МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ПОЛІСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет обліку та фінансів
Кафедра комп'ютерних технологій
і моделювання систем

Кваліфікаційна робота
на правах рукопису

Левченко Дмитрій Юрійович

УДК 004.9:004.94

КВАЛІФІКАЦІЙНА РОБОТА

Інформаційно-розважальна система в жанрі RPG з візуалізацією об'єктів в стилі Souls-like

(тема роботи)

122 «Комп'ютерні науки»

(шифр і назва спеціальності)

Подається на здобуття освітнього ступеня бакалавр

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

_____

(підпис, ініціали та прізвище здобувача вищої освіти)

Керівник роботи
Гіваргізов Інвія Геннадійович,
старший викладач КТіМС, к. е. н.

Житомир – 2024

Висновок кафедри

_____

за результатами попереднього захисту:

_____

Протокол засідання кафедри

_____

№ ____ від «_____» _____ 20____ р.

Завідувач кафедри

_____

_____ _____ _____

(науковий ступінь, вчене звання) (підпис) (прізвище, ім’я, по батькові)

«_____» _____ 20__ __ р.

Результати захисту кваліфікаційної роботи

Здобувач вищої освіти _____ захистив (ла)

(прізвище ,ім’я, по батькові)

кваліфікаційну роботу з оцінкою:

сума балів за 100-бальною шкалою _____

за шкалою ECTS _____

за національною шкалою _____

Секретар ЕК

_____ _____ _____

(науковий ступінь, вчене звання) (підпис) (прізвище, ім’я, по батькові)

АНОТАЦІЯ

Левченко Д. Ю. Інформаційно-розважальна система в жанрі RPG з візуалізацією об’єктів в стилі souls-like – Кваліфікаційна робота на правах рукопису.

Кваліфікаційна робота на тему присвячена розробці та реалізації інформаційно-розважальної системи, яка поєднує елементи рольової гри RPG та естетику, характерну для ігор в стилі Souls-like. Метою роботи є створення інтерактивної системи, яка забезпечує захоплюючий ігровий досвід завдяки продуманому дизайну, високоякісній візуалізації та глибокій ігровій механіці.

У процесі дослідження проведено аналіз інформаційних потреб користувачів та визначено ключові вимоги до розроблюваної системи. Було здійснено детальний огляд технічного забезпечення, необхідного для реалізації проекту, та обрано оптимальні технології і платформи для розробки.

Робота також включає моделювання бізнес-процесів та системи в цілому, що дозволило створити детальну архітектуру системи. Особливу увагу приділено проектуванню інтерфейсу користувача, який відповідає сучасним стандартам ергономіки та зручності використання.

Реалізація інформаційно-розважальної системи виконана на базі рушія Unity, що забезпечило високий рівень продуктивності та інтерактивності. У роботі також представлено інструкцію користувача та результати тестування системи, які підтверджують її функціональність та відповідність заявленим вимогам.

Результати даного дослідження можуть бути використані для подальшого розвитку ігор в жанрі RPG та створення високоякісних інформаційно-розважальних систем з акцентом на візуалізацію в стилі Souls-like.

Кваліфікаційна робота на здобуття освітнього ступеня бакалавра за спеціальністю 122 – Комп’ютерні науки. – Поліський національний університет, Житомир, 2024.

Ключові слова: RPG, Souls-like, Unity.

SUMMARY

Levchenko D. Y. Information and entertainment system in the RPG genre with object visualization in the souls-like Style – Qualification work as a manuscript.

The qualification work is dedicated to the development and implementation of an information-entertainment system that combines elements of role-playing games RPG and the aesthetics characteristic of Souls-like games. The aim of the work is to create an interactive system that provides an engaging gaming experience through thoughtful design, high-quality visualization, and deep game mechanics.

During the research process, an analysis of user information needs was conducted, and the key requirements for the system being developed were identified. A detailed review of the technical infrastructure required for project implementation was carried out, and the optimal technologies and platforms for development were selected.

The work also includes the modeling of business processes and the system as a whole, which allowed for the creation of a detailed system architecture. Special attention was given to the design of the user interface, which meets modern standards of ergonomics and usability.

The implementation of the information-entertainment system was carried out using the Unity engine, ensuring a high level of performance and interactivity. The work also presents a user manual and the results of system testing, confirming its functionality and compliance with the stated requirements.

The results of this research can be used for further development of RPG genre games and the creation of high-quality information-entertainment systems with an emphasis on Souls-like visualization.

Qualification work for the Bachelor’s degree in specialty 122 – Computer Science. – Polissia National University, Zhytomyr, 2024.

Keywords: RPG, Souls-like, Unity.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

RPG – рольова гра

ПК – персональний комп’ютер

ВСТУП

Створення комп’ютерних ігор набуває великої важливості в сучасному світі, оскільки значна частина населення віддає перевагу саме гральній активності у своєму дозвіллі. Цей феномен відображає не лише зміни в культурних та соціальних уподобаннях, але й значний вплив цифрових технологій на повсякденне життя людей. Індустрія відеоігор розвивається надзвичайно швидкими темпами, надаючи мільйонам користувачів по всьому світу нові можливості для розваг, навчання та соціалізації.

Використання передових ігрових рушіїв має велике значення, оскільки воно спрощує процес розробки на різних рівнях і дозволяє зосередитися на детальному вдосконаленні ігрового світу. Ігрові рушії, такі як Unity, Unreal Engine та інші, надають розробникам потужні інструменти для створення високоякісного контенту з мінімальними затратами часу та ресурсів. Завдяки цьому розробники можуть реалізувати найсміливіші ідеї, забезпечуючи гравцям захоплюючий та інноваційний ігровий досвід [35].

З огляду на зростаючий попит на ігри та розваги в цифровому середовищі, вивчення ефективних методів використання таких інструментів є актуальним завданням для досягнення успішної та конкурентоспроможної розробки ігрового контенту.

Метою кваліфікаційної роботи є розробка інформаційно-розважальної системи в жанрі RPG з візуалізацією об’єктів в стилі Souls-like.

Предметом дослідження є сукупність засобів та методів, які використовуються для розробки інформаційно-розважальної системи в жанрі RPG з візуалізацією об’єктів в стилі Souls-like.

Об’єктом дослідження є процес розробки базових ігрових механік в жанрі RPG.

Завданнями на кваліфікаційну роботу є:

- моделювання інформаційно-розважальної системи в жанрі RPG з візуалізацією об’єктів в стилі Souls-like

- розробка інтерфейсу для інформаційно-розважальної системи в жанрі RPG з візуалізацією об’єктів в стилі Souls-like
- реалізація інформаційно-розважальної системи в жанрі RPG з візуалізацією об’єктів в стилі Souls-like

Для досягнення поставлених цілей було застосовано наступні методи дослідження: аналітичний метод (використовувався для аналізу предметної області); метод моделювання (застосовувався для аналізу бізнес-процесів з метою побудови інформаційної системи); експериментальний метод (використовувався метод тестування для перевірки відповідності отриманого результату вимогам до інформаційної системи).

Автором було опубліковано статті на двох конференціях за темами «Особливості розробки сучасних комп’ютерних ігор» та «Головоломка (жанр відео ігор)», що відповідають тематиці розробки.

1. Левченко Д. Ю. Особливості розробки сучасних комп’ютерних ігор. Безпека, технології, інновації: нові горизонти : збірник праць учасників міжфакультетської науково-практичної інтернет-конференції здобувачів вищої освіти і молодих вчених, 15 листопада 2023 р. Житомир : Поліський національний університет, 2023. 25 с.

2. Левченко Д. Ю Головоломка (жанр відео ігор). Інформаційні технології та моделювання систем : збірник праць учасників Всеукраїнської науково-практичної конференції здобувачів вищої освіти і молодих вчених, 10 квітня 2024 р. Житомир : Поліський національний університет, 2024. 49 с.

Практичне значення полягає в покращенні інтерактивності та якості ігрового досвіду у цій унікальній галузі розваг. Система автоматизує розробку геймплею, управління візуальними ефектами та забезпечує зручний інтерфейс для гравців, що дозволяє їм повною мірою насолоджуватися ігровим процесом.

Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. Загальний обсяг роботи становить 44 сторінки.

РОЗДІЛ 1 ТЕОРЕТИЧНІ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз інформаційних потреб і визначення предметної області

Аналіз інформаційних потреб та визначення предметної області для створення інформаційно-розважальної системи в жанрі RPG з візуалізацією об’єктів у стилі Souls-like є ключовим етапом, що визначає успішність проекту.

Система візуалізації об’єктів у стилі Souls-like, яка буде використовуватися в даній роботі, базується на характерних рисах ігор серії Souls (Dark Souls, Demon’s Souls, Bloodborne), відомих своїм високим рівнем складності, глибокою атмосферою та унікальним дизайном світу і персонажів. Основні аспекти цього стилю включають деталізоване оточення, темну та похмуру атмосферу, а також специфічний підхід до дизайну ворогів та босів.

Для визначення предметної області необхідно визначити основні інформаційні потреби користувачів та зацікавлених сторін. Гравці потребують захоплюючого геймплею з глибокою сюжетною лінією, реалістичними персонажами та оточенням, а також справедливого та чітко збалансованого виклику. Візуальний стиль повинен сприяти зануренню у гру, забезпечуючи одночасно естетичне задоволення та функціональну чіткість.

Важливою частиною аналізу є вивчення існуючих рішень та підходів в області інформаційно-розважальних систем у жанрі RPG. Це включає дослідження поточних трендів у розробці відеоігор, використання сучасних інструментів та технологій для створення ігор, а також вивчення потреб та очікувань цільової аудиторії. Сучасні ігрові движки, такі як Unreal Engine або Unity, надають широкі можливості для створення складних ігор з високоякісною графікою та реалістичною фізикою, що є критично важливим для реалізації проекту в стилі Souls-like [35].

