Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська дипломна робота

на тему: «Розробка мобільної 2d-гри «Dark Medieval»»

Виконав: студент 3 курсу
групи 1ПІ-19мс2
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Муковоз В.С.
(прізвище та ініціали)

Керівник: к.т.н., доц. каф. ПЗ Катєльніков Д.І.
(прізвище та ініціали)

Рецензент: к.т.н., ст. вик. КН Озеранський В.С.
(прізвище та ініціали)

Допущено до захисту
Зав. кафедри _____
«____» _____ 2022 р.

ВНТУ – 2022

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Ступінь вищої освіти – бакалавр
Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н.
25 березня 2022 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Муковозу Віктору Сергійовичу

1. Тема роботи – Розробка мобільної 2d-гри «Dark Medieval».

Керівник роботи: Катєльніков Денис Іванович, к.т.н., доц. кафедри ПЗ, затверджений наказом вищого навчального закладу від 24 березня 2022 р. № 66.

2. Строк подання студентом роботи 13 червня 2022 р.

3. Вихідні дані до роботи: середовище розробки Unity та Visual Studio Code, мова розробки C#, операційна система – Windows 11, базові алгоритми для розробки 2d-ігор у жанрі екшн-платформер.

4. Зміст розрахунково-пояснювальної записки: аналіз використання систем прийняття рішень в ігрових додатках, обґрунтування доцільності розробки; теоретичні основи розробленого методу; варіантний аналіз та обґрунтування вибору засобів реалізації системи; проектування структур даних і модулів ігрової системи; розробка програмного коду ігрової системи; тестування роботи ігрової системи; висновки; перелік посилань; додатки.

5. Перелік графічного матеріалу: графічний інтерфейс мобільного додатку; метод поведінки ворогів у ігровому середовищі; модель ігрового середовища; блок-схеми алгоритмів роботи мобільної гри; тестування мобільної гри.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Катєльніков Д.І., к.т.н, доцент кафедри ПЗ		

7. Дата видачі завдання 25 березня 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз стану галузі, обґрунтування доцільності розробки	26.03.2022 – 01.04.2022	Вик.
2	Розробка методів і моделей реалізації мобільної 2d-гри	04.04.2022 – 15.04.2022	Вик.
3	Проектування структур даних та модулів ігрової системи	18.04.2022 – 22.04.2022	Вик.
4	Варіантний аналіз та обґрунтування вибору засобів реалізації системи	25.04.2022 – 02.05.2022	Вик.
5	Розробка програмного коду ігрової системи	03.05.2022 – 23.05.2022	Вик.
6	Тестування роботи ігрової системи	24.05.2022 – 30.05.2022	Вик.
7	Оформлення матеріалів до захисту БДР	31.05.2022 – 10.06.2022	Вик.

Студент _____ (підпис) Муковоз В.С. (прізвище та ініціали)

Керівник бакалаврської дипломної роботи _____ (підпис) Катєльніков Д.І. (прізвище та ініціали)

Анотація

Бакалаврська дипломна робота складається з 61 сторінок формату А4, на яких є 46 рисунків, 1 таблиця, список використаних джерел містить 18 найменувань

У бакалаврській дипломній роботі проведено аналіз мобільних 2d-ігор у жанрі екшн платформера. Було проведено аналіз аналогів, які існують, визначено їх переваги і недоліки.

На основі проведених досліджень було запропоновано розробити власну гру, яка втілить найкращі механіки 2d-ігор у жанрі екшн платформеру.

Розроблено 2d-гру «Dark Medieval», яка несе розважальний характер і допомагає не лише весело провести свій час, а й прийняти виклик цікавим та важким рівням, що допоможуть розвинути користувачу логіку і реакцію.

Розроблено метод, модель роботи мобільної гри та програмно реалізовано стратегію гри.

Програмний продукт було створено з використанням мови програмування C#. Для розробки графічних елементів гри було використано Adobe Photoshop 2020. В якості середовища розробки гри було обрано редактор вихідного коду Microsoft Visual Studio Code та движок для створення двовимірних ігор Unity.

Ключові слова: мобільна гра, 2D, екшн, Unity.

Abstract

The bachelor's thesis consists of 61 pages of A4 format, on which there are 46 figures, 1 table, the list of used sources contains 18 items

In the bachelor's thesis the analysis of mobile 2d-games in the genre of action platformer is carried out. The analysis of existing analogues was carried out, their advantages and disadvantages were determined.

Based on the research, it was proposed to develop its own game that will embody the best mechanics of 2d games in the action platformer genre.

Developed 2d-game "Dark Medieval", which is entertaining and helps not only to have fun, but also to challenge interesting and difficult levels that will help develop user logic and reaction.

The method, model of mobile game operation was developed and the game strategy was implemented.

The software product was created using the C# programming language. Adobe Photoshop 2020 was used to develop the game's graphics. The Microsoft Visual Studio Code source code editor and the Unity 2D game engine were chosen as the game development environment.

Keywords: mobile game, 2D, action, Unity.

ЗМІСТ

ВСТУП

Актуальність. Сьогоднішній світ перейшов на новий етап життя, де головну роль виконує інформація. Сучасний розвиток інформаційного суспільства безпосередньо пов'язаний з необхідністю збору, обробки і передачі величезних об'ємів інформації, перетворенням інформації у товар, як правило, значної вартості. Це стало причиною глобального переходу від індустріального суспільства до інформаційного[1].

На сьогодні інформаційні технології займають важливе місце в нашому житті. Застосування ЕОМ стало буденною справою, хоча ще зовсім недавно робоче місце, обладнане комп'ютером, було великою рідкістю. Інформаційні технології дали нові можливості для роботи і відпочинку, багато в чому полегшили працю і просто життя кожної сучасної людини. Теперішнє суспільство навряд чи можна уявити без інформаційних технологій.

Розробка мобільних ігор – це галузь, яка швидко розвивається. Її темпи зростання з кожним роком дивують нас. Очікується, що дохід від мобільних ігор досягне приголомшливих 102,8 мільярдів доларів до 2023 року проти 77,2 мільярда доларів у 2020 році [2].

Кількість гравців у мобільні ігри зростає, і ця кількість додатково зросла значною мірою через пандемію Covid-19. Протягом 2020 року кількість нових гравців значно зросла. Крім того, вони зазвичай проводять більше часу в грі щотижня.

Загалом попит на мобільну розробку збільшується, як і зростає аудиторія. Приблизна кількість користувачів мобільного зв’язку на сьогодні становить 7,1 мільярда. Очікується, що до 2025 року вона збільшиться до 7,49 мільярдів[2].

На відміну від розробки мобільних додатків, розробка ігор дає вам більше творчої свободи. Здебільшого ви не зв’язані стандартними обмеженнями UI/UX, накладеними платформою розповсюдження та

вимогами iOS та Android. Процес розробки мобільних ігор дає вам шанс проявити себе та усвідомити, що ваш продукт приносить людям радість [2]. Тому актуальною є розробка мобільної 2d-гри «Dark Medieval» у жанрі екшн.

Зв'язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення

Мета та завдання дослідження. Метою бакалаврської дипломної роботи є підвищення тренувальних можливостей гравців та реакції користувача за допомогою розробки та використання 2d-гри у жанрі екшн платформера, що дозволяє проводити тренування в ігровій формі.

Основними задачами роботи є:

– розробка методу та моделі ігрової програми;
– розробка алгоритму взаємодії головного персонажа з ігровим світом;
– розробка алгоритму поведінки неігрових персонажів;
– розробка інтуїтивно зрозумілого користувацького інтерфейсу мобільної гри;
– розробка мобільної 2d-гри «Dark Medieval»;
– тестування створеної мобільної гри.

Об'єкт дослідження – процес розробки мобільних 2d-ігор.

Предмет дослідження – засоби реалізації мобільних 2d-ігор у жанрі екшн платформеру.

Методи дослідження. Для реалізації поставлених задач були використані такі методи дослідження:

– теорія алгоритмів для побудови алгоритмів функціонування ігрових систем;
– методи та технології розробки мобільних ігор для реалізація 2d-гри у жанрі екшн платформера;
– методи створення графічних зображень для розробки ігрових рівнів;

– методи тестування для перевірки працездатності створеного програмного продукту.

Наукова новизна отриманих результатів.

- Подальшого розвитку дістав метод реалізації ігрової системи, який, на відміну від існуючих, пропонує користувачу зміну поведінки ворогів залежно від поточної ігрової стратегії користувача з забезпеченням різноманітності ігрового процесу, що адаптує складність гри до вміння конкретного гравця і робить проходження кожного рівня не схожим до попереднього.
- Подальшого розвитку отримала модель ігрового процесу, яка, на відміну від існуючих, зосереджена на компонентах, розроблених за допомогою логічного мислення, і забезпечує розширену складність гри за допомогою системи підказок, що дозволяє налаштувати ігрове середовище під конкретних користувачів і реалізує комфортне проходження гри.

Практична цінність отриманих результатів. Практична цінність полягає у кінцевій реалізації мобільної 2d-гри «Dark Medieval» у жанрі екшн платформера.

Особистий внесок здобувача. Усі наукові результати, викладені у бакалаврській дипломній роботі, отримані автором особисто. У науковій роботі, опублікованій у співавторстві [3], автору належать алгоритми взаємодії головного героя з ігровим світом та алгоритми поведінки ігрових персонажів.

Апробація матеріалів бакалаврської дипломної роботи. Основні положення бакалаврської дипломної роботи доповідалися та обговорювалися на Всеукраїнській науково-технічній конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2022). Секція Інформаційні технології та комп‘ютерна інженерія».

Публікації. Основні результати дослідження опубліковані в науковій роботі – в тезах доповіді на Всеукраїнській науково-технічній конференції

«Молодь в науці: дослідження, проблеми, перспективи (МН-2022). Секція Інформаційні технології та комп‘ютерна інженерія» [3].

Аналіз. У пояснювальній записці до бакалаврської дипломної роботи було розглянуто 4 розділи та було використано 18 літературних джерел.

1 ОБГРУНТУВАННЯ ДОЦІЛЬНОСТІ РОЗРОБКИ

1.1 Аналіз та класифікація мобільних ігор

Існує багато класифікацій відеоігор. Ці класифікації ще називають жанрами. Відеоігри можна розділити на кілька різних типів. Доволі часто їх поділяють на групи, в залежності на якій системі вони граються, наприклад ігри, призначені для комп'ютерних систем чи консолей або ігри, призначені для мобільних телефонів[4].

Ці жанри включають екшн, шутер, пригодницький екшн, пригоди, рольові ігри, симулятори, стратегії, головоломки, карти та гонки. Відеоігри можна розділити на багато різних категорій залежно від того, як ви в них граєте. Ігри-шутери, такі як Call of Duty, сильно відрізняються від ігор-головоломок, таких як Тетріс.

Екшн – це ігри, які зазвичай акцентують увагу на фізичних труднощах, включаючи координацію рук і очей і час реакції. У більшості екшн-ігор гравець зазвичай керує персонажем головного героя. Персонаж повинен орієнтуватися на рівні, збираючи предмети, уникаючи перешкод і борючись з ворогами різними атаками. Гравець виграє гру, закінчивши послідовність рівнів.

Платформер - це жанр відеоігор основною метою яких є переміщення персонажа гравця між точками у відтвореному середовищі. Платформні ігри характеризуються дизайном рівнів із нерівним рельєфом і підвісними платформами різної висоти, що вимагає використання здібностей персонажа гравця, таких як стрибки та лазіння , для навігації в оточенні гравця та досягнення своєї мети [4].

Відеоігри-шутери також перевіряють швидкість і час реакції гравців. Ігри-шутери, як правило, зосереджуються на персонажі, який використовує якусь зброю, як правило, пістолет. Мета шутерів - стріляти в супротивників і виконувати місії.

Екшн-пригодницькі ігри — це відеоігри, що поєднують елементи жанру пригодницьких ігор з різними елементами екшн-ігор. Це, мабуть, найширший і різноманітний жанр в іграх.

Пригодницька гра — це відеогра, в якій гравець бере на себе роль головного героя в інтерактивній історії, керованої дослідженнями та розгадуванням головоломок. Майже всі пригодницькі ігри розроблені для одного гравця.

Рольова гра — це гра, в якій гравці виконують ролі персонажів у вигаданій обстановці. Рольові відеоігри, як правило, покладаються на добре розвинену історію та обстановку, яка поділена на ряд квестів.

Відеоігри-симулятори — це ігри, розроблені для точного моделювання аспектів реальної чи вигаданої реальності. Більшість усіх ігор вписується в цю категорію.

Стратегічні відеоігри зосереджені на ретельному плануванні та вмілому управлінні ресурсами для досягнення мети. Вони також відомі як ігри на мислення[4].

Відеоігри-головоломки вимагають від гравця розгадувати логічні головоломки або навіть переміщатися по складних місцях.

Крім того, відеогра може вписуватися в багато різних жанрів. Чим більше жанрів включає гра, тим краще гравцеві подобається гра.

1.2 Порівняльний аналіз аналогів

Існує досить велика кількість ігор у жанрі екшн платформеру. Проведемо аналіз існуючих ігор та визначимо плюси і недоліки конкурентів.

Apple Knight - це сучасний платформер з точним сенсорним керуванням, плавним рухом та плавною анімацією[5]. Гра має великі рівні наповнені секретами, квестами та здобиччю. Різноманітну кількість ворогів та босів.

Особливості:

- налаштування скінів персонажів, зброї та здібностей;

- домашні улюбленці;
- 6 макетів управління сенсорним екраном, що настроюються;
- підтримка геймпаду;
- досягнення та таблиці лідерів.

На рисунку 1.1 зображена гра «Apple Knight».

Рисунок 1.1 – Мобільна гра «Apple Knight»

Sword Of Xolan - це екшн-платформер гра, яка включає в себе піксельний художній стиль [6]. В даній грі вам доведеться грати за хороброго чоловіка, який боротися за справедливість, незалежно від того, які перешкоди трапляються на його шляху.

Особливості:

- більше 30 різних ворогів, таких як: зомбі, велетні та літаючі істоти;
- 10 унікальних ігрових рівнів;
- налаштування сенсорного управління;
- підтримка контролера;
- Оригінальний саундтрек від Бурак Каракас.

На рисунку 1.2 зображена гра «Sword Of Xolan».

Рисунок 1.2 – Мобільна гра «Sword Of Xolan»

NinjAwesome - ніндзя-гра у стилі екшн-платформеру з піксельним художнів стилем[7]. Гра дозволяє руйнувати смертельні пастки і викрадати монети за допомогою майстерних ударів, прориватися крізь зарослі піксельних рослин та знищувати супротивників кидком сюрикену.

Особливості:

- присутність секретних місій;
- велика кількість різноманітних пасток;
- підтримка контролера.

На рисунку 1.3 зображена гра «NinjAwesome».

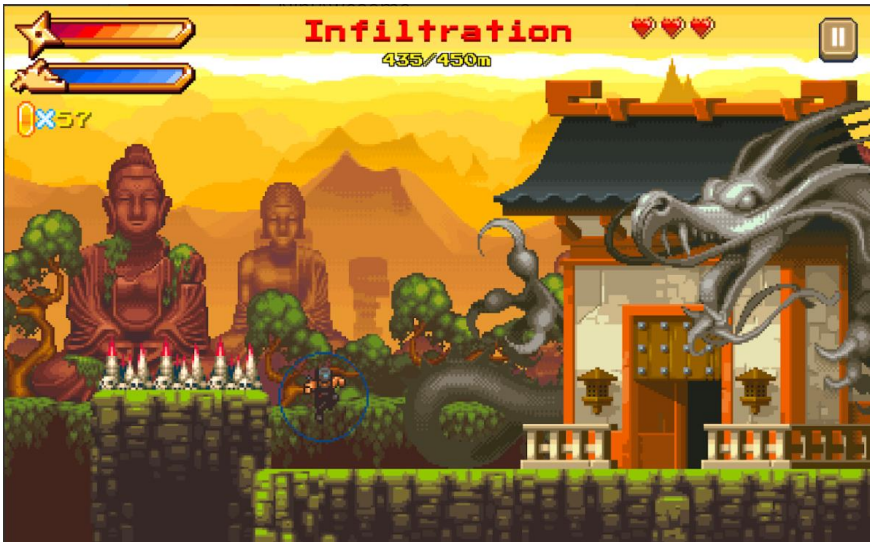

Рисунок 1.3 – Мобільна гра «NinjAwesome»

Oddmar – гра у стилі пригодницького екшну. Епічна історія про вікінгів, подана у вигляді анімованого коміксу [8]. Гра містить велику кількість різноманітних рівнів та головоломок побудованих на законах фізики.

Особливості:

- доволі пророблена графіка гри;
- велика кількість рівнів;
- присутність законів фізики;
- підтримка контролера.
- досягнення та таблиці лідерів.

На рисунку 1.4 зображена гра «Oddmar».

