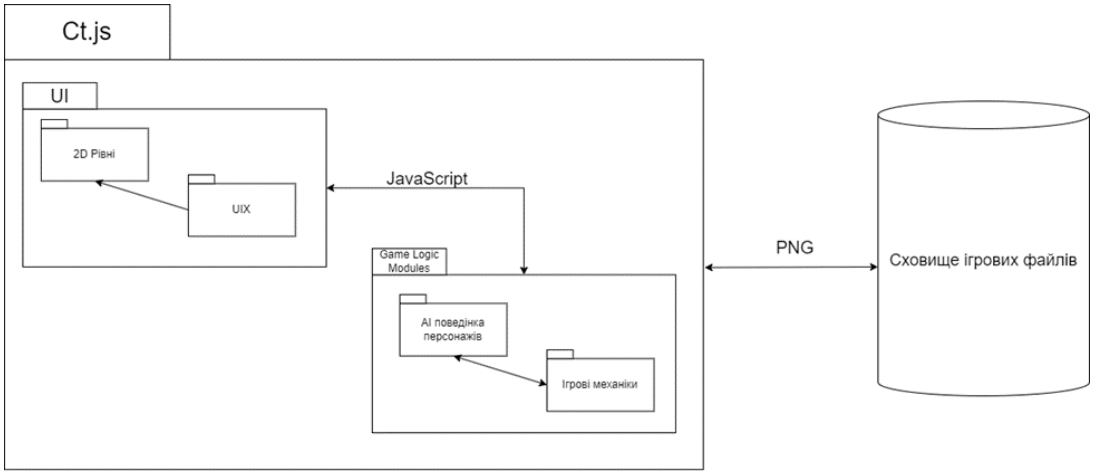


Рисунок 3.10 - Діаграма, яка демонструє комунікаційний протокол

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).

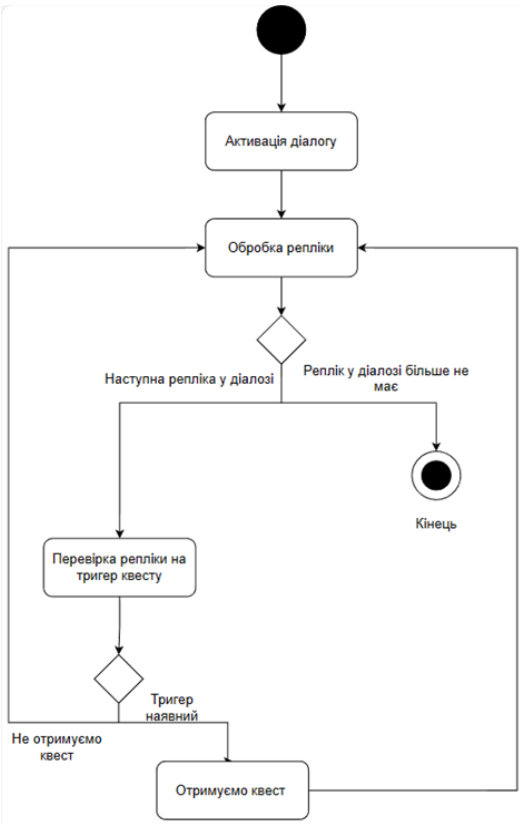


Рисунок 3.10 - Діаграма, яка демонструє діалогову функцію

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).