Таким чином, детальний аналіз інформаційних потреб та предметної області є фундаментом для розробки успішної інформаційно-розважальної системи в жанрі RPG з візуалізацією об’єктів у стилі Souls-like. Це забезпечує

чітке розуміння вимог до гри, необхідних ресурсів та технологій, а також визначає напрямки подальшої розробки та оптимізації проекту.

1.2. Аналіз технічного забезпечення

Серед найпопулярніших ігрових рушіїв варто виділити Unity, Unreal Engine, Godot та CryEngine. Кожен з них має свої переваги та недоліки, які розглянемо нижче. Порівняльна характеристика ігрових рушіїв зображено на Табл. 1.1.

Unreal Engine відомий своїми графічними можливостями та потужністю, що робить його відмінним вибором для AAA-проектів. Однак, складність використання та необхідність знання C++ можуть бути перешкодами для менш досвідчених розробників. Blueprints надають можливість візуального скриптування, але це не завжди замінює повноцінну роботу з кодом [19].

Godot є чудовим вибором для інді-розробників завдяки простоті використання та повністю безкоштовній ліцензії. Хоча його графічні можливості та продуктивність поступаються Unity та Unreal, він забезпечує достатній функціонал для створення якісних ігор. Підтримка кількох мов програмування робить його гнучким та зручним для різних типів проектів [22].

CryEngine забезпечує високу продуктивність та чудові графічні можливості, але складність його використання та менш активна спільнота можуть бути обмеженням для розробників. Він найкраще підходить для проектів, які вимагають найвищої якості графіки, але не є оптимальним вибором для тих, хто потребує інтуїтивних інструментів та широкої підтримки [23].

Unity є оптимальним вибором для розробки завдяки своїй універсальності, потужним інструментам, широким можливостям кастомізації та великій спільноті. Це робить його ідеальним для створення складних та захоплюючих ігор, забезпечуючи при цьому високу продуктивність та якість кінцевого продукту[18].

Unity було обрано через його високу продуктивність, широкі графічні можливості та інтуїтивні інструменти для анімацій та UI. Мова програмування

C# є потужною та зручною для скриптування, що дозволяє створювати складні ігрові механіки. Мультиплатформенність забезпечує доступ до широкої аудиторії, а велика спільнота та бібліотека ресурсів значно полегшують розробку[31].

Висновки до першого розділу

У першому розділі здійснено аналіз інформаційних потреб користувачів та визначено предметну область, в якій функціонуватиме інформаційно-розважальна система. Було також проведено аналіз технічного забезпечення, необхідного для реалізації системи. В результаті цього аналізу визначено основні вимоги до системи, що створюється, та обрано оптимальну технічну базу для її реалізації.

РОЗДІЛ 2 МОДЕЛЮВАННЯ ІНФОРМАЦІЙНО-РОЗВАЖАЛЬНОЇ СИСТЕМИ

2.1. Моделювання бізнес-процесів інформаційно-розважальної системи

Загальний вигляд реалізації продукту зображено на Рис. 1.1.

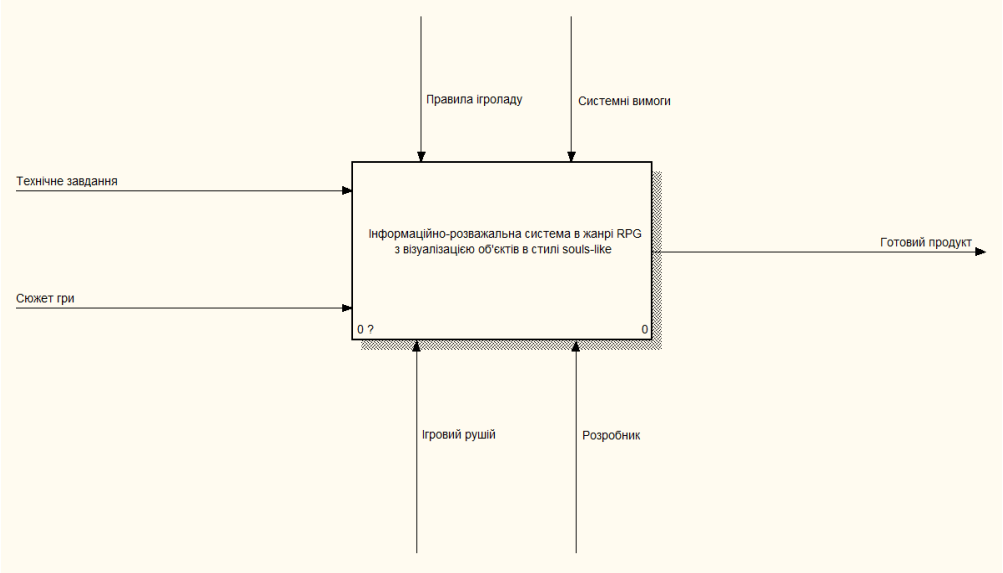

Рис. 1.1 – IDEF0 діаграма реалізації продукту

Система приймає дані у вигляді технічного завдання та сюжету для розробки, і на виході отримаємо готовий продукт. Декомпозицію IDEF0 моделі наведено на Рис 1.2.

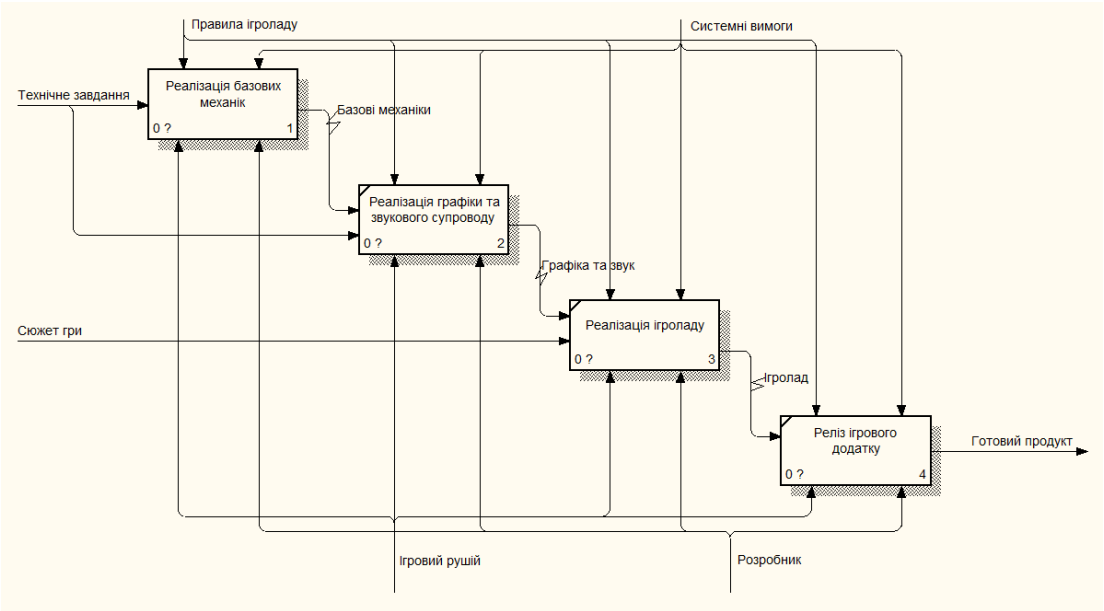

Рис. 1.2 – IDEF0 «Розробка ІРС в жанрі RPG»

На декомпозиції розписано, як саме система працюватиме. Спершу, розробляються базові механіки для гри, без який не можливо зробити перший

грабельний прототип. Далі реалізуються всі елементи, що відповідають за наповнення гри, такі як графіка та звук. Після реалізації всіх побічних елементів створюються більш складні механіки, за допомогою який і формується геймплейна складова. Останнім кроком є випуск готового ігрового продукту. Декомпозицію для блоку «Розробка базових механік» зображено на Рис. 1.3.

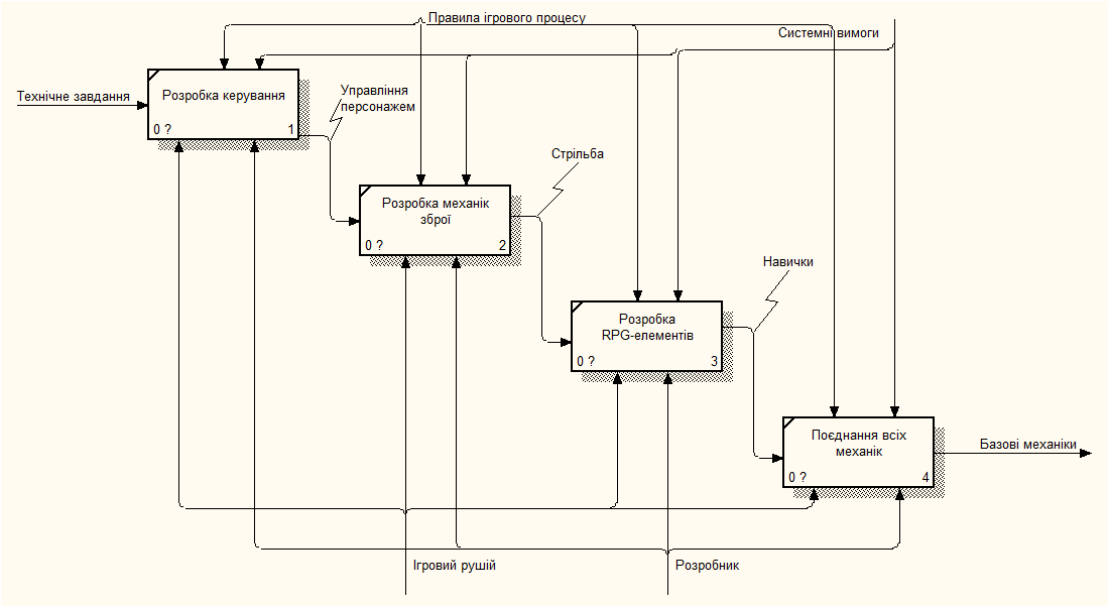

Рис. 1.3 – IDEF0 «Реалізація базових механік»

У цій декомпозиції визначено основні механіки, які повинні бути втілені в грі. Початковий етап передбачає створення скриптів для керування персонажем, що охоплює переміщення, управління камерою, а також скрипти для взаємодії з іншими персонажами та об’єктами. Наступний крок передбачає розробку арсеналу зброї. У процесі розробки рольових елементів, в основному, визначаються характеристики персонажа та система його прокачки. Злагодивши всі ці компоненти, ми отримаємо готового ігрового персонажа із повним комплексом базових механік.