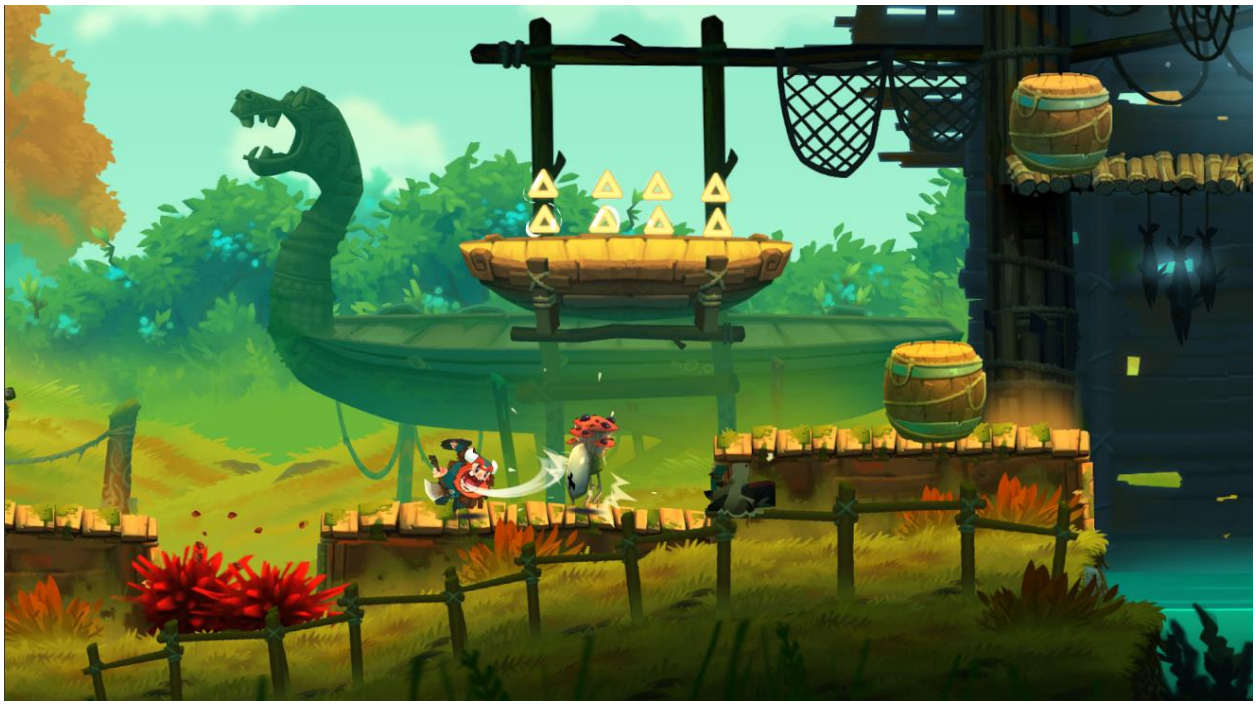

Рисунок 1.4 – Мобільна гра «Oddmar»

Виконавши аналіз схожих мобільних 2d-ігор у жанрі екшн, можна скласти порівняльну таблицю (табл. 1.1), яка показує головні відмінності порівнювальних мобільних ігор.

Таблиця 1.1 – Порівняння аналогів

Функції	Apple Knight	Sword Of Xolan	NinjAwesome	Oddmar	Dark Medieval
Присутність різноманітних пасток	-	+	+	-	+
Вороги, що переслідують головного персонажа	+	-	+	+	+
Не потрібно використовувати Інтернет з'єднання	+	+	+	+	+
Відсутність внутрішніх покупок	-	-	-	-	+
Гра є безкоштовною	+	+	+	+	+

Переглянувши результати порівняння в таблиці 1.1, можна зробити висновок, що всі мобільні додатки мають доволі широкий функціонал, проте мають і недоліки, тому не задовільняють усім вимогам сучасного екшн-платформеру.

Отже, суттєвими перевагами розробки гри «Dark Medieval», порівняно з аналогами, є відсутність внутрішніх покупок та різноманітність ігрового процесу.

1.3 Аналіз технологій та мов програмування

Розробники ігор використовують кілька мов програмування, іноді більше однієї одночасно. Правильна мова залежить від проекту, функцій які вам потрібні, а також досвіду програміста

C++ є основою більшості основних консольних ігор Microsoft Windows. Це мова об’єктно-орієнтованого програмування (ООП), це означає, що вона організовує код у самостійні блоки, які можна використовувати повторно. Ці об’єкти призначені для багаторазового використання та переміщення, тому можна кодувати складні ігри, не створюючи все з нуля[9].

C++ — це потужна мова низького рівня, створена для машиночитання. Це складна мова для початківців у порівнянні з такими мовами, але вона варта клопотів для розробників ігор, які прагнуть максимізувати свою творчість.

C++ — популярна мова програмування відеоігор, оскільки вона швидка, потужна та гнучка:

- Це швидка мова. Геймери очікують високої продуктивності та мінімальних затримок або взагалі без затримок у якісній грі. На жаль, відеоігри схильні до високого навантаження на сервер, що може призвести до затримки, якщо код перетворюється занадто повільно. C++ швидко перетворюється, навіть якщо одночасно виконується кілька завдань.
- Працює в безпосередній близькості від обладнання. Відеоігри покладаються на апаратне забезпечення для функцій керування. C++ працює близько до цього обладнання, тому двигуни працюють краще з меншою ймовірністю зупинки.
- Працює кросплатформено. C++ дозволяє випускати проекти на кількох операційних системах та ігрових платформах.

- Має високопродуктивні інструменти. C++ має багатий набір бібліотек з відкритим кодом , які позбавляють від роботи з кодування. Існують бібліотеки C++ для багатьох різних аспектів розробки ігор, від графічного дизайну до налагодження.
- Сумісна з іншими мовами програмування ігор. C++ дає вам можливість використовувати такі мови, як C# або JavaScript, щоб покращити певні аспекти гри, як-от графіку чи аудіо.

C# («C-sharp») є простішою та доступнішою мовою, ніж C++. Це мова високого рівня, розроблена для розуміння людьми. Його синтаксис сприяє повторному використанню коду, що робить його ефективним, а його компілятори зменшують кількість помилок під час виконання, попереджаючи програміста заздалегідь[10].

C# лідирує як мова сценаріїв відеоігор. Мови сценаріїв викладають механіку гри та говорять комп’ютеру, що робити під час гри. Якщо є потреба, щоб гравець бачив певну графіку або повідомлення, коли його персонаж бере предмет, сценарії забезпечують це.

C# є мовою за замовчуванням для Unity, популярного ігрового движка. 61% розробників у 2021 році вибрали Unity , що зробило його найкращим двигуном року. Це потужний центр багатьох популярних мобільних ігор, зокрема Pokémon Go і Temple Run 2.

C# також є внутрішньою мовою. У контексті веб-розробки розробники використовують C# для спілкування із сервером, а не з користувачем. Коли гравець виконує дію в багатокористувацькій грі, C# повідомляє серверу, як інтерпретувати цю дію[10].

Якщо є потреба створювати гру з нуля, то потрібно обирати C++. Він пропонує:

- більший контроль над обладнанням;
- параметри налаштування для візуалізації графіки;
- краще управління пам'яттю.

Якщо є потреба створювати сценарії гри, використовуючи наявні фреймворки та движки, доцільно використовувати C#. Такий підхід дає можливість отримати:

- швидше вивчення основи розробки відеоігор;
- можливість кодувати в популярних движках, таких як Unity;
- попередження про помилки, якщо зробили щось не так, що дасть змогу швидше навчатися та досягати успіху.

1.4 Варіантний аналіз та обґрунтування вибору технологій і мов програмування

Для написання мобільної 2d-гри було обрано мову програмування C#, оскільки ця мова є простішою і доступнішою ніж C++ та дозволяє створювати ігри за допомогою сценаріїв.

Особливості мови програмування C#[10]:

– підтримку інкапсуляції, наслідування і поліморфізму;

– підтримку компонентів;

– використання «збору сміття»;

– чітку типізацію змінних;

– автоматичну ініціалізацію змінних;

– використання обробки виключень;

– можливість перевантаження операторів.

В якості середовища програмування було обрано Unity. Unity є найпопулярнішим ігровим движком у світі[11]. Він об’єднує масу функцій і є достатньо гнучким, щоб створити практично будь-яку гру, яку можна уявити.

Завдяки неперевершеним міжплатформним можливостям Unity користується популярністю як у одиноких розробників хобі, так і у студій.

Unity набув популярності завдяки API сценаріям C# і вбудованій інтеграції Visual Studio.

Основні переваги Unity:

– зрозумілий і доступний інтерфейс;

– можливість створення 2D ігор;

– вбудована підтримка мережі;

– підтримка імпорту великої кількості форматів;

– можливість тестування гри безпосередньо в редакторі.

Отже, вибір C# і Unity є найкращим для створення 2d-гри «Dark Medieval», оскільки дана мова програмування має достатній перелік переваг порівняно з іншими та дозволить створити якісний програмний продукт.

1.5 Постановка задач

Завданням бакалаврської дипломної роботи є розробка мобільної 2d-гри «Dark Medieval» у жанрі екшн-платформер.

Гра буде створюватися з використанням мови програмування C# на платформі Unity для покращення навичок програмування та вивчення техніки розробки мобільних ігор. Інтерфейс гри необхідно розробляти на платформі Unity.

Для зручності використання буде розроблено меню з декількома пунктами, додавши музичний супровід і можливість перервати гру, а потім перейти в головне меню.

Гра має кілька рівнів, які ускладняються по мірі проходження гри, що дозволить користувачу отримувати нові, більш складні виклики, які будуть розвивати його логіку та реакцію.

При невдачі в проходженні гри потрібно вивести відповідне повідомлення і забезпечити можливість почати гру повторно.

«Dark Medieval» – це гра у жанрі екшн-платформер, дії якої відбуваються у вигаданому середньовіччі.

Завдання гри – знайти ключ, що відкриває двері для завершення рівню, попутно проходячи різні пастки та долаючи ворогів, які будуть намагатися знищити головного персонажа.

Для зручності проходження гри поставленні наступні задачі:

- зупинка гри, з виходом у головне меню в будь-який момент гри;
- використання підказок;
- рухомі платформи, які допомагають гравцю долати пастки та перешкоди;
- можливість обрати будь-який рівень з раніше пройдених;
- можливість додати музичний супровід.

Для запуску програми апаратне забезпечення повинне відповідати мінімальним вимогам:

- смартфон з процесором Spreadtrum SC7731E;
- 1 Гб оперативної пам’яті;
- відеоядро ARM Mali-T820;
- сенсорний дисплей;
- ОС Android;

Розмір дискового простору, що займає програма, 58,59 Мб.

1.6 Висновки

У першому розділі бакалаврської дипломної роботи проведено аналіз предметної області створення мобільних 2d-ігор у жанрі екшн платформеру. Поставлено задачу розробити ігрову систему «Dark Medieval» розважального характеру, яка допоможе користувачу тренувати логіку та реакцію при проходженні складних рівнів.

Проаналізовавши аналоги, було створено вимоги до мобільної гри «Dark Medieval» та сформовано перелік задач, які необхідно реалізувати.

В якості мови програмування було обрано об’єктно-орієнтовану мову високого рівня C#, а середовищем програмування – інструмент для створення ігор Unity 3D.

2 РОЗРОБКА МЕТОДУ І МОДЕЛІ РОБОТИ ІГРОВОЇ СИСТЕМИ

2.1 Аналіз завдання

Розробка бакалаврської дипломної роботи розпочинається з визначення завдання, тобто чіткого формулювання вимог до мобільної гри, що розробляється. Для цього потрібно сформулювати вимоги до програми та її інтерфейсів, визначити особливості, функції, загальні принципи побудови програмних продуктів.

Мобільна 2d-гра «Dark Medieval» у жанрі екшн платформеру несе розважальний характер і допомагає не лише весело провести свій час, а й прийняти виклик цікавим та важким рівням, що допоможуть розвинути користувачу логіку і реакцію.

Для розробки ігрової системи необхідно:

- розробити користувацький інтерфейс;
- розробити методи і моделі реалізації мобільної гри у жанрі екшн-платформер;
- розробити графічні матеріали для ігрового додатку;
- спроектувати користувацький інтерфейс;
- розробити основні програмні модулі додатку;
- протестувати розроблений програмний продукт.

2.2 Розробка стратегії розвитку швидкості реакції

У багатьох повсякденних ситуаціях швидкість має основне значення. Однак швидкі рішення зазвичай означають більше помилок. До цього дня залишається невідомим, чи можна скоротити час реакції за допомогою відповідного навчання в межах однієї людини, для виконання цілого ряду завдань і без шкоди для точності [12]. Саме гра в екшн-ігри значно скорочує час реакції без шкоди для точності. Важливо, що це збільшення швидкості

спостерігається в різних завданнях, крім ігрових ситуацій. Тому відеоігри можуть забезпечити ефективний режим тренувань, щоб викликати загальне прискорення часу реакції сприйняття без зниження точності виконання.

Гра в екшн-ігри потребує швидкої обробки сенсорної інформації та швидких дій, змушуючи гравців приймати рішення та відповідати набагато швидше, ніж це типово в повсякденному житті. Під час гри затримки в обробці часто мають серйозні наслідки, надаючи гравцям великий стимул до збільшення швидкості. Відповідно, є свідчення того, що завзяті гравці легше реагують на навколишнє середовище.

Дослідження показують, що гравці у відеоігри розвивають підвищену чутливість до того, що відбувається навколо них, і ця перевага не тільки покращує їх у відеоіграх, але й покращує широкий спектр загальних навичок, які можуть допомогти в повсякденній діяльності, як-от багатозадачність. , водіння, читання дрібним шрифтом, відстеження друзів у натовпі та навігація містом.

Дослідники перевірили десятки підлітків у віці від 18 до 25 років, які зазвичай не гравці у відеоігри [12]. Вони поділили випробовуваних на дві групи. Одна група грала 50 годин у швидкі екшн-ігри Call of Duty 2 і Unreal Tournament , а інша група грала 50 годин у повільну стратегічну гру The Sims2.

Після цього періоду навчання всім випробовуваним було запропоновано швидко прийняти рішення у кількох завданнях, розроблених дослідниками. Під час виконання завдань учасники повинні були подивитися на екран, проаналізувати те, що відбувається, і відповісти на просте запитання про дію за якомога коротший час.

Щоб переконатися, що ефект не обмежується лише візуальним сприйняттям, учасникам також було запропоновано виконати аналогічне завдання, яке було суто слуховим.

Гравці екшн-ігор на 25 відсотків швидше підходили до висновків і відповідали на стільки ж запитань правильно, як і їхні колеги, які грали в стратегічні ігри [12].

Нейронне моделювання проливає світло на те, чому екшн-геймери розширили можливості прийняття рішень.

Люди приймають рішення на основі ймовірностей, які вони постійно прораховують і доопрацьовують у своїй голові. Цей процес називається імовірнісним висновком.

Мозок безперервно накопичує невеликі шматочки візуальної або слухової інформації, коли людина оглядає сцену, зрештою збираючи достатньо, щоб людина могла прийняти те, що вона вважає точним рішенням.

Дослідники виявили, що мозок гравців екшн-ігор ефективніше збирає візуальну та слухову інформацію, а отже, досягає необхідного порога інформації, необхідної їм для прийняття рішення, ніж гравців, які не є гравцями[12].

Тож комп’ютерні ігри розвивають реакції та навчають швидких тактичних рішень у критичних ситуаціях, тому для розробки гри «Dark Medieval» було обрано найактуальніший вид гри «Екшн». Гра заснована на принципах зосередження уваги гравця на діях по сюжетній лінії та подолання різноманітних перешкод і ворогів. Гра буде мати обмежену території по якій буде переміщуватися гравець. У гравця буде п’ять життів, якщо під час проходження рівню він втратить їх усіх, то буде повинний проходити рівень знову від початку. Для проходження рівню потрібно зайти схований ключ, що відкриває двері в кінці рівня.

Послідовність процесу навчання:

1. Обмежена кількість життів головного персонажа.
2. Аналіз складності рівня.
3. Пошук рішень.
4. Побудова стратегії проходження рівня.

5. Кожен наступний рівень збільшує кількість перешкод та ворогів, яких потрібно пройти, щоб успішно завершити рівень.

Головне завдання гравця – подолати усі перешкоди, знищити усіх ворогів та знайти ключ для виходу з рівня.

Послідовність виконання власних дій:

1. Аналіз рівня, перешкод та ворогів.
2. Розрахунок стратегії проходження рівня, подолання перешкод та знищення ворогів.
3. Реалізація стратегії:
    a. переміщення головного персонажа відбувається у горизонтальному та вертикальному просторі;
    b. перешкоди проходяться за допомогою стрибків та пересувань на рухомих плитах;
    c. знищення ворога відбувається за допомогою атак мечем головного персонажа, по кожному ворогу потрібно нанести різну кількість ударів для знищення.
4. Перейти до наступного рівня.

У разі втрати усіх життів у головного персонажа рівень буде провалений, висвітиться відповідне меню з інформацією про поразку та пропонуванням розпочати рівень заново.

2.3 Розробка методу поведінки ворогів у ігровому середовищі

Для процесу взаємодії гравця з ігровим середовищем було вирішено розробити метод поведінки ворогів. Даний метод передбачає три стадії поведінки ворога:

1. Патрулювання. Ворог патрулює задану йому місцевість ігрового середовища, із заданою швидкістю рухаючись з однієї точки до іншої. Рух здійснюється за допомогою компонента «Transform», який дозволяє рухати об’єкти за допомогою двовимірного вектору. Під час ініціалізації ворога задаються параметри швидкості, радіус видимості, радіус атаки, відстань

руху від точки та сама точка, яка є окремим об’єктом на сцені. Одна й та сама точка може бути вказана в декількох ворогів.