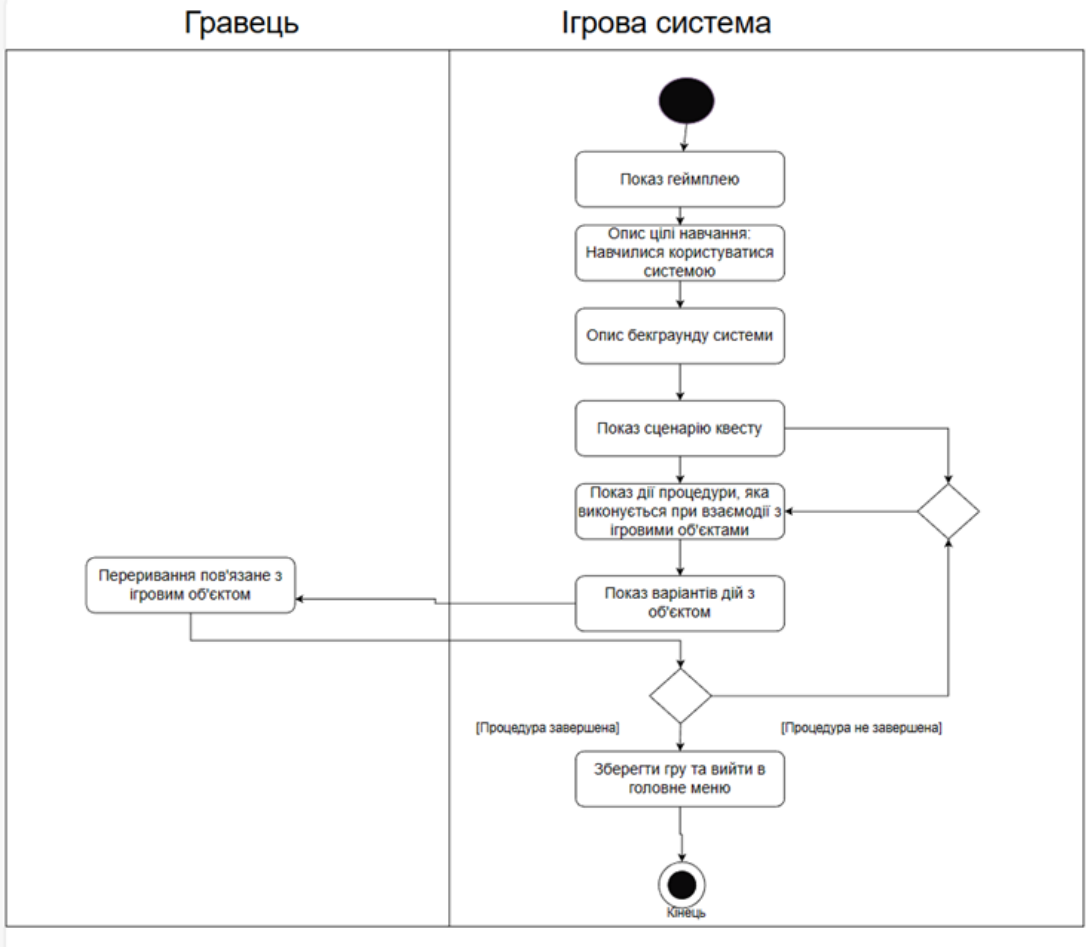


Рисунок 3.11 - Діаграма, яка демонструє взаємодію гравця із системою

6. Припущення й залежності

До припущень та залежностей можна віднести такі ситуації:

1. Користувачі будуть використовувати гру на персональному комп’ютері, мобільному пристрої або у Web браузері.
2. Користувачі будуть керувати головним героєм шляхом натискання відповідних клавіш на клавіатурі.
3. У певний момент часу користувачі захочуть вийти з гри та перейти до свого робочого екрану.
4. Користувачам буде необхідне графічне відображення кожної дії, яку вони зробили.
5. 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

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.

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. Безпека

Стосовно безпеки, у нашій грі імплементовано основні принципи, які защищають 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.

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.

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.