Першим етапом є розробка концепція сюжету, де формуються основні ідеї та концепції для подальшого розвитку гри. Процес створення сюжету зображений на у додатках Рис. 1.4 за допомогою IDEF3.

На основі цієї концепції створюється головна лінія сюжету, що визначає основний напрямок розвитку гравця. Паралельно з цим розробляються побічні сюжетні лінії, які збагачують гру глибиною та різноманіттям у світі гри. Також

створюється перелік квестів, що розширюють можливості гравця та допомагають йому вивчати навколишній світ. Процес створення елементів ігрового оточення зображений у додатках на Рис. 1.5 за допомогою IDEF3.

На першому етапі створюється загальне ігрове оточення, аналізуються концепції для встановлення стилю, тематики та настрою гри. Потім розробляється дизайн персонажів з унікальними характеристиками, історіями та мотиваціями. На наступному етапі визначаються ігрові механіки, включаючи систему прокачки та взаємодію гравця. Далі розробляється дизайн навколишнього середовища, включаючи локації та архітектуру. Останнім етапом створюється звуковий дизайн для підсилення атмосфери гри. Останньою є IDEF3 візуальної частини гри зображено у додатках на Рис. 1.6.

2.2 Моделювання інформаційно-розважальної системи

Для розуміння роботи системи було створено UML-діаграму станів на Рис. 2.1 додатку В.

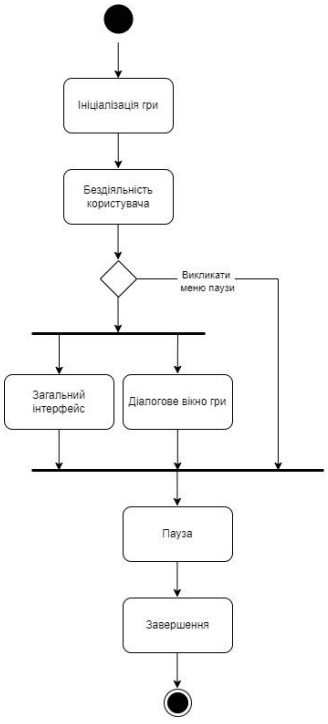

Рис. 2.1 – UML-діаграма станів

На діаграмі станів показано основні стани, через які проходить гра від моменту ініціалізації до завершення. Після ініціалізації гри користувач може перебувати в стані бездіяльності, звідки є можливість викликати меню паузи. З

цього стану користувач може перейти до загального інтерфейсу або діалогового вікна гри. В будь-який момент можна активувати паузу, яка є перехідним станом до завершення гри. Таким чином, діаграма описує послідовність станів гри та основні точки взаємодії користувача.

Для розуміння як працює головне меню було розроблено діаграму активності зображено на Рис. 2.2.

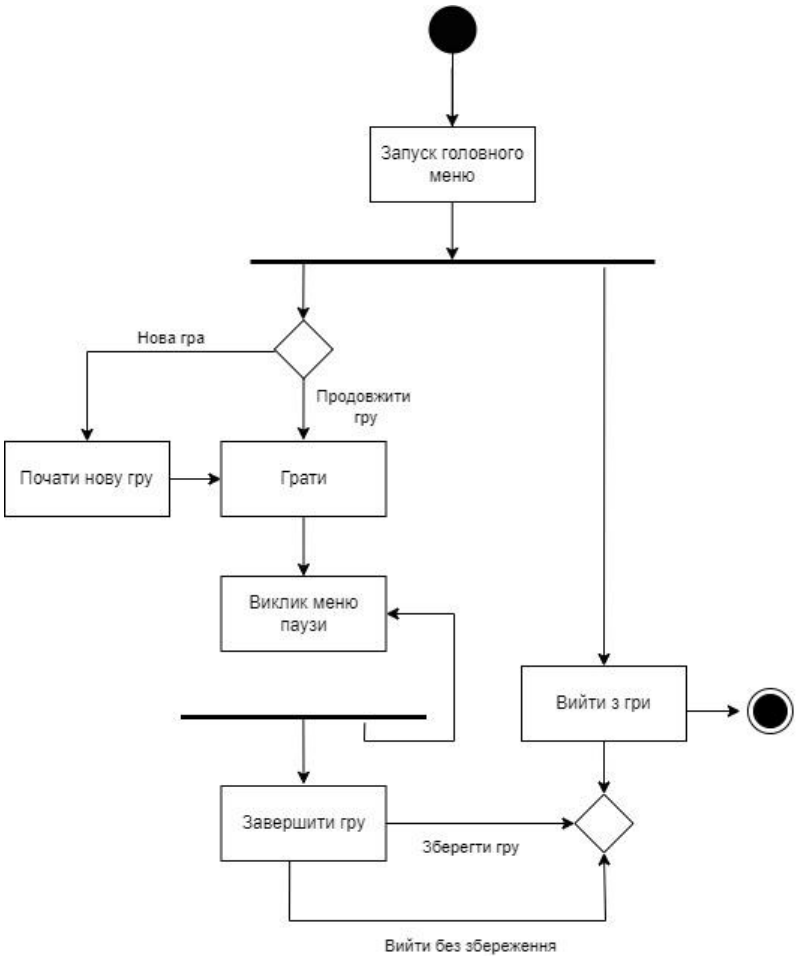

Рис. 2.2 – UML-діаграма активності ігрового меню

На діаграмі показано процес взаємодії користувача з головним меню гри через UML-діаграму активності. Починаючи з запуску головного меню, користувач може вибрати нову гру або продовжити попередню. У разі вибору нової гри, користувач починає нову сесію. Під час гри можна викликати меню паузи, де є можливість завершити гру або зберегти її прогрес. У разі завершення гри без збереження, користувач виходить з гри. Таким чином, діаграма демонструє основні етапи ігрового процесу з акцентом на вибір користувача та відповідні дії системи.

Щоб показати, що може робити гравець – розроблена ще одна діаграма активності зображена на Рис. 2.2.

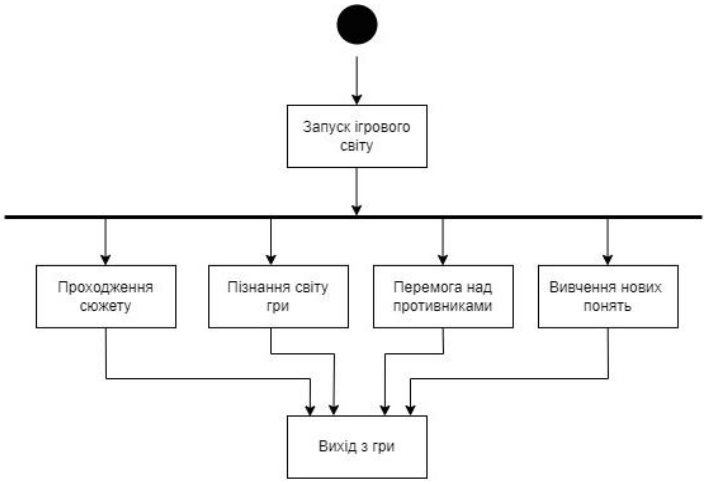

Рис. 2.3 – UML-діаграма активності дій гравця

На діаграмі активності дій гравця зображено основні етапи взаємодії гравця з ігровим світом. Починаючи із запуску ігрового світу, гравець має кілька можливих активностей: проходження сюжету, пізнання світу гри, перемога над противниками та вивчення нових понять. Кожна з цих активностей може призвести до виходу з гри, що завершує сесію гравця. Діаграма ілюструє різноманітні шляхи, якими може йти гравець у процесі гри, відображаючи його різні дії та можливі переходи між ними. На Рис. 2.4 представлено UML-діаграму класів системи.

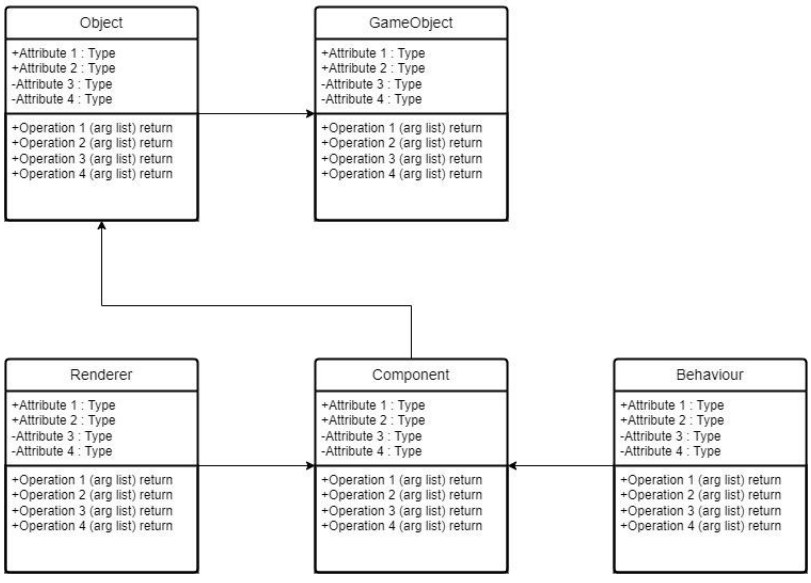

Рис. 2.4 – UML-діаграма класів

На діаграмі класів зображено структуру ієрархії та взаємозв’язків між основними класами в системі. В основі діаграми знаходиться базовий клас Object, від якого наслідуються інші класи. Клас GameObject успадковує властивості та методи класу Object і є базовим класом для трьох інших класів: Renderer, Component та Behaviour. Кожен з цих класів містить перелік атрибутів та операцій, що визначають їхні специфічні характеристики та функціональні можливості. Дана діаграма дозволяє візуально представити спадковість та зв’язки між класами, що сприяє кращому розумінню архітектури системи та полегшує процес її розробки.