2. Переслідування. За допомогою методу «Distance» аналізується відстань до головного персонажу, якщо відстань менше заданої видимості, то ворог розпочинає переслідування і рухається в сторону персонажа. Під час руху ворог за допомогою методу «OverlapCircleAll» записує у масив список усіх об’єктів, що знаходяться в радіусі його атаки, далі за допомогою циклу «foreach» відбувається проходження по масиву об’єктів та перевіряється назва, якщо назва об’єкту «Player», то ворог наносить удар. Щоб ворог не міг нескінченно наносити удари, за допомогою класу «Time» відбувається перезарядка.

3. Повернення на позицію. Якщо під час аналізу відстані за допомогою методу «Distance» головний персонаж вийде з радіусу видимості ворога, то ворог розпочне рух у бік точки патрулювання.

Блок-схема алгоритму методу поведінки ворогів у ігровому середовищі зображена на рисунку 2.1.

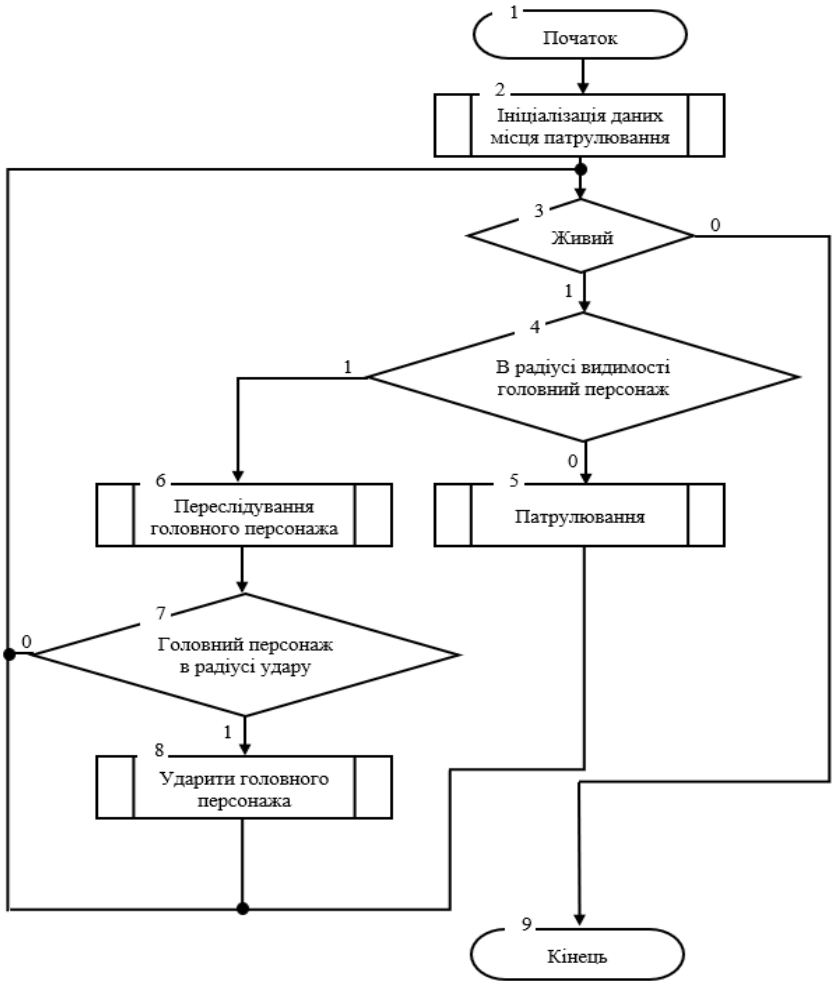

Рисунок 2.1 – Блок-схема алгоритму методу поведінки ворогів у ігровому середовищі

2.4 Розробка моделі роботи мобільної екшн гри

Модель реалізації мобільної 2d-гри «Dark Medieval» розпочинається з завантаженням з завантаженням головного меню, яке надає наступний вибір дій: «Нова гра», «Рівні», «Вихід з додатку». Компонент «Нова гра» надає можливість завантажити перший рівень та перейти до ігрової сцени цього рівня. Вихід з ігрової сцени здійснюється шляхом завершення рівня і переходом до сцени «Результат гри» або шляхом дострокового завершення гри через меню паузи та переходом до компоненту «Меню». Компонент меню «Вибір рівня» дозволяє обрати рівень та перейти до ігрової сцени або повернутися до компоненту «Меню». Компонент «Вихід з додатку» здійснює завершення роботи мобільного додатку. На рисунку 2.2 наведено загальний алгоритм роботи мобільної гри «Dark Medieval».

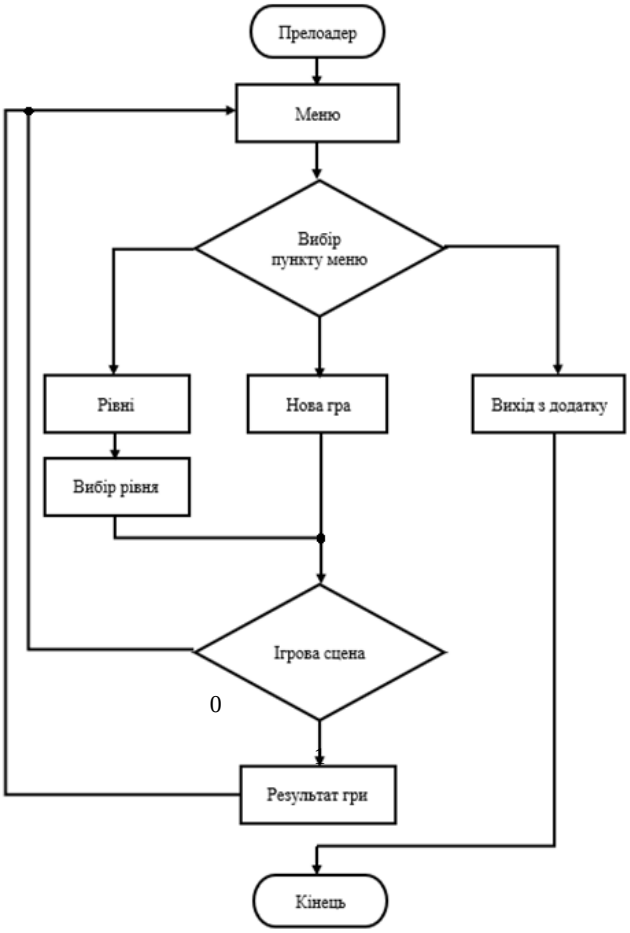

Рисунок 2.2 – Загальний алгоритм роботи мобільної 2d-гри «Dark Medieval»

Перед входом до ігрової сцени активується модуль налаштування рівня. Далі ігрова сцена здійснює обмін даними з модулем обробки руху головного персонажа, який в свою чергу обмінюється даними з модулем обробки дій гравця. Після проходження рівня здійснюється перехід до модуля нарахування результату. На рисунку 2.3 наведено модель реалізації ігрової системи.

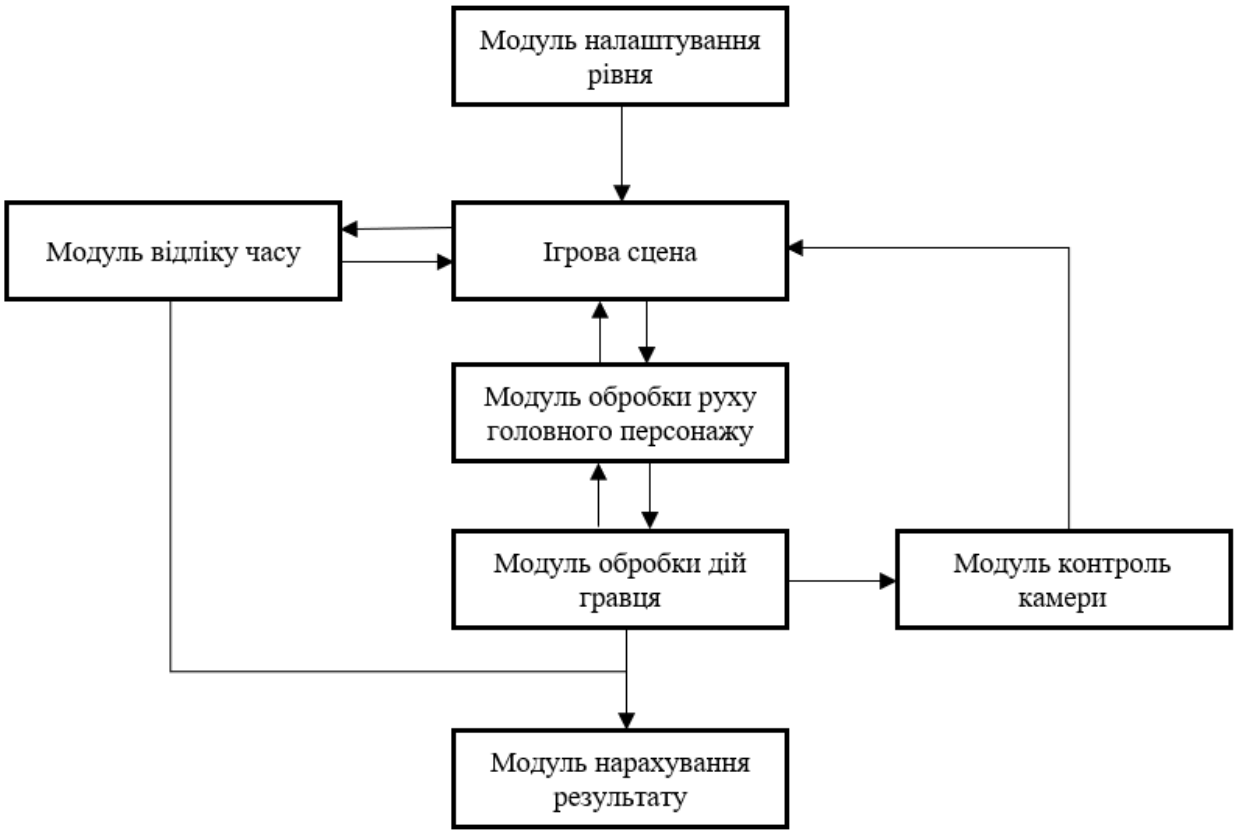

Рисунок 2.3 - Модель реалізації ігрової системи

2.5 Висновки

У другому розділі розглянуто стратегії розвитку швидкості реакції у процесі прийняття тактичних рішень.

Розроблено метод поведінки ворогів у ігровому середовищі.

Розроблено моделі реалізації мобільної 2d-гри у жанрі екшн платформеру, що підвищить можливості популяризації розробленого програмного продукту.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ІГРОВОГО ДОДАТКУ

3.1 Проектування інтерфейсу користувача

Інтерфейс користувача (UI) — це точка, на якій люди взаємодіють з комп'ютером, веб-сайтом або програмою. Метою ефективного інтерфейсу є зробити роботу користувача простою та інтуїтивно зрозумілою, вимагаючи мінімальних зусиль з боку користувача для отримання максимального бажаного результату[13].

Інтерфейс користувача створюється у вигляді шарів взаємодії, які звертаються до органів чуття людини (зір, дотик, слух тощо). Вони включають обидва пристрої введення, такі як клавіатура, миша, трекпад, мікрофон, сенсорний екран, сканер відбитків пальців, електронне перо та камера, а також пристрої виведення, такі як монітори, колонки та принтери.

Розроблюваний ігровий додаток призначений для використання на мобільних пристроях з сенсорним дисплеєм.

Для виконання проектування інтерфейсу користувача складено список необхідних сцен для відображення елементів інформативного та контролюючого характеру:

1. Сцена «Меню», що складається із кнопок «Нова гра», «Рівні» та «Вихід».
2. Сцена «Рівні», що складається зі списку рівнів.
3. Сцена «Гра», що відображає ігровий процес та містить наступні інтерфейси користувача:
    a. меню паузи;
    b. меню смерті персонажу.
4. Сцена «Результат», що відображає результат гри.

На основі висунутих вимог розроблено схематичні зображення користувацького інтерфейсу ігрового додатку.

Схему сцени «Меню» наведено на рисунку 3.1.

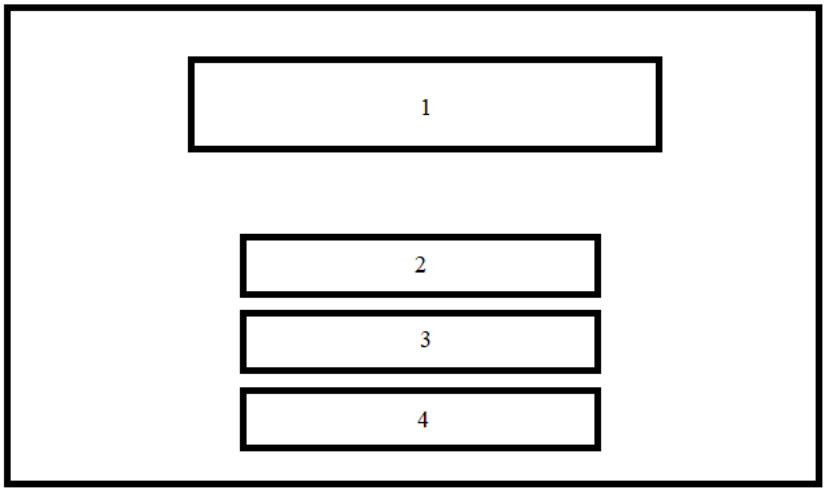

Рисунок 3.1 – Схема інтерфейсу користувача сцени «Меню»

Опис елементів схеми відповідно до їх нумерації на рисунку 3.1:

1. Логотип, який відображає назву гри «Dark Medieval».

2. Елемент головного меню, кнопка «Нова гра», активація якої здійснює перехід на сцену «Гра».

3. Елемент головного меню, кнопка «Рівні», активація якої здійснює перехід на сцену «Рівні».

4. Елемент головного меню, кнопка «Вихід», активація якої здійснює завершення роботи додатку.

Сцена «Рівні» надає можливість обрати рівень та перейти до сцени «Гра». Схема цієї сцени зображена на рисунку 3.2.

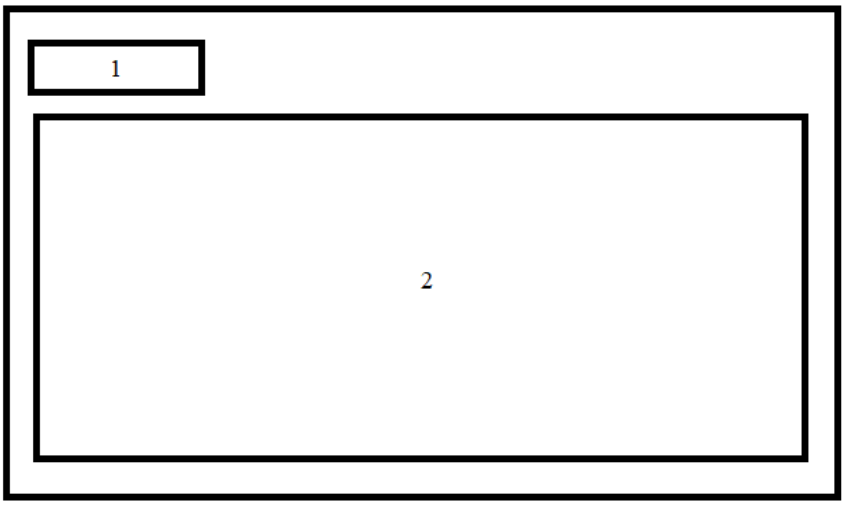

Рисунок 3.2 – Схема інтерфейсу користувача сцени «Рівні»

Опис елементів схеми відповідно до їх нумерації на рисунку 3.2:

1. Кнопка «Назад», активація якої здійснює повернення на сцену «Меню».

2. Елемент вибору рівня складається із пронумерованого списку завдань. Активація елементу списку здійснює перехід на сцену «Гра».

Схема сцени «Гра» зображена на рисунку 3.3.

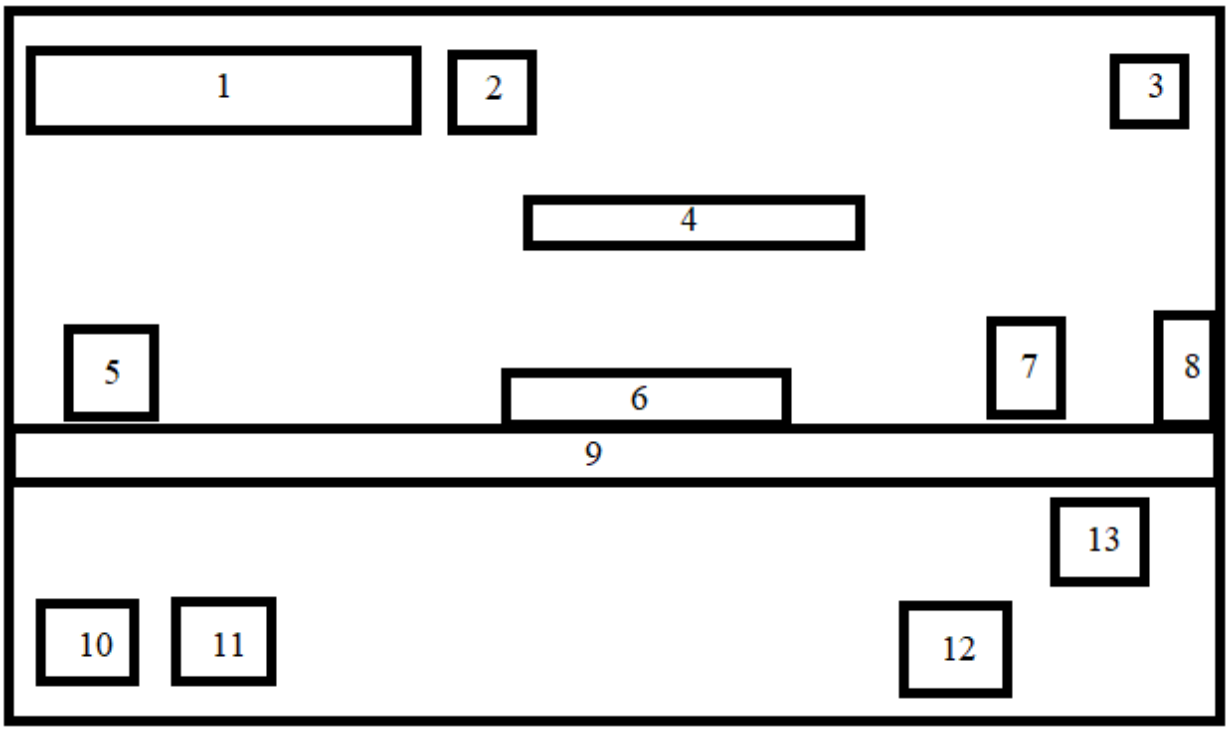

Рисунок 3.3 – Схема інтерфейсу користувача сцени «Гра»

Опис елементів схеми відповідно до їх нумерації на рисунку 3.3:

1. Масив зображень, що відображаються кількість життів, що залишилися у головного персонажа.