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 кодів програми

```
      this.maxSpeed = 10;
      this.jumpSpeed = 30;
      this.wallJumpSpeed = 10;
      this.wallJumpSideSpeed = 15;
      this.wallSlideSpeed =  5;

      this.gravity = 1;
      this.gravityDir = 90;


      ct.camera.follow = this;

      console.log( "Animation Speed: " + this.animationSpeed)

      // Acceleration / Deceleration values for horizontal speed
      this.groundAccDec = 1.5;
      this.airAccDec = 0.3; // Give little control over character while in
air
      this.iceAccDec = 0.2; // Give even lesser control while on ice

      this.doubleJumped = false;

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

      this.attackMode = false;
      this.dialogueMode = false;
      this.isAnimationReversed = false;
      this.loop = true;

      this.health = 100
      this.mana = 100;

      // Skip gameplay logic if dying
      if (this.dying) {
          return;
      }

      // Get the surrounding medium
      this.onGround = ct.place.occupied(this, this.x, this.y + 2, 'Solid');
      var onIce = false;
      if (this.onGround) {
          if (ct.place.meet(this, this.x, this.y + 2, 'IceBlock')) {
              onIce = true;
          }
      }

      //   Calculate   acceleration/deceleration   values   based   on   the
surrounding medium
      var accDec = this.groundAccDec;
      if (!this.onGround) {
          accDec = this.airAccDec;
```

```
    } else if (onIce) {
        accDec = this.iceAccDec;
    }

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

    var jumped = false;
    // Regular jumps
    if (ct.actions.Jump.down) {
        if (this.onGround) {
            this.vspeed = -this.jumpSpeed;
            this.doubleJumped = false;
            jumped = true;
        }
    }
    if (ct.actions.Jump.pressed && !jumped) {
        if (!this.onGround) {
            this.tex = 'jumpingF'
            this.isAnimationReversed = true;
            this.attackMode = false;

            // Wall jump | Right
            if (ct.place.occupied(this, this.x - 2, this.y, 'Solid')) {
                this.vspeed = -this.wallJumpSpeed;
                this.hspeed = this.wallJumpSideSpeed;
                this.doubleJumped = false;
                jumped = true;
            }
            else
            // Wall jump | Left
            if (ct.place.occupied(this, this.x + 2, this.y, 'Solid')) {
                this.vspeed = -this.wallJumpSpeed;
                this.hspeed = -this.wallJumpSideSpeed;
                this.doubleJumped = false;
                jumped = true;
            } else if (!this.doubleJumped) {
                // Double jumps
                this.doubleJumped = true;
                this.vspeed = -this.jumpSpeed;
            }
        }
    }

    // Perform the actual movement. If the hero is facing an obstacle on
one of the
    // sides, stop movement along this axis
    var collisions = this.moveContinuousByAxes('Solid');
    if (collisions) {
        if (collisions.x) {
            this.tex = 'walkingF'
            this.isAnimationReversed = true;
```

```
            this.attackMode = false;
            this.hspeed = 0;
            // Limit falling speed while sliding on walls
            this.vspeed = Math.min(this.vspeed, this.wallSlideSpeed);
        }
        if (collisions.y) {
            this.vspeed = 0;
        }
    }




    // Animations
    if (ct.actions.Attack.pressed && !this.dialogueMode) {
        let x = this.x;
        let y = this.y;
        this.tex = "fightingF";
        // this.animationSpeed = 1;
        this.loop = true;
        this.play()
        this.x = x;
        this.y = y;
        this.attackMode = true;
        this.isAnimationReversed = true;
        ct.sound.spawn('Sword')
    }
    if (ct.actions.MoveX.pressed || ct.actions.MoveY.pressed) {
        this.tex = 'walkingF'
        this.loop = true;
        this.play();
        this.attackMode = false;
        // this.animationSpeed = 1;
        this.isAnimationReversed = true;
    }

    if    (!ct.actions.MoveX.down    &&    !ct.actions.MoveY.down    &&
!this.attackMode == true) {
        this.tex = 'waitingF'
        // this.animationSpeed = 1;
        this.isAnimationReversed = false;
        this.attackMode = false;
    }

    if    (ct.actions.Card1.down    ||    ct.actions.Card2.down    ||
ct.actions.Card3.down) {
        this.tex = "magickF";
        this.loop = false;
        // this.animationSpeed = 1;
        this.isAnimationReversed = true;
        this.attackMode = true;
    }

    if (ct.actions.Jump.down) {
        this.tex = 'jumpingF'
        // this.animationSpeed = 1.5;
```

```
    this.isAnimationReversed = true;
    this.attackMode = false;
}
```

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

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

if (this.health <= 0) {
    this.kill = true;
    ct.room.destroy();
    ct.rooms.switch('GameOver');
}
```

```
this.dying = this.health <= 0;
ct.camera.follow = null;
// Make a dying animation
ct.tween.add({
    obj: this,
    fields: {
        y: this.y + 400,
        angle: 360,
        alpha: 0
    },
    curve: ct.tween.easeInBack,
    duration: 1000
})
.then(() => {
    // Schedule a jump to the last checkpoint once animation finishes
```

```
    this.x = this.checkpointX;
    this.y = this.checkpointY;
    // Reset dying state and movement speed
    this.vspeed = this.hspeed = 0;
    this.dying = false;
    // Reset drawing settings
    this.angle = 0;
    this.alpha = 1;
    ct.camera.follow = this;
});
```