Для розуміння, яку роль виконує розробник, а яку користувач (гравець) – створено діаграму прецедентів зображено на Рис. 2.5.

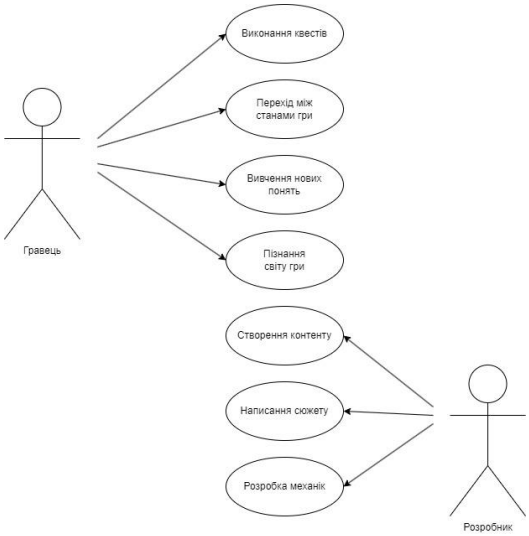

Рис. 2.5 – UML-діаграма прецедентів

На діаграмі зображено взаємодію користувачів із системою, а саме гравця та розробника. Гравець взаємодіє з системою через такі прецеденти: виконання квестів, перехід між станами гри, вивчення нових понять та пізнання світу гри. З іншого боку, розробник взаємодіє з системою через прецеденти створення контенту, написання сюжету та розробка механік. Ця діаграма допомагає наочно представити функціональність системи з точки зору різних користувачів, що є ключовим для розуміння вимог до системи та її подальшої розробки.

Щоб відобразити взаємодію користувача з системою було побудовано діаграму послідовності зображено Рис. 2.6.

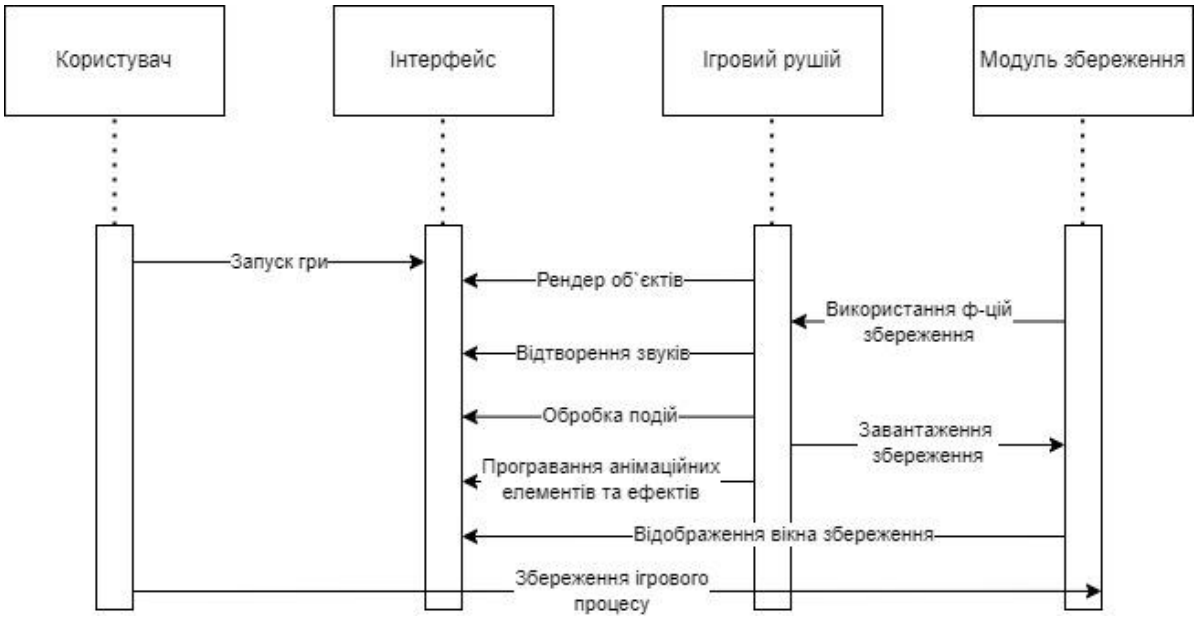

Рис. 2.6 – UML-діаграма послідовності

На діаграмі послідовності показано взаємодію між різними компонентами системи під час запуску ігрового процесу. Діаграма включає такі компоненти: Користувач, Інтерфейс, Ігровий рушій і Модуль збереження.

Процес починається з дії "Запуск гри" від користувача до інтерфейсу. Інтерфейс виконує рендер об’єктів, відтворення звуків і обробку подій. Далі інтерфейс відправляє запити до ігрового рушія для програвання анімаційних елементів та ефектів і збереження ігрового процесу. Ігровий рушій, в свою чергу, взаємодіє з модулем збереження для використання функцій збереження та завантаження збережень. Відображення вікна збереження завершує процес.

Висновки до другого розділу

Другий розділ присвячено моделюванню бізнес-процесів та інформаційно-розважальної системи. В рамках моделювання бізнес-процесів було виявлено ключові етапи функціонування системи, що дозволило чітко визначити вимоги до її архітектури. Моделювання самої системи дало змогу створити детальну модель, яка відображає її структуру та функціональні можливості, забезпечуючи ефективність та зручність використання для кінцевих користувачів.

РОЗДІЛ 3 ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНО-РОЗВАЖАЛЬНОЇ СИСТЕМИ

3.1 Проектування інтерфейсу інформаційно-розважальної системи

Для забезпечення належної функціональності інформаційної системи необхідно розробити користувацький інтерфейс. Він забезпечує можливість взаємодії гравців з грою і може суттєво впливати на загальне враження від гри.

Для RPG жанру, де часто присутні складні системи та механіки, належно розроблений користувацький інтерфейс має велике значення. Він повинен бути легким у використанні та інтуїтивно зрозумілим, щоб гравці могли швидко оволодіти керуванням грою. Гра складається з різних інтерфейсів, з якими користувач взаємодіє, і які надають йому конкретну інформацію. Після запуску гри користувача вітає головне меню зображено у додатках на Рис. 2.7.

Гравець може вибрати одну з доступних опцій для подальшої взаємодії з грою або вийти з неї. Після запуску гри завантажується стандартний інтерфейс від третьої особи зображено у додатках на Рис. 2.8.

У лівому нижньому кутку екрану розташовано зображення, яке є іконкою та вказує на поточну зброю в руках гравця, а також показує кількість доступних боєприпасів. Зверху ліворуч розташовано три повзунки: червоний повзунок відображає рівень здоров’я, блакитний повзунок відображає рівень енергії, а зелений повзунок відображає рівень втоми.

3.2 Процес реалізації інформаційно-розважальної системи на базі рушія Unity

Розробка проекту була розпочата з налаштування проекту в Unity. Були встановлені базові параметри, такі як якість графіки, розширення екрану та управління ресурсами, для створення оптимального середовища для розробки гри. Потім було перейдено до налаштування системи введення гравця з використанням Unity Input System. Були створені схеми вводу (Input Actions), які дозволяли обробляти різні дії гравця, такі як рух, стрибки та атаки, що стали основою для інтерактивності гри.

Потім було імпортовано спрайти гравця, налаштовано Sprite Renderer для їх коректного відображення на сцені, а також створено анімації для різних дій гравця з використанням Animator, включаючи біг, стрибки та ривки, що додало динамічності та реалізму грі. Використання осциляції та затримки дозволило створювати плавні переходи та складні анімації. Метод OnValidate застосовано для автоматичної перевірки та оновлення властивостей об’єктів під час редагування в Unity Editor.

Для забезпечення реалістичного руху персонажа додано компонент Rigidbody2D та налаштовано фізичні параметри, такі як маса, гравітація та тертя, що дозволило гравцеві природно взаємодіяти з навколишнім середовищем та іншими об’єктами у грі. Розроблено систему управління здоров’ям: коли гравець отримував ушкодження, його здоров’я зменшувалося, а при досягненні нульового здоров’я активувалися певні анімації та ефекти, після чого гравець повертався до останньої точки збереження або стартового місця. Це спонукало гравця бути обережнішим та стратегічно підходити до своїх дій.

Система витривалості додала стратегії, оскільки витривалість використовувалася для різних дій, таких як біг та ухиляння, і відновлювалася з часом. Відкидання ворогів при отриманні ударів реалізовано через додавання сили до Rigidbody2D ворога, а також налаштовано спалах пошкодження, який змінював колір спрайта ворога при отриманні ударів для візуального відгуку.

Реалізовані види зброї мали свої унікальні властивості. Зброя була окремими об’єктами, що легко додавалися до інвентаря гравця. Для кожного виду зброї налаштовано властивості, такі як шкода, дальність атаки та швидкість. Анімації меча включали анімаційні кліпи та візуальні ефекти для реалістичності бою. Взаємодія меча з ворогами здійснювалася за допомогою Collider2D та компонента Health для відслідковування стану ворогів.

Лук та магічний посох додали більше варіативності у грі. Лук дозволяв стріляти стрілами, а магічний посох - запускати магічні снаряди з унікальними властивостями та анімаціями. Для обох видів зброї реалізовано систему перезарядки, що обмежувала частоту атак та вимагала стратегічного підходу.

Фізика стріл була налаштована так, щоб вони летіли по параболічній траєкторії та взаємодіяли з навколишнім середовищем.

Для створення рівнів та навколишнього середовища використано Tilemap, що спростило процес створення складних рівнів за допомогою правил плиток та автоматичного розміщення. Додано предмети, які гравець міг підбирати, такі як зброя, зілля та ресурси, що дозволяло поповнювати інвентар та використовувати їх за потребою. Реалізовані об’єкти, що руйнуються, реагували на атаки гравця, додавши більше можливостей для взаємодії з оточенням.