2. Елемент зображення, що відображає наявність ключа у головного персонажа.

3. Кнопка «Пауза», взаємодія з якою, викликає меню паузи.

4. Об’єкт «Рухома платформа», який має статичну швидкість і переміщається горизонтально по заданій траєкторії.

5. Об’єкт «Головний персонаж», який має статичну швидкість, може переміщуватися горизонтально та вертикально, здійснювати атаки по ворогам.

6. Об’єкт «Пастка», віднімає одне життя головному персонажу при їх зіткненні.

7. Об’єкт «Ворог», який має статичну швидкість та переміщається горизонтально. Веде патрулювання заданої ділянки, якщо помічає головного персонажа у заданій зоні видимості, то починає переслідувати його, при можливості також атакує головного персонажа, від чого той втрачає одне життя. Якщо головний персонаж виходить з зони видимості, об’єкт «Ворог» повертається до заданої зони патрулювання.

8. Об’єкт «Закінчення рівня», слугує точкою, яка дозволяє закінчити рівень при наявності ключа у головного персонажа, якщо ключа не буде, то користувач отримає підказку.

9. Ігрова поверхня по якій рухається головний персонаж.

10. Кнопка «Ліворуч», взаємодія з якою, переміщую головного персонажа ліворуч.

11. Кнопка «Праворуч», взаємодія з якою, переміщую головного персонажа праворуч.

12. Кнопка «Стрибок», після взаємодії з якою, головний персонаж виконує стрибок.

13. Кнопка «Атака», після взаємодії з якою, головний персонаж виконує атаку.

Також на сцені «Гра» присутні інтерфейси меню паузи та відображення смерті персонажу. При натисканні кнопки «Пауза» на ігровій сцені зупиняються будь-які дії та заморожується час, після чого з’являється меню паузи гри. Схему меню паузи зображено на рисунку 3.4

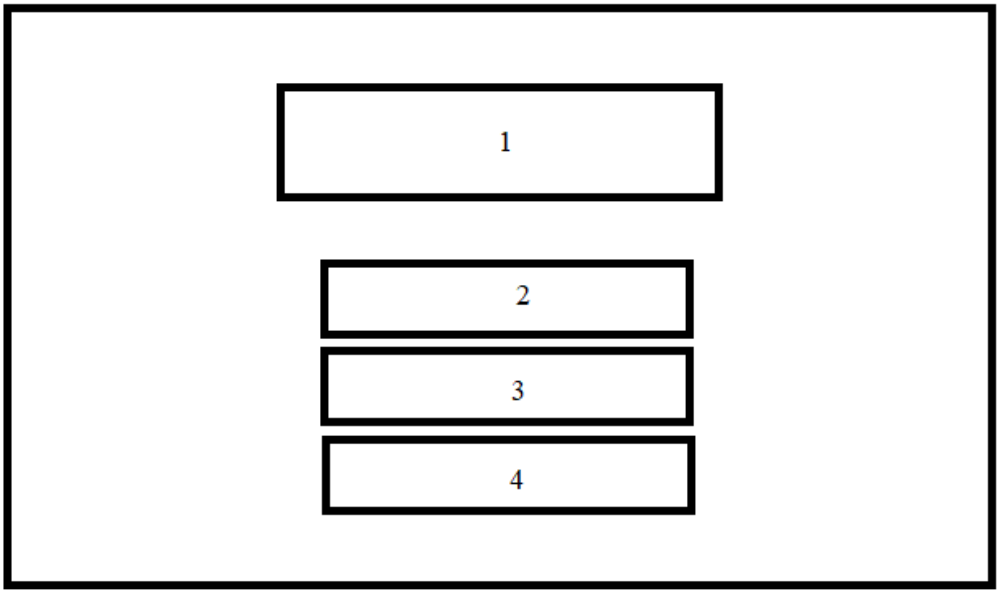

Рисунок 3.4 – Схема інтерфейсу меню паузи

Опис елементів схеми відповідно до їх нумерації на рисунку 3.4:

1. Текстовий елемент з надписом «Пауза».
2. Кнопка «Продовжити гру», яка знімає паузі та продовжує гру.
3. Кнопка «Розпочати рівень заново», яка перезапускає ігрову сцену і переміщує все в початкове положення.
4. Кнопка «Меню», яка завантажує сцену «Меню».

Схему інтерфейсу смерті персонажу зображено на рисунку 3.5.

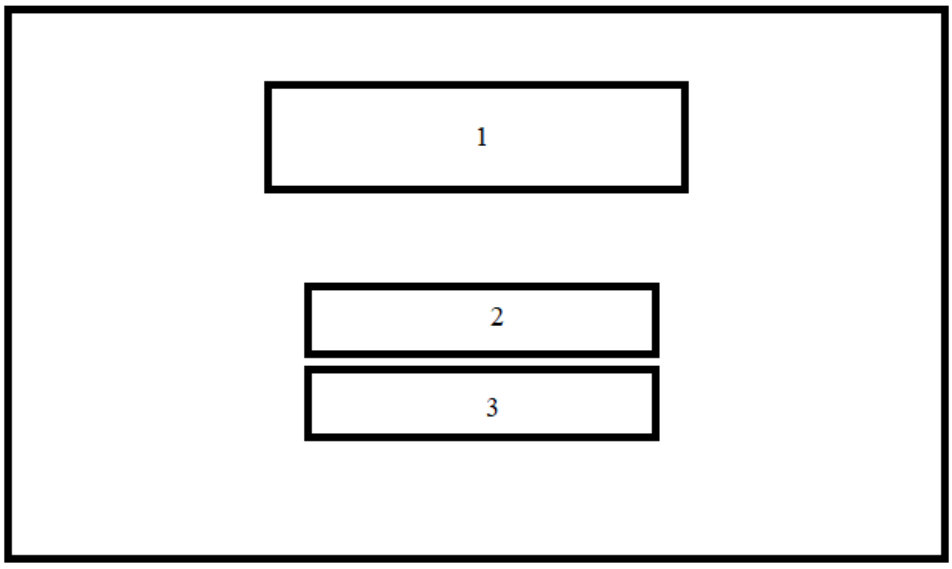

Рисунок 3.5 - Схема інтерфейсу смерті персонажу

Опис елементів схеми відповідно до їх нумерації на рисунку 3.5:

1. Текстовий елемент з надписом «Ви загинули».
2. Кнопка «Розпочати рівень заново», яка перезапускає ігрову сцену і переміщує все в початкове положення.
3. Кнопка «Меню», яка завантажує сцену «Меню».

Схема сцени «Результат» зображена на рисунку 3.6.

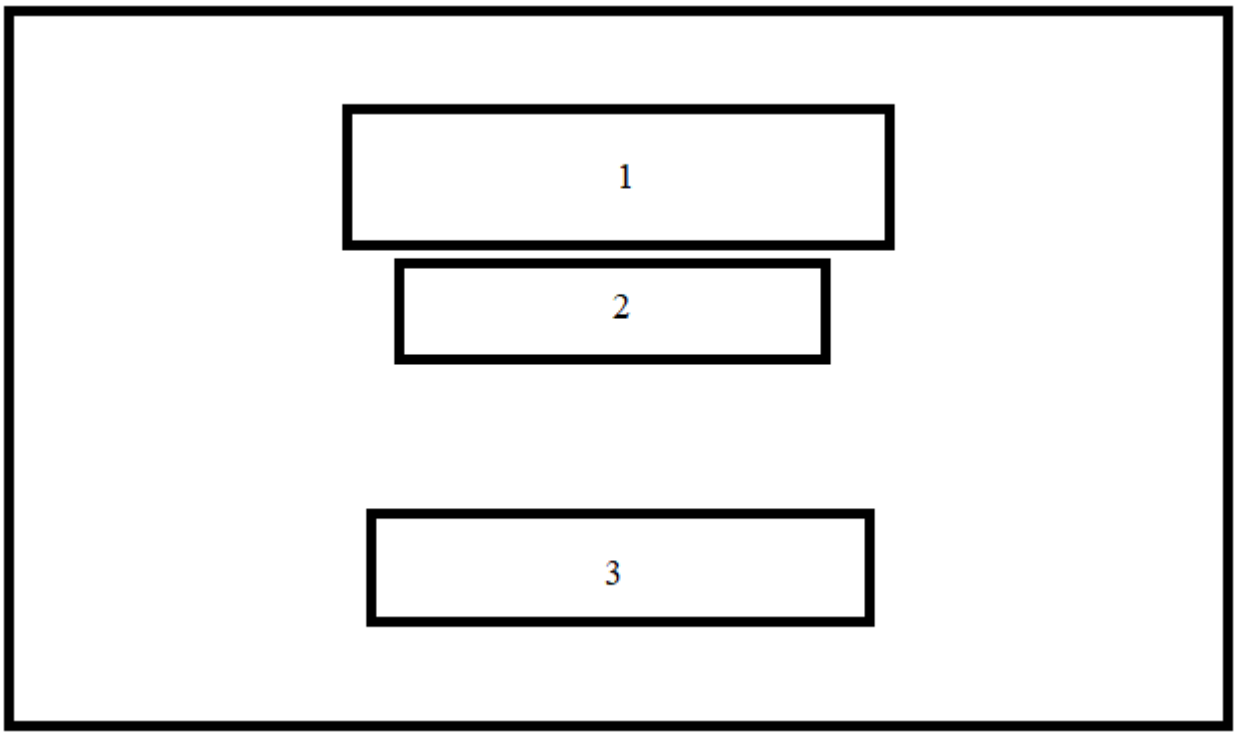

Рисунок 3.6 - Схема інтерфейсу користувача сцени «Результат»

Опис елементів схеми відповідно до їх нумерації на рисунку 3.6:

1. Текстовий елемент з надписом «Рівень пройдено».
2. Текстовий елемент, що відображає час, за який пройдено рівень.
3. Кнопка «Продовжити», яка завантажує сцену «Меню».

Розроблений інтерфейс користувача є макетом, який має бути заповнений графічними матеріалами, щоб забезпечити простоту використання та покращити зрозумілість користувача.

3.2 Розробка графічних матеріалів

Невід'ємною частиною комп'ютерних ігор є візуалізація ігрового процесу, яка виконується шляхом відображення специфічної для гри графіки.

Використання вдалої графіки має вирішальне значення для життєвого циклу комп'ютерної гри, оскільки користувачам подобається не тільки ідея гри, а й ігровий процес, який забезпечується взаємодією з графічним вмістом гри[14].

Графічний редактор — це тип комп'ютерної програми, яка дозволяє користувачеві редагувати та маніпулювати графічними зображеннями різними методами[15].

Adobe Photoshop — професійний графічний редактор для створення та редагування растрових зображень.

Розробка графічних матеріалів починається зі створення, затвердження та перекладу ескізів в електронний вигляд. Створення ескізу є важливою частиною, тому що якщо ви створюєте готовий графічний елемент по одному, це займає більше часу, ніж створення ескізу, і, можливо, доведеться переробляти всю роботу, якщо елемент буде відхилено[14].

Часто відібраних ескізів значно менше, ніж тих, які з якихось причин відхиляються.

Після перетворення ескізів в електронну форму починається розробка графічних елементів. Розглянемо процес розробки графічного зображення на прикладі сцени «Гра».

Дизайн сцени включатиме побудову растрових зображень. Спочатку було створене фонове зображення, яке складається з набору растрових зображень.

На рисунку 3.7 наведено візуалізацію процесу створеної групи фонових об'єктів.

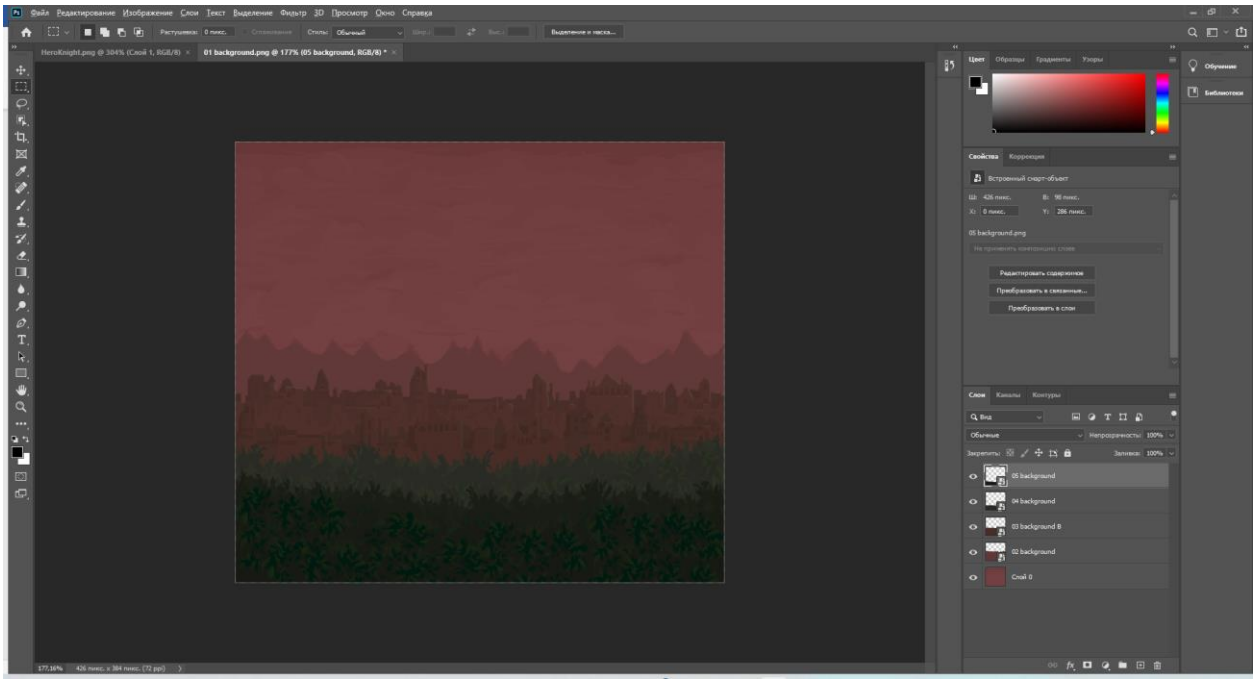

Рисунок 3.7 – Група растрових об’єктів, яка відображує фонове зображення

Використовуючи інструменту олівець, для сцени «Гра» було створено растрові об’єкти користувацького інтерфейсу, елемент, що відображує рівень здоров’я.

На рисунку 3.8 наведено вигляд елементів, які відображують здоров’я.

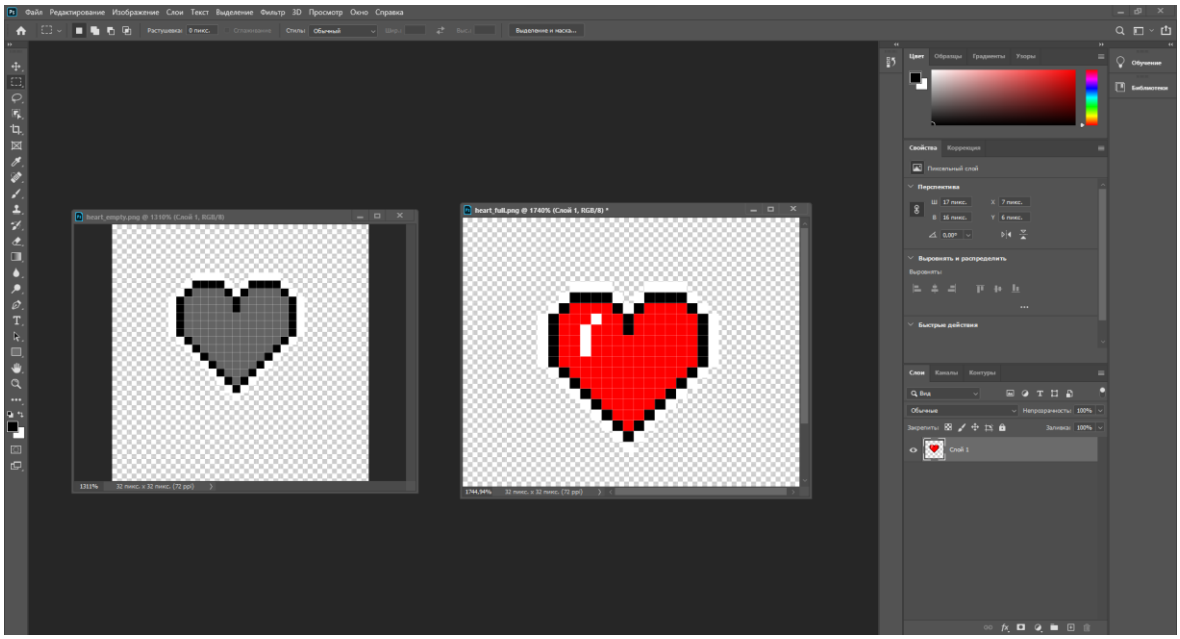

Рисунок 3.8 - Графічний елемент – «Здоров’я»

За допомогою поєднання простих фігур та інструментів графічного редактору Adobe Photoshop було створено набір растрових зображень для побудови рівнів. Завдяки інструменту Tile Palette у Unity, вибираються потрібні елементи та створюється ігрове середовище.