Інтерфейс гри створено з використанням Unity UI, включаючи панелі, кнопки, слайдери та інші елементи для зручної взаємодії гравця з грою. Система перемикання активності UI елементів дозволяла показувати або приховувати певні елементи залежно від ситуації в грі. Відображення стану здоров’я та витривалості гравця реалізовано через спеціальний інтерфейс з використанням слайдерів та інших UI елементів. Для управління інвентарем створено спливаюче вікно вибору предметів, що дозволяло швидко переглядати та вибирати предмети.

Менеджер економіки відстежував кількість зібраних ресурсів гравцем та дозволяв купувати нові предмети та покращення. Для додавання візуальних ефектів реалізовано систему уламків, що активувалася при розбитті об’єктів або перемозі над ворогами, створюючи динамічні та естетично привабливі ефекти.

Додано противників та розроблено для них штучний інтелект. Противники можуть патрулювати, переслідувати та ухилятися, що робить гру більш викликовою та цікавою. Патрулювання дозволяє противникам пересуватися за певними маршрутами, створюючи відчуття постійної загрози. При виявленні гравця противники починають переслідування, намагаючись знищити його або витіснити з певної зони. Ухиляння від атак гравця робить бойові зіткнення більш динамічними та додає складності, вимагаючи від гравця більшої майстерності та планування своїх дій.

Щоб зробити гру більш інтерактивною та насиченою, було додано систему квестів. Квести включали різноманітні завдання, такі як пошук предметів, знищення ворогів або вирішення головоломок, що додавало глибини сюжету та

мотивації для гравця. Кожен квест мав свої етапи та нагороди, що стимулювало гравця до подальшого проходження.

Фінальним етапом стало додавання звукових ефектів та музичного супроводу для створення атмосфери гри. Було реалізовано систему аудіо, яка адаптувала музику та звукові ефекти залежно від ситуації в грі: спокійні мелодії під час дослідження та драматичні звуки під час боїв. Звукові ефекти додавали реалістичності діям гравця та ворогів, а також взаємодії з навколишнім середовищем.

3.3 Інструкція користувачу та тестування інформаційно-розважальної системи

Незалежно від платформи, на якій створена гра, важливим елементом є інструкції з керування для користувача. Вони надають необхідну інформацію про те, як виконувати різні дії та керувати грою. Відсутність таких інструкцій може викликати плутанину та розчарування у гравців, що може призвести до втрати інтересу до гри. Особливо важливі інструкції для рольових ігор, створених на Unity, оскільки вони часто мають складний інтерфейс та геймплей. Якщо гравці не зрозуміють, як керувати грою та досягати своїх цілей, вони можуть зіткнутися з труднощами та незадоволенням. Тому інструкції з керування повинні бути чіткими, зрозумілими та максимально інформативними.

Для запуску ігор на ПК необхідні певні технічні характеристики. Рекомендовані вимоги для нашої гри включають процесор Intel Core i5-8250U або AMD Ryzen 5 4500U, відеокарту Nvidia 450 GTS або Radeon HD 5750, та 4 ГБ оперативної пам’яті. Інші характеристики менш критичні. Для запуску гри потрібна операційна система Microsoft Windows 10 або новіша версія.

Після успішного завантаження головного меню ви можете продовжити гру, вибравши одну з опцій. Якщо хочете розпочати нову гру, оберіть "Play" або схожу опцію. Для відновлення попереднього прогресу виберіть "Continue".

Управління в грі подібне до інших рольових ігор, що полегшує його освоєння досвідченим гравцям. Для новачків передбачено навчання під час першого запуску гри, щоб ознайомити їх з основними функціями та кнопками. Управління

персонажем здійснюється за допомогою клавіш "WASD" для руху в чотирьох напрямках, клавіші "Space" для стрибків, клавіші "E" для взаємодії з об’єктами, правої кнопки миші для атаки та стрільби, а колеса миші для перемикання між предметами. Клавіша "Escape" відкриває меню паузи.

Гра автоматично надає підказки через внутрішні системи, що забезпечує детальну інформацію та допомагає освоїти всі аспекти геймплею. Це робить гру більш доступною та приємною для вивчення, дозволяючи гравцю ефективно взаємодіяти з усіма аспектами віртуального світу гри.

Тестування та аналіз результатів інформаційно-розважальної системи в жанрі RPG з візуалізацією об’єктів в стилі Souls-like, розробленої на рушії Unity, були ретельно сплановані та реалізовані для забезпечення високої якості кінцевого продукту. Процес тестування складався з кількох етапів: внутрішнє тестування розробниками, бета-тестування з залученням зовнішніх користувачів, а також аналіз та обробка зворотного зв’язку.

Внутрішнє тестування почалося з модульного тестування окремих компонентів системи. Кожен модуль, такий як бойова система, система управління персонажем, інвентар та інтерфейс користувача, був перевірений на коректність функціонування. За допомогою Unity Test Framework розробники створювали автоматизовані тести, які перевіряли основні функції та логіку гри. Це дозволило швидко виявити та виправити помилки на ранніх етапах розробки [20].

Наступним кроком було інтеграційне тестування, під час якого всі модулі об’єднувалися та перевірялися на сумісність один з одним. Особлива увага приділялася перевірці продуктивності та стабільності гри. Для цього використовувалися інструменти профайлінгу Unity, такі як Unity Profiler та Frame Debugger, які дозволяли виявляти та усувати вузькі місця в продуктивності, оптимізувати використання пам’яті та графічні ресурси[34]. Важливим аспектом було забезпечення стабільної частоти кадрів та оптимізації графічних налаштувань.

Після успішного завершення внутрішнього тестування було проведено бета-тестування із залученням вибраної групи гравців, які відповідають цільовій

аудиторії проєкту. Бета-тестери грали в гру в різних умовах, що дозволило виявити специфічні проблеми, пов’язані з продуктивністю та сумісністю.

Бета-тестери активно спілкувалися з розробниками, надаючи зворотний зв’язок щодо зручності використання, складності гри та загального враження від ігрового процесу. Було виявлено кілька областей, які потребували доопрацювання:

- Система управління: Деякі користувачі повідомили про труднощі з налаштуванням управління. Це призвело до внесення змін в систему налаштувань, що дозволило гравцям більш гнучко налаштовувати управління під свої потреби.
- Бойова система: Кілька гравців зазначили, що певні аспекти бойової системи могли б бути більш інтуїтивно зрозумілими. Це призвело до переробки деяких механік та додавання детальніших підказок і навчальних матеріалів.
- Візуальні ефекти: Частина тестерів висловила побажання щодо покращення певних візуальних ефектів, що було враховано при фінальному поліруванні графіки.

Після завершення бета-тестування був проведений детальний аналіз зібраного зворотного зв’язку. Всі зауваження були систематизовані та пріоритизовані. Команда розробників внесла відповідні зміни та покращення в гру, використовуючи гнучкі можливості Unity для швидкого прототипування та тестування змін.

Важливу роль у цьому процесі відіграла можливість швидкого оновлення гри та розповсюдження оновлень серед тестувальників, що дозволило оперативно перевіряти внесені зміни та отримувати нові відгуки. Unity Cloud Build спрощував процес автоматизованого складання нових версій гри, що значно прискорило тестування.

Після внесення всіх необхідних коректив було проведено фінальне тестування, яке підтвердило, що гра працює стабільно та відповідає заявленим вимогам. Особлива увага приділялася перевірці на відповідність ігрового процесу

концепції Souls-like, включаючи високий рівень складності, атмосфери та візуального стилю.

Фінальне тестування також включало:

- Перевірку продуктивності на різних апаратних конфігураціях: Це забезпечило оптимальну роботу гри на широкому спектрі пристроїв.
- Перевірку сумісності з різними операційними системами: Тестування гри на різних версіях Windows для забезпечення стабільної роботи.
- Перевірку на наявність помилок та збоїв: Повторне та розширене тестування всіх ігрових механік та функцій для виявлення будь-яких залишкових помилок.

Загалом, інформаційно-розважальна система показала високий рівень задоволення користувачів, що є ключовим показником успішності проєкту. Використання рушія Unity дозволило досягти високої якості гри завдяки гнучкості, потужним інструментам для розробки та тестування, а також можливостям крос-платформної підтримки. Тестування та аналіз результатів стали невід’ємною частиною процесу розробки, забезпечуючи високу якість кінцевого продукту та задоволення гравців. Завдяки ретельному тестуванню на всіх етапах розробки, проєкт виявився стабільним, продуктивним та привабливим для цільової аудиторії.

Висновки до третього розділу

У третьому розділі описано процес проектування та реалізації інформаційно-розважальної системи. Було розроблено інтерфейс користувача, що відповідає сучасним вимогам до ергономіки та зручності використання. Реалізація системи здійснена на базі рушія Unity, що забезпечує високу продуктивність та інтерактивність. Крім того, розроблено детальну інструкцію користувача та проведено тестування системи, результати якого підтвердили відповідність функціональних можливостей заявленим вимогам та високу якість програмного продукту.

ВИСНОВОК

В процесі виконання кваліфікаційної роботи було досягнуто таких результатів.

Проведено аналіз інформаційних потреб і визначено предметну область. Це дозволило сформулювати чіткі вимоги до розробки інформаційно-розважальної системи. Здійснено аналіз переваг і недоліків існуючих рішень, що дозволило виявити слабкі місця та визначити напрямки для вдосконалення розроблюваного продукту. Проведено детальний аналіз технічного забезпечення, що включало оцінку апаратного та програмного забезпечення, необхідного для успішної реалізації проекту. Виконано моделювання бізнес-процесів, що дозволило побудувати ефективну структуру роботи системи та оптимізувати її функціонування.

Виконано моделювання інформаційно-розважальної системи, що включало розробку архітектури системи, визначення основних компонентів та їх взаємодії. Спроектовано інтерфейс інформаційно-розважальної системи, який забезпечує зручність використання та задовольняє вимоги користувачів. Реалізовано інформаційно-розважальну систему на базі рушія Unity, що дозволило створити ефективну та продуктивну гру з високою якістю графіки та функціональністю.