На рисунку 3.9 наведено вигляд елементів, що використовуються для побудови рівнів.

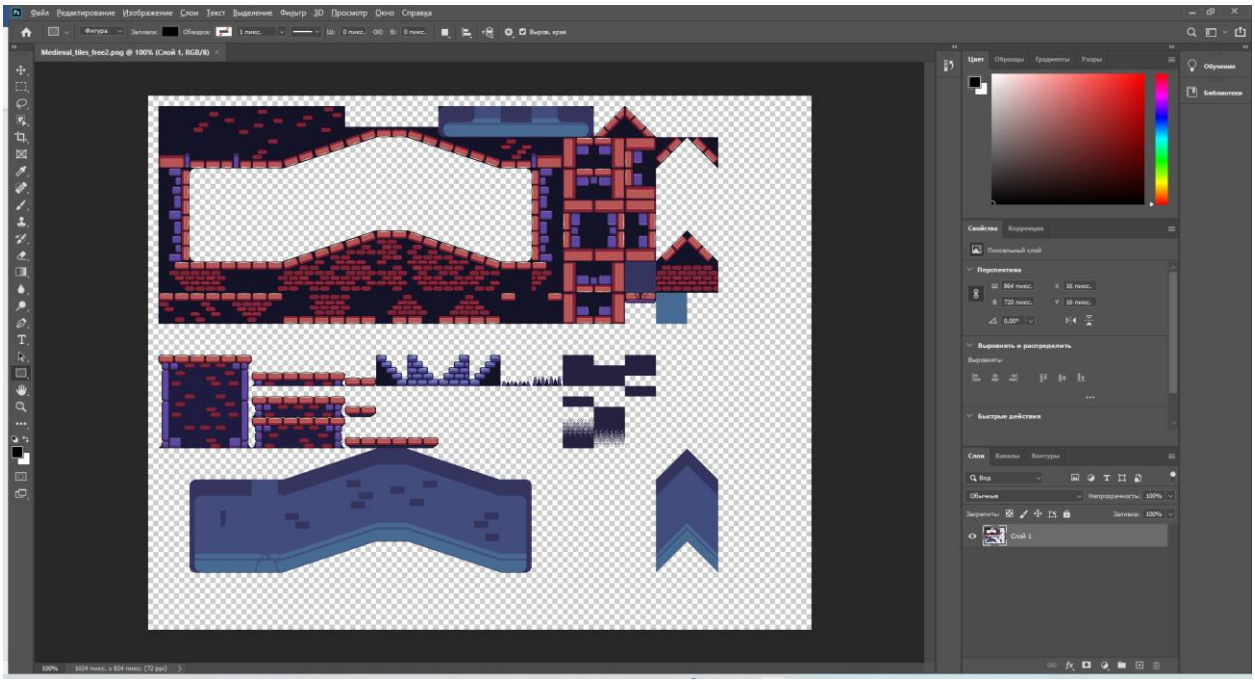

Рисунок 3.9 – Група растрових зображень для побудови рівнів гри

Наступним графічним елементом є головний персонаж, складається із багатьох растрових зображень, що відображають різні елементи його руху. Це робиться для того, щоб за допомогою вбудованих інструментів Unity можна було відтворити анімацію руху персонажа. За схожим принципом було також створено ворогів.

На рисунку 3.10 наведено вигляд елементів, що відображують головного персонажа.

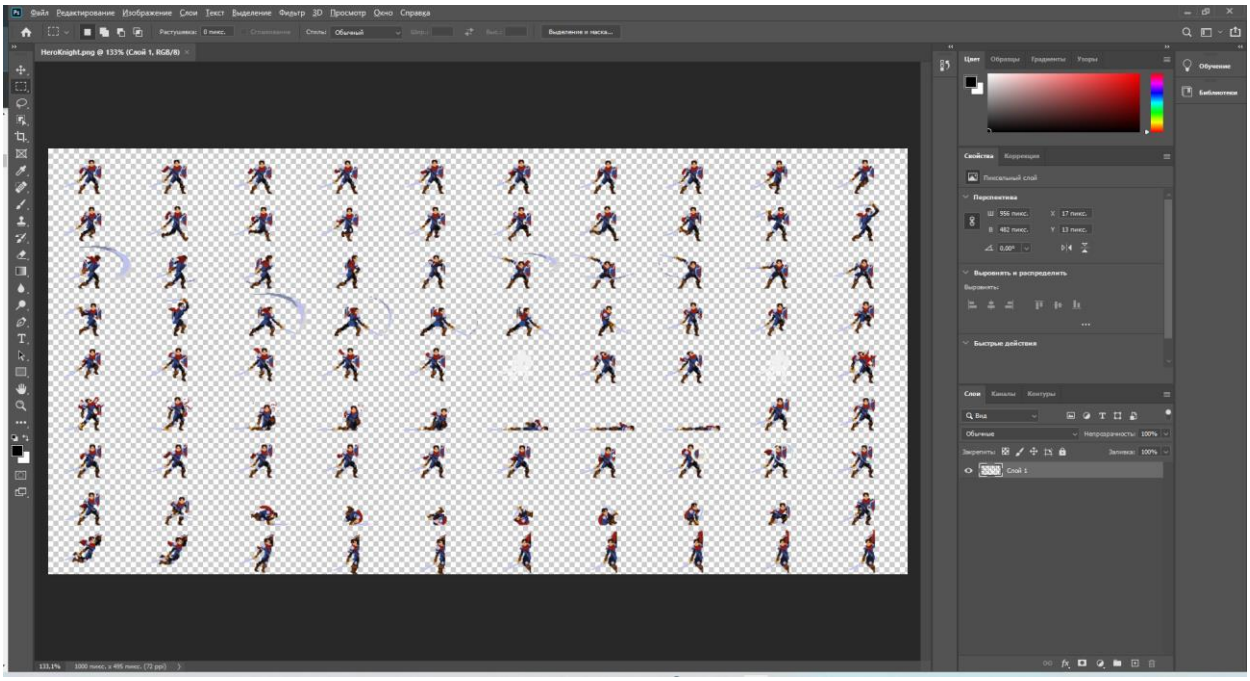

Рисунок 3.10 – Графічний елемент – об’єкт «Головний персонаж»

Після розробки всіх об’єктів потрібно поєднати їх для відображення сцени «Гра». Вигляд поєднаних графічних елементів наведено на рисунку 3.11.

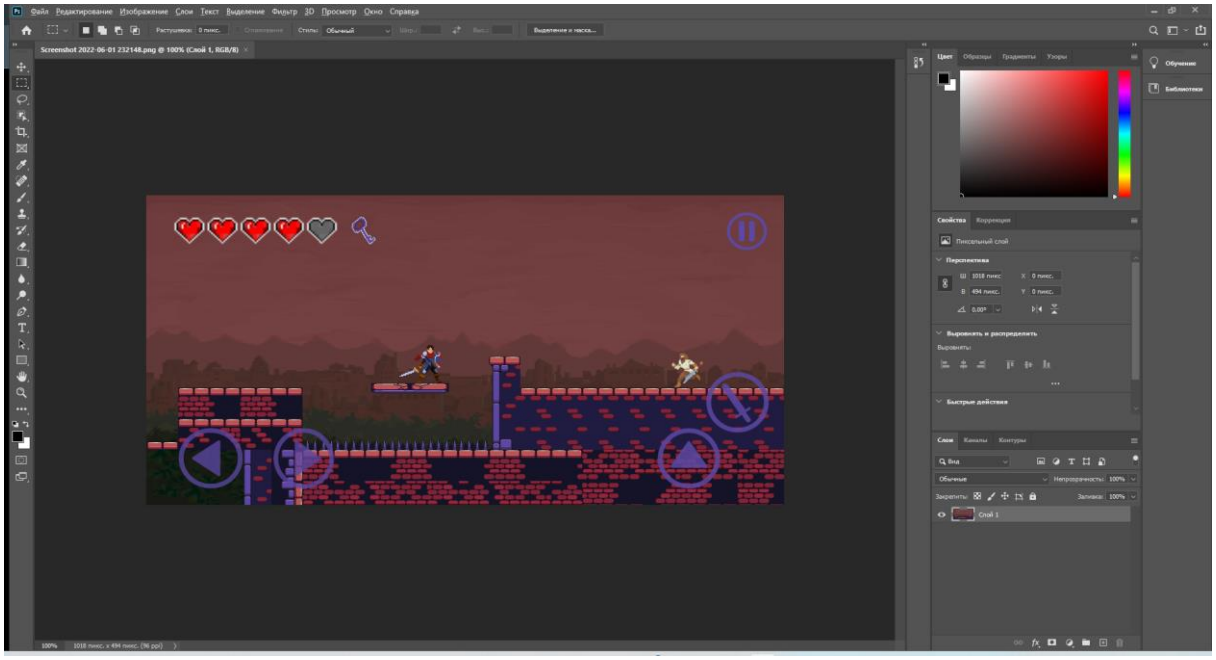

Рисунок 3.11 – Поєднання розроблених графічних елементів та формування вигляду сцени «Гра»

Після цього необхідно зберегти графічні елементи у вигляді, придатному для використання в ігровій системі. Для Unity це графічні формати PNG, BMP, TIF, TGA, JPG, та PSD. Формат PNG дозволяє зберегти прозорість, тому саме він буде використаний у рамках розробки графічних елементів гри «Dark Medieval».

3.3 Розробка основних модулів додатку

Основним модулем мобільної 2d-гри у стилі екшн платформеру є клас Player, який відповідає за поведінку і взаємодію головного персонажу, яким керує користувач, з ігровим середовищем.

Рух головного персонажу здійснюється за допомогою кнопок на екрані смартфону користувача. Реалізація руху відбувається у методах onLeftButtonDown, onRightButtonDown та onJumpButtonDown. Частину коду, що реалізовує рух головного персонажу наведено на рисунках 3.12 та 3.13.

```
public void onLeftButtonDown()
{
    if (speed >= 0f)
    {
        speed = -normalSpeed;
        sprite.flipX = true;
        attackPoint.position = new Vector2(spriteTransform.position.x - 1.2f, spriteTransform.position.y - 0.3f);
    }

        if (!inAir)
    {
        animator.SetInteger("AnimState", 1);
    }
}

0 references
public void onRightButtonDown()
{
    if (speed <= 0f)
    {
        speed = normalSpeed;
        sprite.flipX = false;
        attackPoint.position = new Vector2(spriteTransform.position.x - 0.2f, spriteTransform.position.y - 0.3f);
    }

        if (!inAir)
    {
        animator.SetInteger("AnimState", 1);
    }
}
```

Рисунок 3.12 – Частина коду методів onLeftButtonDown та onRightButtonDown

```
public void onJumpButtonDown()
{
    if (!inAir)
    {
        Jump();
    }
}

2 references
private void Jump()
{
    rb.AddForce(transform.up * jumpForce, ForceMode2D.Impulse);
    inAir = true;
    animator.SetTrigger("Jump");

}
```

Рисунок 3.13 – Частина коду методу onJumpButtonDown

Атака головного персонажу відбувається за допомогою натискання клавіші удару на екрані смартфону, що викликає метод onAttackButtonDown. В даному методі перевіряється перезарядка атаки та викликається метод Attack, у якому реалізований вибір однієї з трьох анімацій атаки, пошук ворогів у радіусі дії меча та нанесення шкоди знайденим ворогам. Частину коду, що реалізовує атаку головного персонажу наведено на рисунках 3.14 та 3.15.

```
public void onAttackButtonDown()
{
    if (Time.time >= nextAttackTime){
            Attack();
            nextAttackTime = Time.time + 1f / attackRate;
        }
}
```

Рисунок 3.14 - Частина коду методу onAttackButtonDown

```
    void Attack()
{
    // Анімація удару
    if (typeAttack == 1)
    {
        animator.SetTrigger("Attack1");
        typeAttack++;
    }
    else if (typeAttack == 2)
    {
        animator.SetTrigger("Attack2");
        typeAttack++;
    }
    else if (typeAttack == 3)
    {
        animator.SetTrigger("Attack3");
        typeAttack = 1;
    }
    // Розпізнавання ворогів у радіусі атаки
    Collider2D[] hitEnemies = Physics2D.OverlapCircleAll(attackPoint.position, attackRange, enemyLayers);

    // Атака ворога
    foreach(Collider2D enemy in hitEnemies)
    {
        Debug.Log("we hit " + enemy.name);
        enemy.GetComponent<Bandit>().TakeDamage();
    }
}
```

Рисунок 3.15 - Частина коду методу Attack

Отримання шкоди головному персонажу та його смерть описано в методах GetDamage, GetDamageFromPrickle та Die. Частину коду, в якому реалізовано отримання шкоди та смерть наведено на рисунках 3.16.

```
private void Die()
{
    alive = false;
    animator.SetTrigger("Death");
    this.enabled = false;
    died.SetActive(true);
}
2 references
private void GetDamage()
{
    health --;
    Debug.Log(health);
        if (health < 1) {
            foreach (var h in hearts)
            {
                h.sprite = deadHeart;
            }
            Die();
        } else animator.SetTrigger("Hurt");

}
1 reference
public void GetDamageFromPrickle()
{
    GetDamage();
    if (health > 0)
        Jump();
}
```

Рисунок 3.16 - Частина коду методів GetDamage, GetDamageFromPrickle та Die

Взаємодія головного персонажу з ігровим середовищем описана в методах OnCollisionEnter2D та OnCollisionExit2D (рис. 3.17). Метод OnCollisionEnter2D перевіряє чи персонаж знаходиться на землі, прив'язує положення персонажу до рухомої платформи та описує взаємодію з ігровим об'єктом ключ. Метод OnCollisionExit2D описує схід персонажу з рухомої платформи. Частину кодів, що описує взаємодію з ігровим середовищем зображена на рисунку 3.17.

```
private void OnCollisionEnter2D(Collision2D collision)
{
    if (collision.gameObject.layer == LayerMask.NameToLayer("Ground")
    || collision.gameObject.layer == LayerMask.NameToLayer("MoveingPlatform"))
    {
        inAir = false;
        animator.SetBool("Grounded", true);
    }

    if (collision.gameObject.layer == LayerMask.NameToLayer("MoveingPlatform"))
    {
        this.transform.parent = collision.transform;
    }

    if (collision.gameObject.layer == LayerMask.NameToLayer("Key"))
    {
        logoKey.enabled = true;
        haveKey = true;
        Destroy(Key);
        Door.GetComponent<SpriteRenderer>().sprite = openedDoor;
    }
}

0 references
private void OnCollisionExit2D(Collision2D collision)
{
    if (collision.gameObject.layer == LayerMask.NameToLayer("MoveingPlatform"))
    {
        this.transform.parent = null;
    }
}
```

Рисунок 3.17 – Частина коду методів OnCollisionEnter2D та OnCollisionExit2D

Досить важливою частиною гри є вороги головного персонажа. Реалізація логіки їх дій відбувається за допомогою методів Chill, Angry, GoBack та Update. У методі Chill реалізовується патрулювання ворога у заданому радіусі. Метод Angry реалізовує переслідування головного персонажу та нанесення йому ударів. GoBack описує повертання ворогу до точки патрулювання. У Update виконується автоматичний вибір необхідного методу у певний момент ігрового часу. Частину коду, що описує логіку дій ворога зображено на рисунку 3.18 та 3.19.

```
private void Chill()
{
    if (transform.position.x > point.position.x + positionOfPatrol)
    {
        moveingRight = false;
    }
    else if(transform.position.x < point.position.x - positionOfPatrol)
    {
        moveingRight = true;

    }

    if (moveingRight)
    {
        transform.position = new Vector2(transform.position.x + speed * Time.deltaTime, transform.position.y);
    }
    else
    {
       transform.position = new Vector2(transform.position.x - speed * Time.deltaTime, transform.position.y);
    }
}
private void Angry()
{
    transform.position = Vector2.MoveTowards(transform.position, player.position, speed * Time.deltaTime);


    Collider2D[] hitEnemies = Physics2D.OverlapCircleAll(attackPoint.position, attackRange, enemyLayers);

    if (hitEnemies.Length > 0 && Time.time >= nextAttackTime) {
        animator.SetTrigger("Attack");
    foreach(Collider2D enemy in hitEnemies)
    {
        Debug.Log("we hit " + enemy.name);
        enemy.GetComponent<Player>().GetDamageFromBandit();
    }
    nextAttackTime = Time.time + 1f / attackRate;

    }
}

1 reference
private void GoBack()
{
    transform.position = Vector2.MoveTowards(transform.position, point.position, speed * Time.deltaTime);
}
```

Рисунок 3.18 - Частина коду методів Chill, Angry та Angry

```
if (Vector2.Distance(transform.position, point.position) < positionOfPatrol && angry == false)
{
    chill = true;
}

if (Vector2.Distance(transform.position, player.position) < stoppingDistance)
{
    angry = true;
    chill = false;
    goBack = false;
}

if (Vector2.Distance(transform.position, player.position) > stoppingDistance)
{
    goBack = true;
    angry = false;
}

if (chill == true)
{
    Chill();
}
else if (angry == true)
{
    Angry();
}
else if (goBack == true)
{
    GoBack();
}
```

Рисунок 3.19 – Частина коду методу Update

Підказки у грі реалізовані за допомогою трьох класів: Dialogue, DialogueManager та DialogueAnimator. Підказки реалізовані таким чином, що коли гравець стане у певну область гри перед ним з’явиться діалогове вікно зпідказкою. Частина коду класу DialogueManager, Dialogue та DialogueAnimator зображена на рисунку 3.20, 3.21, 3.22 та 3.23.

```
public Text dialogueText;
1 reference
public Text nameText;
1 reference
public Animator dialogAnim;
5 references
private Queue<string> sentences;
0 references
private void Start()
{
    sentences = new Queue<string>();
}
```

Рисунок 3.20 - Частина коду класу DialogueManager

```
public void StartDialogue(Dialogue dialogue)
{

    nameText.text = dialogue.name;
    sentences.Clear();

    foreach(string sentence in dialogue.sentences)
    {
        sentences.Enqueue(sentence);
    }
    DisplayNextSentence();
}

1 reference
public void DisplayNextSentence()
{
    if (sentences.Count == 0)
    {
        EndDialogue();
        return;
    }
    string sentence = sentences.Dequeue();
    StopAllCoroutines();
    StartCoroutine(TypeSentence(sentence));

}

1 reference
IEnumerator TypeSentence(string sentences)
{
    dialogueText.text = "";
    foreach(char letter in sentences.ToCharArray())
    {
        dialogueText.text += letter;
        yield return null;
    }
}
public void EndDialogue()
{
    dialogAnim.SetBool("dialogOpen", false);
}
```

Рисунок 3.21 - Частина коду класу DialogueManager

```
[System.Serializable]
2 references
public class Dialogue
{
    1 reference
    public string name;
    [TextArea(3, 10)]
    1 reference
    public string[] sentences;
}
```

Рисунок 3.22 – Частина коду класу Dialogue

```
public class DialogueAnimator : MonoBehaviour
{
    1 reference
    public Animator dialogAnim;
    1 reference
    public DialogueManager dm;
    1 reference
    public Dialogue dialogue;

    0 references
    public void OnTriggerEnter2D(Collider2D other)
    {
        if (other.CompareTag("Player")) {
            dialogAnim.SetBool("dialogOpen", true);
            FindObjectOfType<DialogueManager>().StartDialogue(dialogue);
        }
    }

        0 references
        public void OnTriggerExit2D(Collider2D other)
    {
            dm.EndDialogue();
    }
}
```

Рисунок 3.23 – Частина коду класу DialogueAnimator

Клас CameraController дозволяє камері стежити за головним персонажем у межах ігрового середовища. Частина коду класу CameraController наведена на рисунку 3.24.

```
public class CameraController : MonoBehaviour
{
    3 references
    [SerializeField] private Transform player;
    3 references
    private Vector3 pos;

    0 references
    void Start()
    {
        if (!player)
            player = FindObjectOfType<Player>().transform;
    }

        0 references
        void Update()
    {
        pos = player.position;
        pos.z = -10f;

        transform.position = Vector3.Lerp(transform.position, pos, Time.deltaTime);
    }
}
```

Рисунок 3.24 - Частина коду класу CameraController

3.4 Висновки

У третьому розділі було спроектовано інтерфейс користувача ігрового додатку, розроблена схема переходів між сценами.

Розроблено графічний матеріал для наповнення контенту ігрового додатку. Для розробки графічних матеріалів було обрано графічний редактор Adobe Photoshop та пошук медіа матеріалів в інтернет середовищі.

Виконано розробку ігрової системи, описано розробку основних її модулів.

4 ТЕСТУВАННЯ ДОДАТКУ ТА РОЗРОБКА РЕКОМЕНДАЦІЙ КОРИСТУВАЧУ

4.1 Вибір методів тестування програмного забезпечення

Тестування програмного забезпечення – це процес оцінки та перевірки того, що програмний продукт або додаток виконує все те, що він повинен робити. Переваги тестування включають запобігання помилкам, зниження витрат на розробку та підвищення продуктивності[16].

Тестування «білої скриньки» – це метод тестування програмного забезпечення, при якому тестувальнику відома внутрішня структура, дизайн, реалізація предмета, що тестується[17].

Методи тестування білого ящика аналізують внутрішні структури, використані структури даних, внутрішній дизайн, структуру коду та роботу програмного забезпечення, а не лише функціональність, як у тестуванні чорного ящика. Його також називають тестуванням скляної коробки або тестуванням прозорої коробки або структурним тестуванням.

Робочий процес тестування білої коробки:

- Вхідні дані: вимоги, функціональні специфікації, конструкторська документація, вихідний код.
- Обробка: Виконання аналізу ризику для керівництва через весь процес.
- Правильне планування тестування: розробка тестових випадків так, щоб охопити весь код. Виконайте повторне проходження, поки не буде досягнуто програмне забезпечення без помилок. Також повідомляється про результати.
- Вихід: Підготовка остаточного звіту про весь процес тестування.

Методики тестування білої скриньки:

- Покриття операторів: у цій техніці мета полягає в тому, щоб обійти всі оператори принаймні один раз. Отже, кожен рядок коду

перевіряється. У випадку блок-схеми кожен вузол потрібно пройти принаймні один раз. Оскільки всі рядки коду охоплені це допомагає вказати несправний код.

- Покриття гілок: у цій техніці тестові випадки розроблені таким чином, що кожна гілка з усіх точок рішення проходить принаймні один раз. У блок-схемі всі ребра потрібно пройти принаймні один раз.

- Тестування основного шляху: у цій техніці графи потоків контролю створюються з коду або блок-схеми, а потім обчислюється цикломатична складність, яка визначає кількість незалежних шляхів, щоб мінімальна кількість тестових випадків могла бути розроблена для кожного незалежного шляху.

Тестування «чорної скриньки» - це метод тестування програмного забезпечення, який аналізує функціональність програмного забезпечення чи додатка, не знаючи особливостей про внутрішню структуру, дизайн елемента, який тестується, і порівнює вхідне значення з вихідним значенням[18].

Основна увага тестуванню чорної скриньки приділяється функціональності системи в цілому. Термін «тестування поведінки» також використовується для тестування чорної скриньки.

Тестування чорної скриньки відбувається протягом усього життєвого циклу розробки та тестування програмного забезпечення, тобто на етапах тестування одиниць, інтеграції, приймання та регресійного тестування.

Типи тестування чорної скриньки:

- Функціональне тестування. Цей тип тестування стосується функціональних вимог або специфікацій програми. Тут тестуються різні дії або функції системи шляхом надання вхідних даних і порівняння фактичного результату з очікуваним.

- Нефункціональне тестування. Окрім функціональних можливостей вимог, є навіть кілька нефункціональних аспектів, які потрібно перевірити, щоб покращити якість та продуктивність програми.

Методи тестування чорної скриньки:

- Розбиття еквівалентності
- Аналіз граничних значень
- Тестування таблиці рішень
- Тестування переходу стану
- Помилка вгадування
- Методи тестування на основі графіків
- Порівняльне тестування

Проаналізовано переваги та недоліки методів тестування «чорної скриньки» та «білої скриньки». Обрано метод «чорної скриньки» для тестування програмного забезпечення з урахуванням мети програми.

4.2 Тестування розробленого додатку

Першим кроком тестування мобільної гри є її запуск. Після запуску гри очікується побачити головне меню з пунктами: «Нова гра», «Рівні» та «Вихід». Результат запуску зображено на рисунку 4.1.

Рисунок 4.1 – Головне меню гри

Після успішного запуску гри, розпочинається тестування її функцій. Для даної мобільної гри першим кроком у тестуванні є запуск першого рівня. Для того, щоб перейти до першого рівня потрібно натиснути клавішу «Нова гра», результатом чого буде завантаження першого рівня гри. Результат натискання кнопки «Нова гра» зображено на рисунку 4.2.

Рисунок 4.2 - Результат натискання кнопки «Нова гра»

Подальшим кроком буде тестування руху головного персонажа. Для того щоб перемістити персонажа потрібно натиснути екранні кнопки «Вліво», «Вправо» та «Стрибок». Результат натискання кнопок руху головного персонажу зображено на рисунку 4.3.

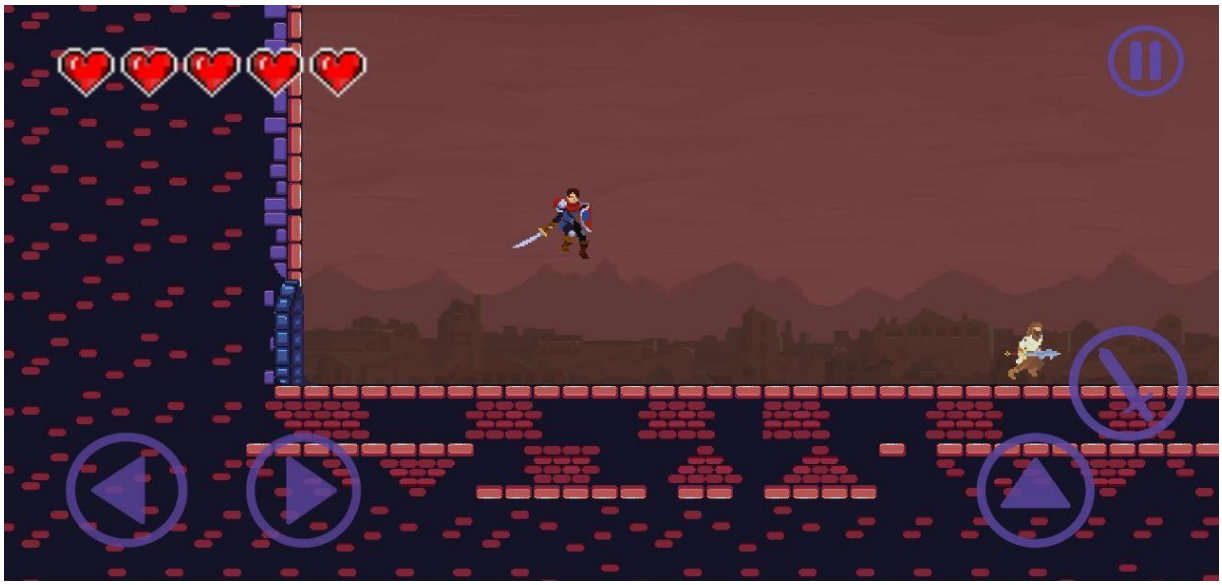

Рисунок 4.3 - Результат натискання кнопок руху головного персонажу

Після успішної перевірки руху персонажу необхідно перевірити його атаку. Атака персонажа здійснюється за допомогою екранної кнопки «Атака». Результат натискання кнопки «Атака» зображено на рисунку 4.4.

Рисунок 4.4 - Результат натискання кнопки «Атака»

Наступним кроком буде перевірка роботи підказок. Якщо підійти у певну область ігрового світу, на екрані повинна з’явитися підказка. Поява підказки наведена на рисунку 4.5.

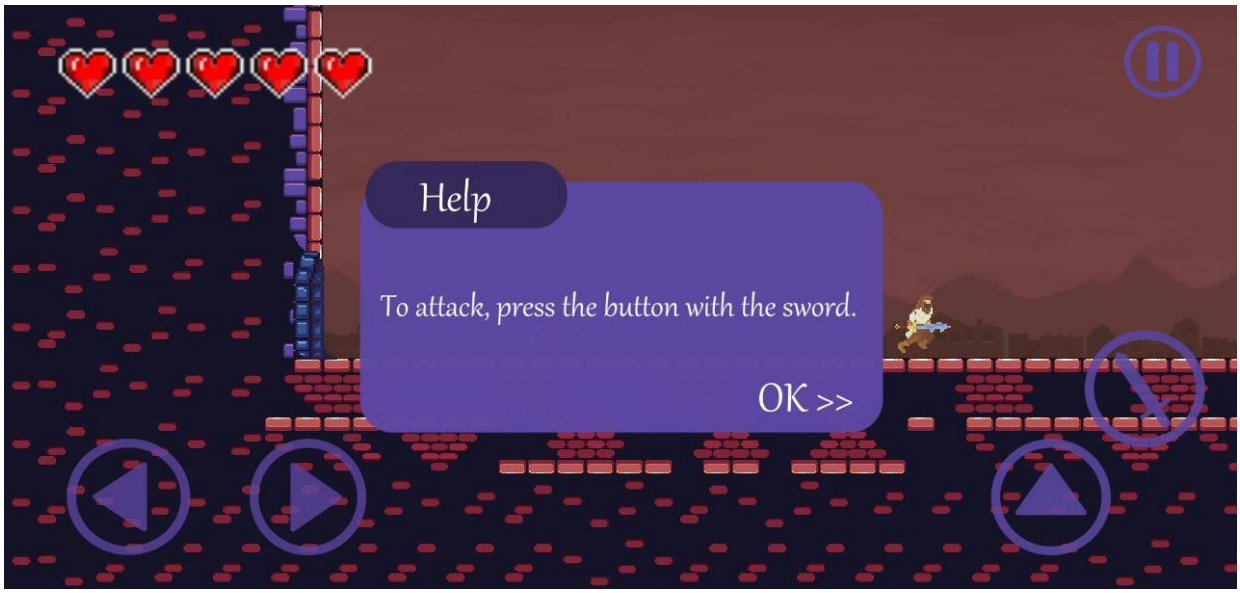

Рисунок 4.5 – Поява підказки

Далі слід перевірити логіку роботи ворога та процес отримання шкоди головного персонажу. Для цього слід підійти до ворога та дати йому змогу вдарити головного персонажа. Результат атаки ворога зображено на рисунку 4.6.

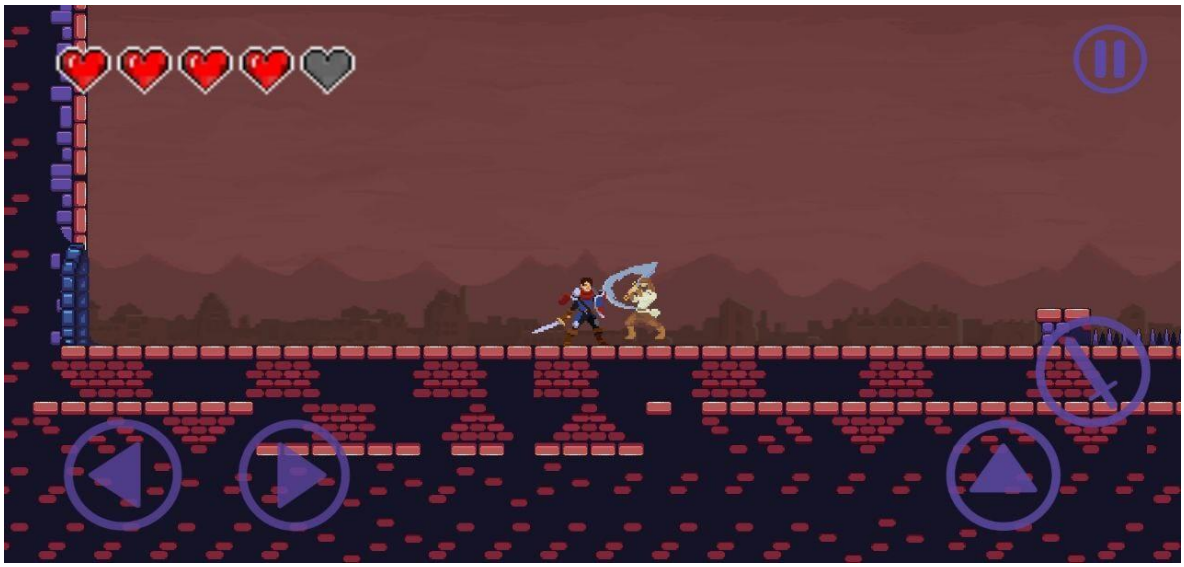

Рисунок 4.6 – Результат атаки ворога

Перевіримо смерть персонажу. Для цього втратимо всі життя, після чого повинно з’явитися меню смерті персонажу. Меню смерті персонажу зображено на рисунку 4.7.