Розроблено інструкцію користувача, що допомагає швидко і ефективно освоїти роботу з інформаційно-розважальною системою. Проведено тестування системи та аналіз результатів, що дозволило виявити і виправити помилки, а також забезпечити високу стабільність і надійність роботи системи.

Таким чином, в результаті виконання кваліфікаційної роботи було створено інформаційно-розважальну систему, яка відповідає сучасним вимогам та забезпечує високу якість і зручність використання. Проведені дослідження і розробка дозволили вирішити поставлені задачі та досягти мети роботи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Левченко Д. Ю. Особливості розробки сучасних комп’ютерних ігор. Безпека, технології, інновації: нові горизонти : збірник праць учасників міжфакультетської науково-практичної інтернет-конференції здобувачів вищої освіти і молодих вчених, 15 листопада 2023 р. Житомир : Поліський національний університет, 2023. 68 с.

2. Левченко Д. Ю Головоломка (жанр відео ігор). Інформаційні технології та моделювання систем : збірник праць учасників Всеукраїнської науково-практичної конференції здобувачів вищої освіти і молодих вчених, 10 квітня 2024 р. Житомир : Поліський національний університет, 2024. 76 с.

3. What Does Role-Playing Game Mean? – Definition from Techopedia. URL: http://surl.li/uqfuh (дата звернення: 15.06.2024)

4. Role-playing video game. URL: http://surl.li/uqfuk (дата звернення: 10.12.2023)

5. RPG Guide: 6 Types of Role-Playing Games. URL: http://surl.li/uqfuo (дата звернення: 15.06.2024)

6. What is D&D, Dungeons & Dragons. URL: http://surl.li/uqfup (дата звернення: 15.06.2024)

7. Learn To Create A RPG Game - Full Lifetime Access. URL: URL: http://surl.li/uqfvo (дата звернення: 15.06.2024)

8. What Is A Role-Playing Game? RPG Types Explained. URL: http://surl.li/uqfwx (дата звернення: 15.06.2024)

9. Role-playing video game | History & Examples. URL: http://surl.li/uqfwy (дата звернення: 15.06.2024)

10. Top free Role-Playing games tagged 2D. URL: http://surl.li/uqfxl (дата звернення: 15.06.2024)

11. PC RPG Video Games. URL: http://surl.li/uqfxq (дата звернення: 15.06.2024)

12. The 10 Best Souls-like Games. URL: http://surl.li/uqfys (дата звернення: 15.06.2024)

13. Salt and Sanctuary, Ska Studio. URL: http://surl.li/uqfuq (дата звернення: 15.06.2024)

14. Blasphemous. URL: http://surl.li/uqfut (дата звернення: 15.06.2024)

15. Hollow Knight. URL: http://surl.li/uqfuv (дата звернення: 15.06.2024)

16. Death’s Gambit. URL: http://surl.li/uqfuy (дата звернення: 15.06.2024)

17. Dead Cells. URL: http://surl.li/uqfva (дата звернення: 15.06.2024)

18. Unity Real-Time Development Platform.URL: http://surl.li/uqfve (дата звернення: 15.06.2024)

19. Unreal Engine: The most powerful real-time 3D creation tool. URL: http://surl.li/uqfvf (дата звернення: 15.06.2024)

20. About Unity Test Framework. URL: http://surl.li/uqiwq (дата звернення: 15.06.2024)

21. QuickStart Guide: Unity’s Test Framework in Unity 2022. URL: http://surl.li/uqixj (дата звернення: 15.06.2024)

22. Godot Engine - Free and open source 2D and 3D game engine. URL: http://surl.li/uqfvi (дата звернення: 15.06.2024)

23. CryEngine. URL: http://surl.li/uqfvl (дата звернення: 15.06.2024)

24. Junior Programmer Pathway – Unity Learn. URL: http://surl.li/uqfvo (дата звернення: 15.06.2024)

25. What are “RPG mechanics”. URL: http://surl.li/uqfxz (дата звернення: 15.06.2024)

26. Top 10 RPG Mechanics You Should Steal. URL: http://surl.li/uqfyf (дата звернення: 15.06.2024)

27. Browse RPG Mechanics. URL: http://surl.li/uqfyk (дата звернення: 15.06.2024)

28. Great RPG Mechanics: Week One. URL: http://surl.li/uqfyn (дата звернення: 15.06.2024)

29. What Everyone Gets Wrong about Souls-like Design. URL: http://surl.li/uqfzf (дата звернення: 15.06.2024)

30. C# docs - get started, tutorials, reference. URL: http://surl.li/uqifd (дата

звернення: 15.06.2024)

31. Programming concepts (C#). URL: http://surl.li/uqifj (дата звернення: 15.06.2024)

32. Complete C# Unity Game Developer 2D. URL: http://surl.li/uqimw (дата звернення: 15.06.2024)

33. Visual Studio: IDE. URL: http://surl.li/uqink (дата звернення: 15.06.2024)

34. Profiler overview. URL: http://surl.li/uqirv (дата звернення: 15.06.2024)

35. The Best Game Engines You Should Consider for 2024. URL: http://surl.li/uqiuu (дата звернення: 15.06.2024)

36. Mark Reed «C#: The Ultimate Advanced Guide To Master C# Programming»: посібник, 2022. - 138с.

37. Vardan Grigoryan. «Expert C++: Become a proficient programmer by learning coding best practices with C++17 and C++20’s latest features»: посібник, Packt Publishing, 2020. 606 с.

38. Will Goldstone. «Unity Game Development Essentials»: посібник, 2009. 177с.

39. Jane McGonigal. «Reality is Broken: Why Games Make Us Better and How They Can Change the World.»: посібник, Penguin Books, 2020, 416 с.

40. Jesse Schell. «The Art of Game Design: A Book of Lenses»: посібник, CRC Press, 2021, 520 с.

ДОДАТКИ

Табл. 1.1 – Аналіз технічного забезпечення

Параметр	Unity	Unreal Engine	Godot	CryEngine
Продуктивність	Висока	Дуже висока	Середня	Висока
Графічні Можливості	Високі	Дуже високі	Помірні	Дуже високі
Скриптування	C#	C++ / Blueprints	GDScript / C# / C++	C++ / Lua
Інструменти для Анімацій	Потужні	Дуже потужні	Достатні	Потужні
Редагування UI	Інтуїтивне	Потужне	Просте	Середнє
Мультиплатформенність	ПК, мобільні, консолі, ВР	ПК, мобільні, консолі, ВР	ПК, мобільні, веб	ПК, консолі
Спільнота	Дуже велика	Велика	Зростаюча	Середня
Бібліотека Ресурсів	Величезна	Велика	Середня	Середня
Простота Використання	Висока	Середня	Висока	Низька
Ліцензія	Безкоштовно (з обмеженнями)	Безкоштовно (з обмеженнями)	Повністю безкоштовно	Безкоштовно (з обмеженнями)