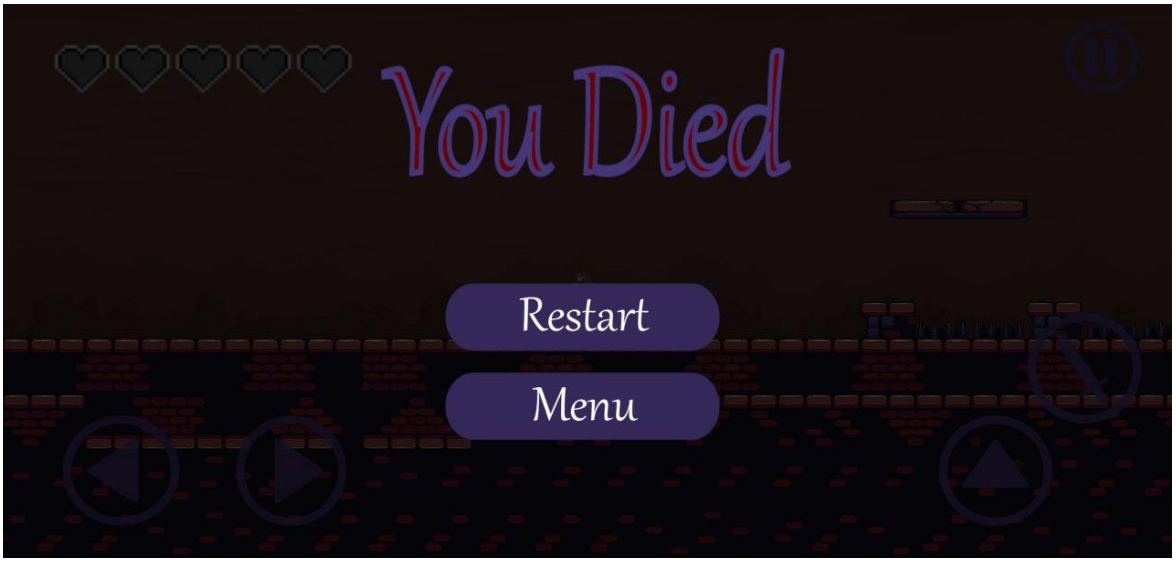

Рисунок 4.7 – Меню смерті персонажу

Також слід перевірити роботу меню паузи. Для цього потрібно натиснути кнопку «Пауза». Результат натискання кнопки «Пауза» зображено на рисунку 4.8.

Рисунок 4.8 - Результат натискання кнопки «Пауза»

Наступним кроком тестування необхідно перевірити знищення ворога. Для знищення ворога необхідно до нього підійти да вдарити його декілька раз мечем. Після чого у ворога відбудеться анімація смерті та він стане не активним. Результат знищення ворога зображено на рисунку 4.9.

Рисунок 4.9 - Результат знищення ворога

При зіткненні з пасткою персонаж повинен отримати шкоду та стрибнути. Результат зіткнення персонажа з пасткою зображено на рисунку 4.10.

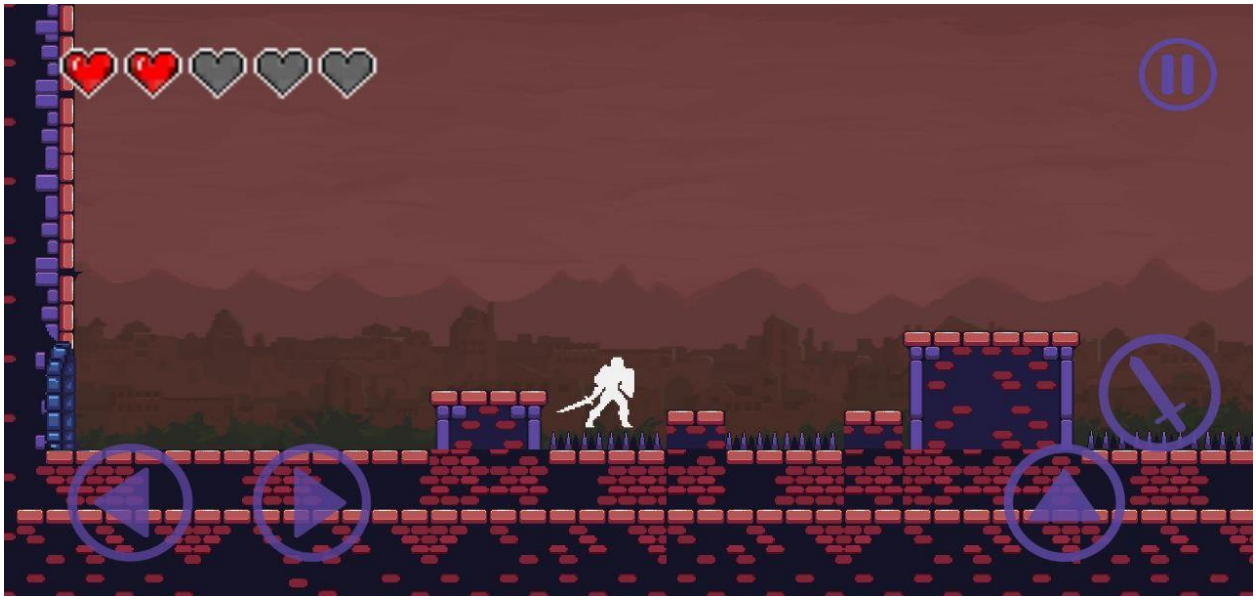

Рисунок 4.10 - Результат зіткнення персонажа з пасткою

Для того щоб завершити рівень необхідно знайти ключ та відчинити двері. Якщо підійти до дверей без ключа, то користувач отримає повідомлення про те що потрібно знайти ключ (рис. 4.11).

Рисунок 4.11 – Повідомлення про необхідність ключа

Після того як буде знайдено ключ, він відобразиться поблизу життів у верхньому лівому куту та двері виходу з рівня будуть відчинені (рис. 4.12).

Рисунок 4.12 – Результат підбирання ключа

Зайшовши у відчинені двері буде відображена сцена закінчення рівня. На сцені повинен бути надпис про завершення рівня та час за який рівень був пройдений. Сцена завершення рівня зображена на рисунку 4.13.

Рисунок 4.13 – Сцена завершення рівня

Після натискання клавіши «Продовжити» завантажиться головне меню, де кнопка «Нова гра» заміниться кнопкою «Продовжити», що завантажує максимально доступний рівень. Головне меню після проходження першого рівня зображено на рисунку 4.14.

Рисунок 4.14 - Головне меню після проходження першого рівня

Для того щоб переглянути усі доступні рівні необхідно натиснути кнопку «Рівні». Після чого відобразяться усі рівні, відкриті будуть активними, а закриті не активними. Сцена списку рівнів зображена на рисунку 4.15.

Рисунок 4.15 - Сцена списку рівнів

Для завершення тесту необхідно натиснути кнопку «Вихід» у головному меню. За результатами тестування мобільного додатку підтверджено працездатність усіх заявлених функцій.

4.3 Висновки

У четвертому розділі бакалаврської дипломної роботи було проаналізовано методики тестування та за результатами аналізу обрано метод тестування «чорного ящика». Відповідно до обраної методики, були проведені тести до розроблюваної мобільної гри та підтверджено повну працездатність.

ВИСНОВКИ

У бакалаврській дипломній роботі було розроблено мобільну 2d-гру «Dark Medieval» у жанрі екшн платформеру. Для розробки було використано середовище програмної розробки Microsoft Visual Studio Code та ігровий рушій Unity.

У бакалаврській дипломній роботі проаналізовано стан проблем на сьогоднішній день. Розглянуто основні аналоги програмних продуктів та у порівнянні з власними програмним продуктом було виявлено їхні недоліки. На основі порівняння визначено основні завдання дипломної роботи для розробки мобільної 2d-гри у жанрі екшн платформеру.

Під час аналізу технологій розробки було обґрунтовано вибір мови програмування C#. Також було розглянуто переваги використання ігрового рушія Unity.

Запропоновано метод реалізації ігрової системи, який, на відміну від існуючих, пропонує користувачу зміну поведінки ворогів залежно від поточної ігрової стратегії користувача з забезпеченням різноманітності ігрового процесу, що адаптує складність гри до вміння конкретного гравця і робить проходження кожного рівня не схожим до попереднього.

Розроблено модель ігрового середовища, яка, на відміну від існуючих, зосереджена на компонентах, розроблених за допомогою логічного мислення, і забезпечує розширену складність гри за допомогою системи підказок, що дозволяє налаштувати ігрове середовище під конкретних користувачів і реалізує комфортне проходження гри.

Спроектовано користувацький інтерфейс ігрового додатку, розроблена схема переходів між ігровими сценами. Виконано розробку програмного коду ігрової системи «Dark Medieval», призначеної для розважального характеру, яка допоможе користувачу тренувати логіку та реакцію при

проходженні складних рівнів. Розроблено графічний матеріал для наповнення ігрового додатку.

Виконано аналіз методів тестування, обрано метод тестування «чорної скриньки». За обраним методом проведено тестування розробленого мобільного ігрового додатку згідно поставлених задач, яке підтвердило його повну працездатність.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Інформаційні технології у суспільстві [Електронний ресурс] – Режим доступу до ресурсу: https://sites.google.com/site/informacijne-suspilstvo26/informacijni-tehnologiie-u-suspilstvi.

2. Mobile Game Development Process: How Mobile Games are Created [Електронний ресурс] – Режим доступу до ресурсу: https://whimsygames.co/blog/game-development-process/.

3. Войтко В.В. Особливості розробки мобільної 2D екшн гри «DARK MEDIEVAL» / В.В. Войтко, Г.О. Черноволик, О.В. Гаврилюк, Н.Є. Барчук, В.С. Муковоз Матеріали Всеукраїнської науково-практичної інтернет-конференції "Молодь в науці: дослідження, проблеми, перспективи -2022", Секція - Інформаційні технології та комп‘ютерна інженерія. [Електронний ресурс] – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/mn/mn2022/paper/view/162 25/13653

4. The Many Different Types of Video Games & Their Subgenres [Електронний ресурс] – Режим доступу до ресурсу: https://www.idtech.com/blog/different-types-of-video-game-genres.

5. Apple Knight: Action Platformer [Електронний ресурс] – Режим доступу до ресурсу: https://play.google.com/store/apps /details?id=online.limitless.appleknight.free&hl=ru&gl=US.

6. Sword Of Xolan [Електронний ресурс] – Режим доступу до ресурсу: https://play.google.com/store/apps/details?id=com.Alper.SwordOfXolan&hl=ru&gl=US.

7. NinjAwesome [Електронний ресурс] – Режим доступу до ресурсу: https://play.google.com/store/apps/details?id=com.gameresort.ninjawesome&hl=ru&gl=US.

8. Oddmar [Електронний ресурс] – Режим доступу до ресурсу: https://play.google.com/store/apps/details?id=com.mobge.Oddmar&hl=ru&gl=US.

9. Программирование: принципы и практика с использованием C++. / Бьярн Страуструп – Диалектика, 2018. – 320 ст.

10. Andrew Troelsen Pro C# 8 with .NET Core 3 : навч. посіб. / Andrew Troelsen, Phil Japikse – Apress, 2020. – 1881 ст.

11. What is Unity? Everything you need to know [Електронний ресурс] – Режим доступу до ресурсу: https://www.androidauthority.com/what-is-unity-1131558/.

12. Video Games Lead to Faster Decisions that are No Less Accurate [Електронний ресурс] – Режим доступу до ресурсу: https://rochester.edu/news/show.php?id=3679.

13. Designing for the Digital Age: How to Create Human-Centered Products and Services. / Kim Goodwin – Wiley, 2009. – 768 ст.

14. How to Create Graphics for a Video Game [Електронний ресурс] – Режим доступу до ресурсу: https://discover.therookies.co/2019/03/29/how-to-create-graphics-for-a-video-game/.

15. Графічний редактор [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Графічний_редактор.

16. Искусство тестирования программ / Гленфорд Майерс, Диалектика, 2012. – 270 ст.

17. Hands-On Mobile App Testing: A Guide for Mobile Testers and Anyone Involved in the Mobile App Business 1st Edition / Daniel Knott, Addison-Wesley Professional, 2015. – 256 ст.

18. What is BLACK Box Testing? Techniques, Example & Types [Електронний ресурс] – Режим доступу до ресурсу: https://www.guru99.com/black-box-testing.html.

ДОДАТКИ

Додаток А – Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
д.т.н., проф. О. Н. Романюк
25 березня 2022 р.

Технічне завдання
на бакалаврську дипломну роботу «Розробка мобільної 2d-гри «Dark Medieval»»
за спеціальністю
121 – Інженерія програмного забезпечення

Керівник бакалаврської дипломної роботи:
_____ к.т.н., доц. Д.І. Катєльніков
"____"_____2022 р.
Виконав:
_____студент гр. 1ПІ-19мс2 В.С. Муковоз
"____"_____2022 р.

Вінниця – 2022 року

1. Найменування та галузь застосування

Бакалаврська дипломна робота: «Розробка мобільної 2d-гри «Dark Medieval»».

Галузь застосування – мобільні додатки.

2. Підстава для розробки.

Завдання на роботу, яке затверджене на засіданні кафедри програмного забезпечення – протокол № 13 від « 07 » лютого 2022 р.

3. Мета та призначення розробки.

Метою бакалаврської дипломної роботи є підвищення тренувальних можливостей логіки прийняття рішень та реакції користувача за допомогою розробки і використання 2d-гри у жанрі екшн платформера, що дозволяє проводити тренування в ігровій формі.

Призначення роботи – мобільна 2d-гра для проведення тренувань з логіки і реакції в ігровій формі.

4. Вихідні дані для проведення НДР

1. Andrew Troelsen Pro C# 8 with .NET Core 3 : навч. посіб. / Andrew Troelsen, Phil Japikse – Apress, 2020. – 1881 ст.

2. Designing for the Digital Age: How to Create Human-Centered Products and Services. / Kim Goodwin – Wiley, 2009. – 768 ст.

3. Hands-On Mobile App Testing: A Guide for Mobile Testers and Anyone Involved in the Mobile App Business 1st Edition / Daniel Knott, Addison-Wesley Professional, 2015. – 256 ст.

5. Технічні вимоги

Склад комплексу *технічних засобів*

Для розв’язку задачі буде використовуватися сучасний смартфон на операційній системі Android.

Вимоги до складових *частин комплексу технічних засобів.*

Для запуску програми апаратне забезпечення повинне відповідати мінімальним вимогам:

- смартфон з процесором Spreadtrum SC7731E;
- 1 Гб оперативної пам’яті;
- відеоядро ARM Mali-T820;
- сенсорний дисплей;
- ОС Android;

6. Конструктивні вимоги.

Конструкція пристрою повинна відповідати естетичним та ергономічним вимогам, повинна бути зручною в обслуговуванні та керуванні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред’являється по закінченню робіт:

– пояснювальна записка до бакалаврської дипломної роботи;

– технічне завдання;

– лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

Додаток Б – Протокол перевірки на плагіат

ПРОТОКОЛ
ПЕРЕВІРКИ БАКАЛАВРСЬКОЇ ДИПЛОМНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Розробка мобільної 2d-гри «Dark Medieval»

Тип роботи: БДР

Підрозділ : кафедра програмного забезпечення, ФІТКІ

Науковий керівник: к.т.н., доц. каф. ПЗ Катєльніков Д.І.