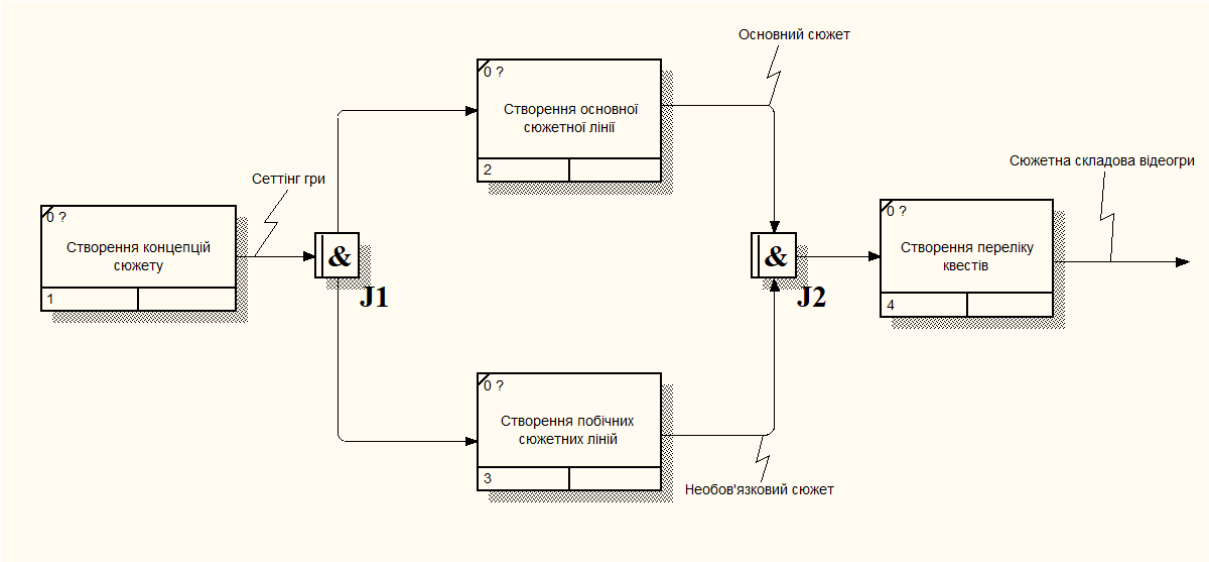

Рис. 1.4 – IDEF3 модель створення сюжету

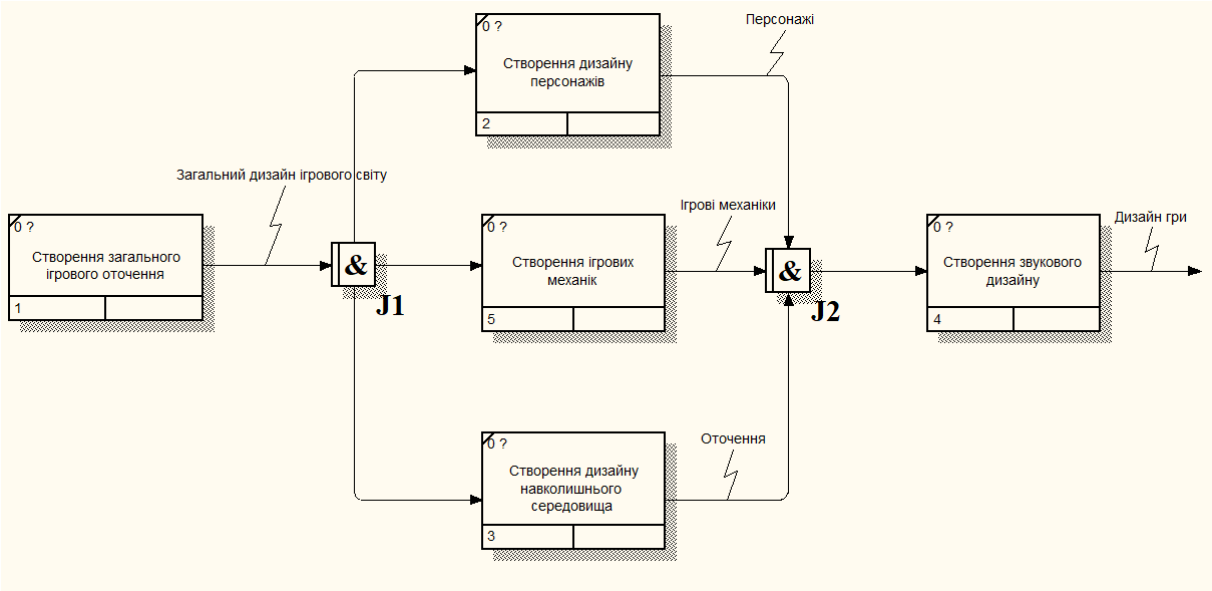

Рис. 1.5 – IDEF3 модель ігрового оточення

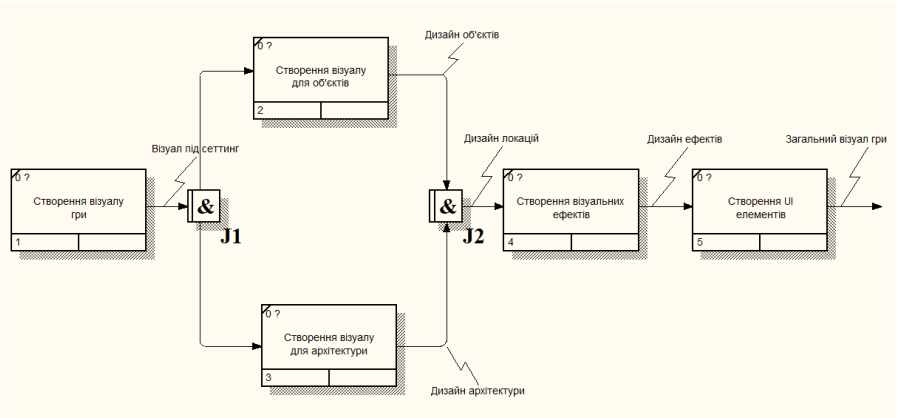

Рис. 1.6 – IDEF3 модель візуальної частини

Рис. 2.7 – Головне меню

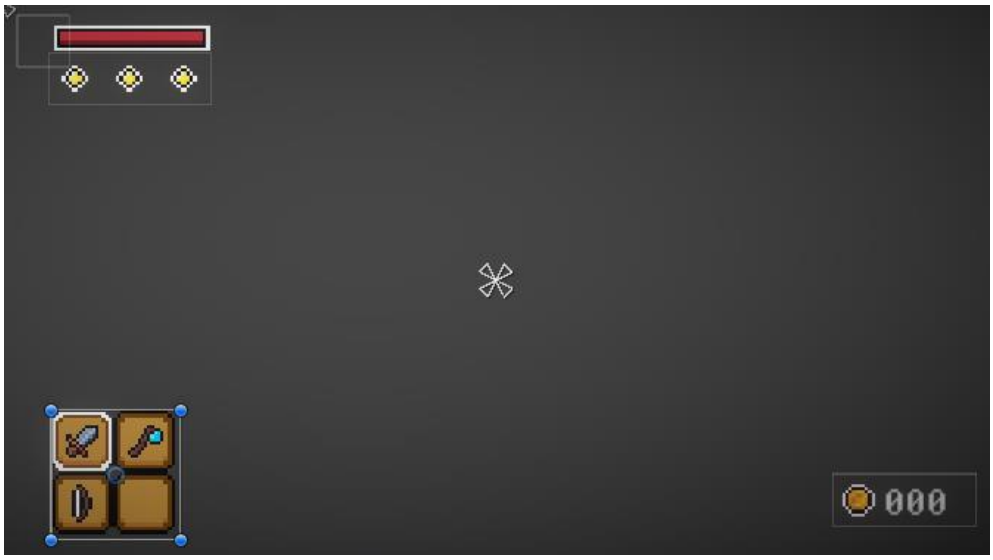

Рис. 2.8 – Інтерфейсу від третьої особи

Файл PlayerController.cs

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

public class PlayerController : Singleton<PlayerController>
{
    public bool FacingLeft { get { return facingLeft; } }


    [SerializeField] private float moveSpeed = 1f;
    [SerializeField] private float dashSpeed = 4f;
    [SerializeField] private TrailRenderer myTrailRenderer;
    [SerializeField] private Transform weaponCollider;

    private PlayerControls playerControls;
    private Vector2 movement;
    private Rigidbody2D rb;
    private Animator myAnimator;
```

```
private SpriteRenderer mySpriteRender;
private Knockback knockback;
private float startingMoveSpeed;

private bool facingLeft = false;
private bool isDashing = false;

protected override void Awake() {
    base.Awake();

    playerControls = new PlayerControls();
    rb = GetComponent<Rigidbody2D>();
    myAnimator = GetComponent<Animator>();
    mySpriteRender = GetComponent<SpriteRenderer>();
    knockback = GetComponent<Knockback>();
}

private void Start() {
    playerControls.Combat.Dash.performed += _ => Dash();

    startingMoveSpeed = moveSpeed;

    ActiveInventory.Instance.EquipStartingWeapon();
}

private void OnEnable() {
    playerControls.Enable();
}

private void OnDisable() {
    playerControls.Disable();
}

private void Update() {
    PlayerInput();
}

private void FixedUpdate() {
    AdjustPlayerFacingDirection();
    Move();
}

public Transform GetWeaponCollider() {
    return weaponCollider;
}

private void PlayerInput() {
    movement = playerControls.Movement.Move.ReadValue<Vector2>();

    myAnimator.SetFloat("moveX", movement.x);
    myAnimator.SetFloat("moveY", movement.y);
}

private void Move() {
    if (knockback.GettingKnockedBack || PlayerHealth.Instance.isDead) { return; }

    rb.MovePosition(rb.position + movement * (moveSpeed * Time.fixedDeltaTime));
}

private void AdjustPlayerFacingDirection() {
    Vector3 mousePos = Input.mousePosition;
    Vector3 playerScreenPoint = Camera.main.WorldToScreenPoint(transform.position);
```

```
        if (mousePos.x < playerScreenPoint.x) {
            mySpriteRender.flipX = true;
            facingLeft = true;
        } else {
            mySpriteRender.flipX = false;
            facingLeft = false;
        }
    }

    private void Dash() {
        if (!isDashing && Stamina.Instance.CurrentStamina > 0) {
            Stamina.Instance.UseStamina();
            isDashing = true;
            moveSpeed *= dashSpeed;
            myTrailRenderer.emitting = true;
            StartCoroutine(EndDashRoutine());
        }
    }

    private IEnumerator EndDashRoutine() {
        float dashTime = .2f;
        float dashCD = .25f;
        yield return new WaitForSeconds(dashTime);
        moveSpeed = startingMoveSpeed;
        myTrailRenderer.emitting = false;
        yield return new WaitForSeconds(dashCD);
        isDashing = false;
    }
}
```

Файл PlayerHealth.cs

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

public class PlayerHealth : Singleton<PlayerHealth>
{
    public bool isDead { get; private set; }

    [SerializeField] private int maxHealth = 3;
    [SerializeField] private float knockBackThrustAmount = 10f;
    [SerializeField] private float damageRecoveryTime = 1f;

    private Slider healthSlider;
    private int currentHealth;
    private bool canTakeDamage = true;
    private Knockback knockback;
    private Flash flash;

    const string HEALTH_SLIDER_TEXT = "Health Slider";
    const string TOWN_TEXT = "Scene1";
    readonly int DEATH_HASH = Animator.StringToHash("Death");

    protected override void Awake() {
        base.Awake();

        flash = GetComponent<Flash>();
        knockback = GetComponent<Knockback>();
```

```
}

private void Start() {
    isDead = false;
    currentHealth = maxHealth;

    UpdateHealthSlider();
}

private void OnCollisionStay2D(Collision2D other) {
    EnemyAI enemy = other.gameObject.GetComponent<EnemyAI>();

    if (enemy) {
        TakeDamage(1, other.transform);
    }
}

public void HealPlayer() {
    if (currentHealth < maxHealth) {
        currentHealth += 1;
        UpdateHealthSlider();
    }
}

public void TakeDamage(int damageAmount, Transform hitTransform) {
    if (!canTakeDamage) { return; }

    ScreenShakeManager.Instance.ShakeScreen();
    knockback.GetKnockedBack(hitTransform, knockBackThrustAmount);
    StartCoroutine(flash.FlashRoutine());
    canTakeDamage = false;
    currentHealth -= damageAmount;
    StartCoroutine(DamageRecoveryRoutine());
    UpdateHealthSlider();
    CheckIfPlayerDeath();
}

private void CheckIfPlayerDeath() {
    if (currentHealth <= 0 && !isDead) {
        isDead = true;
        Destroy(ActiveWeapon.Instance.gameObject);
        currentHealth = 0;
        GetComponent<Animator>().SetTrigger(DEATH_HASH);
        StartCoroutine(DeathLoadSceneRoutine());
    }
}

private IEnumerator DeathLoadSceneRoutine() {
    yield return new WaitForSeconds(2f);
    Destroy(gameObject);
    SceneManager.LoadScene(TOWN_TEXT);
}

private IEnumerator DamageRecoveryRoutine() {
    yield return new WaitForSeconds(damageRecoveryTime);
    canTakeDamage = true;
}

private void UpdateHealthSlider() {
    if (healthSlider == null) {
        healthSlider = GameObject.Find(HEALTH_SLIDER_TEXT).GetComponent<Slider>();
    }
```

```
        healthSlider.maxValue = maxHealth;
        healthSlider.value = currentHealth;
    }
}
```

Файл Stamina.cs

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

public class Stamina : Singleton<Stamina>
{
    public int CurrentStamina { get; private set; }

    [SerializeField] private Sprite fullStaminaImage, emptyStaminaImage;
    [SerializeField] private int timeBetweenStaminaRefresh = 3;

    private Transform staminaContainer;
    private int startingStamina = 3;
    private int maxStamina;
    const string STAMINA_CONTAINER_TEXT = "Stamina Container";

    protected override void Awake() {
        base.Awake();

        maxStamina = startingStamina;
        CurrentStamina = startingStamina;
    }

    private void Start() {
        staminaContainer = GameObject.Find(STAMINA_CONTAINER_TEXT).transform;
    }

    public void UseStamina() {
        CurrentStamina--;
        UpdateStaminaImages();
    }

    public void RefreshStamina() {
        if (CurrentStamina < maxStamina) {
            CurrentStamina++;
        }
        UpdateStaminaImages();
    }

    private IEnumerator RefreshStaminaRoutine() {
        while (true)
        {
            yield return new WaitForSeconds(timeBetweenStaminaRefresh);
            RefreshStamina();
        }
    }

    private void UpdateStaminaImages() {
        for (int i = 0; i < maxStamina; i++)
        {
            if (i <= CurrentStamina - 1) {
                staminaContainer.GetChild(i).GetComponent<Image>().sprite = fullStaminaImage;
            } else {
```

```
            staminaContainer.GetChild(i).GetComponent<Image>().sprite = emptyStaminaImage;
        }
    }

    if (CurrentStamina < maxStamina) {
        StopAllCoroutines();
        StartCoroutine(RefreshStaminaRoutine());
    }
  }
}
```

Файл EnemyAI.cs

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

public class EnemyAI : MonoBehaviour
{
  [SerializeField] private float roamChangeDirFloat = 2f;
  [SerializeField] private float attackRange = 0f;
  [SerializeField] private MonoBehaviour enemyType;
  [SerializeField] private float attackCooldown = 2f;
  [SerializeField] private bool stopMovingWhileAttacking = false;

  private bool canAttack = true;

  private enum State {
    Roaming,
    Attacking
  }

  private Vector2 roamPosition;
  private float timeRoaming = 0f;

  private State state;
  private EnemyPathfinding enemyPathfinding;

  private void Awake() {
    enemyPathfinding = GetComponent<EnemyPathfinding>();
    state = State.Roaming;
  }

  private void Start() {
    roamPosition = GetRoamingPosition();
  }

  private void Update() {
    MovementStateControl();
  }

  private void MovementStateControl() {
    switch (state)
    {
      default:
      case State.Roaming:
        Roaming();
      break;

      case State.Attacking:
        Attacking();
      break;
```

```
        }
    }

    private void Roaming() {
        timeRoaming += Time.deltaTime;

        enemyPathfinding.MoveTo(roamPosition);

        if (Vector2.Distance(transform.position, PlayerController.Instance.transform.position) < attackRange) {
            state = State.Attacking;
        }

        if (timeRoaming > roamChangeDirFloat) {
            roamPosition = GetRoamingPosition();
        }
    }

    private void Attacking() {
        if (Vector2.Distance(transform.position, PlayerController.Instance.transform.position) > attackRange)
        {
            state = State.Roaming;
        }

        if (attackRange != 0 && canAttack) {

            canAttack = false;
            (enemyType as IEnemy).Attack();

            if (stopMovingWhileAttacking) {
                enemyPathfinding.StopMoving();
            } else {
                enemyPathfinding.MoveTo(roamPosition);
            }

            StartCoroutine(AttackCooldownRoutine());
        }
    }

    private IEnumerator AttackCooldownRoutine() {
        yield return new WaitForSeconds(attackCooldown);
        canAttack = true;
    }

    private Vector2 GetRoamingPosition() {
        timeRoaming = 0f;
        return new Vector2(Random.Range(-1f, 1f), Random.Range(-1f, 1f)).normalized;
    }
}
```

Файл EnemyHealth.cs

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

public class EnemyHealth : MonoBehaviour
{
    [SerializeField] private int startingHealth = 3;
    [SerializeField] private GameObject deathVFXPrefab;
    [SerializeField] private float knockBackThrust = 15f;

    private int currentHealth;
```

```
    private Knockback knockback;
    private Flash flash;

    private void Awake() {
        flash = GetComponent<Flash>();
        knockback = GetComponent<Knockback>();
    }

    private void Start() {
        currentHealth = startingHealth;
    }

    public void TakeDamage(int damage) {
        currentHealth -= damage;
        knockback.GetKnockedBack(PlayerController.Instance.transform, knockBackThrust);
        StartCoroutine(flash.FlashRoutine());
        StartCoroutine(CheckDetectDeathRoutine());
    }

    private IEnumerator CheckDetectDeathRoutine() {
        yield return new WaitForSeconds(flash.GetRestoreMatTime());
        DetectDeath();
    }

    public void DetectDeath() {
        if (currentHealth <= 0) {
            Instantiate(deathVFXPrefab, transform.position, Quaternion.identity);
            GetComponent<PickUpSpawner>().DropItems();
            Destroy(gameObject);
        }
    }
}
```

Файл Shooter.cs

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

public class Shooter : MonoBehaviour, IEnemy
{
    [SerializeField] private GameObject bulletPrefab;
    [SerializeField] private float bulletMoveSpeed;
    [SerializeField] private int burstCount;
    [SerializeField] private int projectilesPerBurst;
    [SerializeField][Range(0, 359)] private float angleSpread;
    [SerializeField] private float startingDistance = 0.1f;
    [SerializeField] private float timeBetweenBursts;
    [SerializeField] private float restTime = 1f;
    [SerializeField] private bool stagger;
    [Tooltip("Stagger must be enabled for oscillate to function properly.")]
    [SerializeField] private bool oscillate;

    private bool isShooting = false;

    private void OnValidate() {
        if (oscillate) { stagger = true; }
        if (!oscillate) { stagger = false; }
        if (projectilesPerBurst < 1) { projectilesPerBurst = 1; }
        if (burstCount < 1) { burstCount = 1; }
        if (timeBetweenBursts < 0.1f) { timeBetweenBursts = 0.1f; }
        if (restTime < 0.1f) { restTime = 0.1f; }
```

```
    if (startingDistance < 0.1f) { startingDistance = 0.1f; }
    if (angleSpread == 0) { projectilesPerBurst = 1; }
    if (bulletMoveSpeed <= 0) { bulletMoveSpeed = 0.1f; }
}

public void Attack() {
    if (!isShooting) {
        StartCoroutine(ShootRoutine());
    }
}

private IEnumerator ShootRoutine()
{
    isShooting = true;

    float startAngle, currentAngle, angleStep, endAngle;
    float timeBetweenProjectiles = 0f;

    TargetConeOfInfluence(out startAngle, out currentAngle, out angleStep, out endAngle);

    if (stagger) { timeBetweenProjectiles = timeBetweenBursts / projectilesPerBurst; }

    for (int i = 0; i < burstCount; i++)
    {
        if (!oscillate) {
            TargetConeOfInfluence(out startAngle, out currentAngle, out angleStep, out endAngle);
        }

        if (oscillate && i % 2 != 1) {
            TargetConeOfInfluence(out startAngle, out currentAngle, out angleStep, out endAngle);
        } else if (oscillate) {
            currentAngle = endAngle;
            endAngle = startAngle;
            startAngle = currentAngle;
            angleStep *= -1;
        }


        for (int j = 0; j < projectilesPerBurst; j++)
        {
            Vector2 pos = FindBulletSpawnPos(currentAngle);

            GameObject newBullet = Instantiate(bulletPrefab, pos, Quaternion.identity);
            newBullet.transform.right = newBullet.transform.position - transform.position;


            if (newBullet.TryGetComponent(out Projectile projectile))
            {
                projectile.UpdateMoveSpeed(bulletMoveSpeed);
            }

            currentAngle += angleStep;

            if (stagger) { yield return new WaitForSeconds(timeBetweenProjectiles); }
        }

        currentAngle = startAngle;

        if (!stagger) { yield return new WaitForSeconds(timeBetweenBursts); }
    }

    yield return new WaitForSeconds(restTime);
```

```
        isShooting = false;
    }

    private void TargetConeOfInfluence(out float startAngle, out float currentAngle, out float angleStep, out float endAngle)
    {
        Vector2 targetDirection = PlayerController.Instance.transform.position - transform.position;
        float targetAngle = Mathf.Atan2(targetDirection.y, targetDirection.x) * Mathf.Rad2Deg;
        startAngle = targetAngle;
        endAngle = targetAngle;
        currentAngle = targetAngle;
        float halfAngleSpread = 0f;
        angleStep = 0;
        if (angleSpread != 0)
        {
            angleStep = angleSpread / (projectilesPerBurst - 1);
            halfAngleSpread = angleSpread / 2f;
            startAngle = targetAngle - halfAngleSpread;
            endAngle = targetAngle + halfAngleSpread;
            currentAngle = startAngle;
        }
    }

    private Vector2 FindBulletSpawnPos(float currentAngle) {
        float x = transform.position.x + startingDistance * Mathf.Cos(currentAngle * Mathf.Deg2Rad);
        float y = transform.position.y + startingDistance * Mathf.Sin(currentAngle * Mathf.Deg2Rad);

        Vector2 pos = new Vector2(x, y);

        return pos;
    }
}
```