Оригінальність	93,2%
Схожість	6,8%

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Черноволик Г. О.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи _____ Муковоз Віктор Сергійович

Керівник роботи _____ Катєльніков Денис Іванович

Додаток В – Лістинг програми

Player.cs

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

public class Player : MonoBehaviour
{
    private float speed;
    [SerializeField] private float normalSpeed;
    [SerializeField] private int health = 5;
    [SerializeField] private float jumpForce = 15f;
    [SerializeField] private float attackRange = 0.5f;
    [SerializeField] private Transform attackPoint;
    [SerializeField] private LayerMask enemyLayers;
    [SerializeField] private float attackRate = 2f;
    private float nextAttackTime = 0f;
    private bool inAir;
    private int typeAttack = 1;
    private bool alive = true;
    private int lives;
    public bool haveKey = false;


    // Sounds


    //Game Objects
    [SerializeField] private Image[] hearts;
    [SerializeField] private Sprite aliveHeart;
    [SerializeField] private Sprite deadHeart;
    [SerializeField] private Sprite openedDoor;
    [SerializeField] private Sprite closedDoor;
    [SerializeField] private Image logoKey;
    [SerializeField] GameObject died;
    [SerializeField] GameObject Key;
    [SerializeField] GameObject Door;

    private Rigidbody2D rb;
    private SpriteRenderer sprite;
    private Animator animator;
    private Transform spriteTransform;
```

```
    public static Player Instance {get; set;}

    void Start()
    {

        logoKey.enabled = false;
        speed = 0f;
        lives = 5;
        health = lives;
        rb = GetComponent<Rigidbody2D>();
        sprite = GetComponentInChildren<SpriteRenderer>();
        spriteTransform = GetComponentInChildren<Transform>();
        animator = GetComponentInChildren<Animator>();
        Instance = this;
        alive = true;
        died.SetActive(false);
    }


    void Update()
    {
        if (speed != 0f) {
         Vector3 dir = transform.right;
         transform.position = Vector3.MoveTowards(transform.position, transform.position + dir,
speed * Time.deltaTime);
        }

            else if (alive) {
            animator.SetInteger("AnimState", 0);
        }

        if (health > lives)
            health = lives;

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

            if (i < lives)
```

```
            hearts[i].enabled = true;
        else
            hearts[i].enabled = false;
    }

    animator.SetFloat("AirSpeedY", rb.velocity.y);
}


public void onLeftButtonDown()
{
    if (speed >= 0f)
    {
        speed = -normalSpeed;
        sprite.flipX = true;
        attackPoint.position = new Vector2(spriteTransform.position.x - 1.2f,
spriteTransform.position.y - 0.3f);
    }

        if (!inAir)
    {
        animator.SetInteger("AnimState", 1);
    }
}

public void onRightButtonDown()
{
    if (speed <= 0f)
    {
         speed = normalSpeed;
         sprite.flipX = false;
         attackPoint.position = new Vector2(spriteTransform.position.x - 0.2f,
spriteTransform.position.y - 0.3f);
    }

        if (!inAir)
    {
        animator.SetInteger("AnimState", 1);
    }
}

public void onJumpButtonDown()
{
```

```
    if (!inAir)
    {
        Jump();
    }
}

private void Jump()
{
    rb.AddForce(transform.up * jumpForce, ForceMode2D.Impulse);
    inAir = true;
    animator.SetTrigger("Jump");

}

public void onButtonUp()
{
    speed = 0f;
}

public void onAttackButtonDown()
{
    if (Time.time >= nextAttackTime){
            Attack();
            nextAttackTime = Time.time + 1f / attackRate;
        }
}

    void Attack()
{
    // Анімація удару
    if (typeAttack == 1)
    {
        animator.SetTrigger("Attack1");
        typeAttack++;
    }
    else if (typeAttack == 2)
    {
        animator.SetTrigger("Attack2");
        typeAttack++;
    }
    else if (typeAttack == 3)
    {
        animator.SetTrigger("Attack3");
```

```
            typeAttack = 1;
        }
        // Розпізнавання ворогів у радіусі атаки
        Collider2D[] hitEnemies = Physics2D.OverlapCircleAll(attackPoint.position, attackRange,
enemyLayers);

        // Атака ворога
        foreach(Collider2D enemy in hitEnemies)
        {
            Debug.Log("we hit " + enemy.name);
            enemy.GetComponent<Bandit>().TakeDamage();
        }
    }

    private void OnDrawGizmosSelected()
    {
        if (attackPoint == null)
            return;
        Gizmos.color = Color.red;
        Gizmos.DrawWireSphere(attackPoint.position, attackRange);
    }


    private void Die()
    {
        alive = false;
        animator.SetTrigger("Death");
        this.enabled = false;
        died.SetActive(true);
    }
    private void GetDamage()
    {
        health --;
        Debug.Log(health);
            if (health < 1) {
                foreach (var h in hearts)
                {
                    h.sprite = deadHeart;
                }
                Die();
            } else animator.SetTrigger("Hurt");

    }
```

```
public void GetDamageFromPrickle()
{
   GetDamage();
   if (health > 0)
      Jump();
}

public void GetDamageFromBandit()
{
   GetDamage();
}

   public bool isAlive()
{
   return alive;
}

private void OnCollisionEnter2D(Collision2D collision)
{
   if (collision.gameObject.layer == LayerMask.NameToLayer("Ground")
   || collision.gameObject.layer == LayerMask.NameToLayer("MoveingPlatform"))
   {
      inAir = false;
      animator.SetBool("Grounded", true);
   }

   if (collision.gameObject.layer == LayerMask.NameToLayer("MoveingPlatform"))
   {
      this.transform.parent = collision.transform;
   }

   if (collision.gameObject.layer == LayerMask.NameToLayer("Key"))
   {
      logoKey.enabled = true;
      haveKey = true;
      Destroy(Key);
      Door.GetComponent<SpriteRenderer>().sprite = openedDoor;
   }
}

private void OnCollisionExit2D(Collision2D collision)
{
   if (collision.gameObject.layer == LayerMask.NameToLayer("MoveingPlatform"))
```

```
        {
            this.transform.parent = null;
        }
    }


}
```

Bandit.cs

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

public class Player : MonoBehaviour
{


    private float speed;
    [SerializeField] private float normalSpeed;
    [SerializeField] private int health = 5;
    [SerializeField] private float jumpForce = 15f;
    [SerializeField] private float attackRange = 0.5f;
    [SerializeField] private Transform attackPoint;
    [SerializeField] private LayerMask enemyLayers;
    [SerializeField] private float attackRate = 2f;
    private float nextAttackTime = 0f;
    private bool inAir;
    private int typeAttack = 1;
    private bool alive = true;
    private int lives;
    public bool haveKey = false;


    // Sounds


    //Game Objects
    [SerializeField] private Image[] hearts;
    [SerializeField] private Sprite aliveHeart;
    [SerializeField] private Sprite deadHeart;
    [SerializeField] private Sprite openedDoor;
```

```
    [SerializeField] private Sprite closedDoor;
    [SerializeField] private Image logoKey;
    [SerializeField] GameObject died;
    [SerializeField] GameObject Key;
    [SerializeField] GameObject Door;

    private Rigidbody2D rb;
    private SpriteRenderer sprite;
    private Animator animator;
    private Transform spriteTransform;



    public static Player Instance {get; set;}

    void Start()
    {

        logoKey.enabled = false;
        speed = 0f;
        lives = 5;
        health = lives;
        rb = GetComponent<Rigidbody2D>();
        sprite = GetComponentInChildren<SpriteRenderer>();
        spriteTransform = GetComponentInChildren<Transform>();
        animator = GetComponentInChildren<Animator>();
        Instance = this;
        alive = true;
        died.SetActive(false);
    }


    void Update()
    {
        if (speed != 0f) {
         Vector3 dir = transform.right;
         transform.position = Vector3.MoveTowards(transform.position, transform.position + dir,
speed * Time.deltaTime);
        }

            else if (alive) {
            animator.SetInteger("AnimState", 0);
        }
```

```
        if (health > lives)
            health = lives;

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

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

        animator.SetFloat("AirSpeedY", rb.velocity.y);
    }


    public void onLeftButtonDown()
    {
        if (speed >= 0f)
        {
            speed = -normalSpeed;
            sprite.flipX = true;
            attackPoint.position = new Vector2(spriteTransform.position.x - 1.2f,
spriteTransform.position.y - 0.3f);
        }

            if (!inAir)
        {
            animator.SetInteger("AnimState", 1);
        }
    }

    public void onRightButtonDown()
    {
        if (speed <= 0f)
        {
             speed = normalSpeed;
             sprite.flipX = false;
```

```
            attackPoint.position = new Vector2(spriteTransform.position.x - 0.2f,
spriteTransform.position.y - 0.3f);
        }

           if (!inAir)
        {
           animator.SetInteger("AnimState", 1);
        }
    }

    public void onJumpButtonDown()
    {
        if (!inAir)
        {
           Jump();
        }
    }

    private void Jump()
    {
        rb.AddForce(transform.up * jumpForce, ForceMode2D.Impulse);
        inAir = true;
        animator.SetTrigger("Jump");

    }

    public void onButtonUp()
    {
        speed = 0f;
    }

    public void onAttackButtonDown()
    {
        if (Time.time >= nextAttackTime){
              Attack();
              nextAttackTime = Time.time + 1f / attackRate;
           }
    }

        void Attack()
    {
        // Анімація удару
        if (typeAttack == 1)
```

```
        {
            animator.SetTrigger("Attack1");
            typeAttack++;
        }
        else if (typeAttack == 2)
        {
            animator.SetTrigger("Attack2");
            typeAttack++;
        }
        else if (typeAttack == 3)
        {
            animator.SetTrigger("Attack3");
            typeAttack = 1;
        }
        // Розпізнавання ворогів у радіусі атаки
        Collider2D[] hitEnemies = Physics2D.OverlapCircleAll(attackPoint.position, attackRange,
enemyLayers);

        // Атака ворога
        foreach(Collider2D enemy in hitEnemies)
        {
            Debug.Log("we hit " + enemy.name);
            enemy.GetComponent<Bandit>().TakeDamage();
        }
    }

    private void OnDrawGizmosSelected()
    {
        if (attackPoint == null)
            return;
        Gizmos.color = Color.red;
        Gizmos.DrawWireSphere(attackPoint.position, attackRange);
    }


    private void Die()
    {
        alive = false;
        animator.SetTrigger("Death");
        this.enabled = false;
        died.SetActive(true);
    }
    private void GetDamage()
```

```
{
   health --;
   Debug.Log(health);
      if (health < 1) {
         foreach (var h in hearts)
         {
            h.sprite = deadHeart;
         }
         Die();
      } else animator.SetTrigger("Hurt");

}
public void GetDamageFromPrickle()
{
   GetDamage();
   if (health > 0)
      Jump();
}

public void GetDamageFromBandit()
{
   GetDamage();
}

   public bool isAlive()
{
   return alive;
}

private void OnCollisionEnter2D(Collision2D collision)
{
   if (collision.gameObject.layer == LayerMask.NameToLayer("Ground")
   || collision.gameObject.layer == LayerMask.NameToLayer("MoveingPlatform"))
   {
      inAir = false;
      animator.SetBool("Grounded", true);
   }

   if (collision.gameObject.layer == LayerMask.NameToLayer("MoveingPlatform"))
   {
      this.transform.parent = collision.transform;
   }
```

```
        if (collision.gameObject.layer == LayerMask.NameToLayer("Key"))
        {
            logoKey.enabled = true;
            haveKey = true;
            Destroy(Key);
            Door.GetComponent<SpriteRenderer>().sprite = openedDoor;
        }
    }

    private void OnCollisionExit2D(Collision2D collision)
    {
        if (collision.gameObject.layer == LayerMask.NameToLayer("MoveingPlatform"))
        {
            this.transform.parent = null;
        }
    }


}
```

CameraController.cs

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

public class CameraController : MonoBehaviour
{
    [SerializeField] private Transform player;
    private Vector3 pos;

    void Start()
    {
        if (!player)
            player = FindObjectOfType<Player>().transform;
    }

        void Update()
    {
        pos = player.position;
```

```
        pos.z = -10f;

        transform.position = Vector3.Lerp(transform.position, pos, Time.deltaTime);
    }
}
```

Died.cs

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

public class Died : MonoBehaviour
{
    [SerializeField]
    GameObject died;


    void Start()
    {
        died.SetActive(false);
    }


    void Update()
    {

    }

    public void Die()
    {
        died.SetActive(true);
    }
```

```
    public void Restart()
    {
        string currentSceneName = SceneManager.GetActiveScene().name;
        SceneManager.LoadScene(currentSceneName);
    }

    public void Menu()
    {
        SceneManager.LoadScene(0);
    }
}
```

GoToNextLevel.cs

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

public class GoToNextLevel : MonoBehaviour
{

    [SerializeField] GameObject levelCopmleted;

    public float timeStart = 0;
    public Text timerText;
    bool timerRunning = true;

    void Start()
    {
        levelCopmleted.SetActive(false);
    }
```

```
    void Update()
    {
        if (timerRunning){
        timeStart += Time.deltaTime;
        timerText.text = "Time: " + (Mathf.Round(timeStart / 60 < 1 ? 0 : timeStart /
60)).ToString() + ":" + (Mathf.Round(timeStart % 60)).ToString();
       }
    }
    public void LevelCompleted()
    {
        timerRunning = false;
        levelCopmleted.SetActive(true);
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if(collision.gameObject.layer == LayerMask.NameToLayer("Player") &&
Player.Instance.haveKey)
        {
            UnLockLevel();
            LevelCompleted();
        }
    }
    public void UnLockLevel()
    {
        int currentLevel = SceneManager.GetActiveScene().buildIndex;

        if(currentLevel >= PlayerPrefs.GetInt("levels"))
        {
            PlayerPrefs.SetInt("levels", currentLevel + 1);
        }
    }
}
```

LevelSelector.cs

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

public class LevelSelector : MonoBehaviour
{
    int levelUnLock;
    public Button[] buttons;

    void Start()
    {
        levelUnLock = PlayerPrefs.GetInt("levels", 1);

        for (int i = 0; i < buttons.Length; i++)
        {
            buttons[i].interactable = false;
        }

        for (int i = 0; i <levelUnLock-1; i++)
        {
            buttons[i].interactable = true;
        }
    }

    public void loadLevel(int levelIndex)
    {
        SceneManager.LoadScene(levelIndex);
    }

    public void backToMainMenu()
    {
```

```
        SceneManager.LoadScene(0);
    }
}
```

MainMenu.cs

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

public class MainMenu : MonoBehaviour
{
    public Text gameButtonText;
    void Start()
    {
        if (PlayerPrefs.GetInt("levels", 0) == 0)
        {
            gameButtonText.text = "New game";
        } else
        {
            gameButtonText.text = "Continue";
        }
    }

    public void OpenGame()
    {
        SceneManager.LoadScene(PlayerPrefs.GetInt("levels", 2));
    }

    public void OpenLevelsList()
    {
        SceneManager.LoadScene(1);
    }
```

```
    public void Exit()
    {
        Application.Quit();
    }

}
```

MoveingPlatform.cs

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

public class MoveingPlatform : MonoBehaviour
{
    float dirX;
    [SerializeField] private float speed = 3f;
    [SerializeField] private float distance = 4f;
    private float spawnPosition;

    bool moveingRight = true;

    void Start()
    {
        spawnPosition = transform.position.x;
    }

    void Update()
    {
        if (transform.position.x > spawnPosition + distance)
        {
            moveingRight = false;
        }
        else if (transform.position.x < spawnPosition - distance)
        {
            moveingRight = true;
        }

        if (moveingRight)
        {
```

```
            transform.position = new Vector2(transform.position.x + speed * Time.deltaTime,
transform.position.y);
        } else
        {
            transform.position = new Vector2(transform.position.x - speed * Time.deltaTime,
transform.position.y);
        }

    }

    private void OnDrawGizmosSelected()
    {
        if (transform == null)
            return;
        Gizmos.color = Color.red;
        Gizmos.DrawWireSphere(transform.position, distance);
    }
}
```

Paused.cs

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


public class Paused : MonoBehaviour
{

    [SerializeField]GameObject pause;

    void Start()
    {
        pause.SetActive(false);
    }


    void Update()
    {

    }
```

```
    public void PauseOn()
    {
        pause.SetActive(true);
        Time.timeScale = 0;
    }

    public void PauseOff()
    {
        pause.SetActive(false);
        Time.timeScale = 1;
    }

    public void Menu()
    {
        SceneManager.LoadScene(0);
        Time.timeScale = 1;
    }

    public void Restart()
    {
        string currentSceneName = SceneManager.GetActiveScene().name;
        SceneManager.LoadScene(currentSceneName);
        Time.timeScale = 1;
    }


}
```

Prickle.cs

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

public class Prickle : MonoBehaviour
{
    void Start()
    {

    }
```

```
    void Update()
    {

    }

    private void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.gameObject == Player.Instance.gameObject && Player.Instance.isAlive())
        {
            Player.Instance.GetDamageFromPrickle();
        }
    }
}
```

PushTrap.cs

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

public class PushTrap : MonoBehaviour
{
    [SerializeField] GameObject prickle;
    [SerializeField] GameObject pushButton;
    float coolDownTrap = 0;
    bool trapReady = true;

    void Start()
    {
        prickle.SetActive(false);
        pushButton.SetActive(true);
    }

    void Update()
    {
        if (!trapReady)
        {
            coolDownTrap += Time.deltaTime;
            prickle.SetActive(true);
            pushButton.SetActive(false);
```

```
        }
        else
        {
            prickle.SetActive(false);
            pushButton.SetActive(true);
        }

        if (coolDownTrap >= 0.5)
        {
            trapReady = true;
            coolDownTrap = 0;
        }

    }

    public void OnTriggerEnter2D(Collider2D other)
    {
        if (other.CompareTag("Player")) {
            trapReady = false;
        }
    }
}
```

Dialogue.cs

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

[System.Serializable]
public class Dialogue
{
    public string name;
    [TextArea(3, 10)]
    public string[] sentences;
}
```

DialogueAnimator.cs

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

```
using UnityEngine;

public class DialogueAnimator : MonoBehaviour
{
    public Animator dialogAnim;
    public DialogueManager dm;
    public Dialogue dialogue;

    public void OnTriggerEnter2D(Collider2D other)
    {
        if (other.CompareTag("Player")) {
            dialogAnim.SetBool("dialogOpen", true);
            FindObjectOfType<DialogueManager>().StartDialogue(dialogue);
        }
    }

        public void OnTriggerExit2D(Collider2D other)
    {
            dm.EndDialogue();
    }
}
```

DialogueManager.cs

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

public class DialogueManager : MonoBehaviour
{
    public Text dialogueText;
    public Text nameText;
    public Animator dialogAnim;
    private Queue<string> sentences;
```

```
private void Start()
{
    sentences = new Queue<string>();
}

public void StartDialogue(Dialogue dialogue)
{

    nameText.text = dialogue.name;
    sentences.Clear();

    foreach(string sentence in dialogue.sentences)
    {
        sentences.Enqueue(sentence);
    }
    DisplayNextSentence();
}

public void DisplayNextSentence()
{
    if (sentences.Count == 0)
    {
        EndDialogue();
        return;
    }
    string sentence = sentences.Dequeue();
    StopAllCoroutines();
    StartCoroutine(TypeSentence(sentence));

}

IEnumerator TypeSentence(string sentences)
{
    dialogueText.text = "";
    foreach(char letter in sentences.ToCharArray())
```

```
        {
            dialogueText.text += letter;
            yield return null;
        }
    }

    public void EndDialogue()
    {
        dialogAnim.SetBool("dialogOpen", false);
    }
}
```

Додаток Г – Графічна частина

ГРАФІЧНА ЧАСТИНА
РОЗРОБКА МОБІЛЬНОЇ 2D-ГРИ «DARK MEDIEVAL»

Рисунок Г.1 – Назва роботи

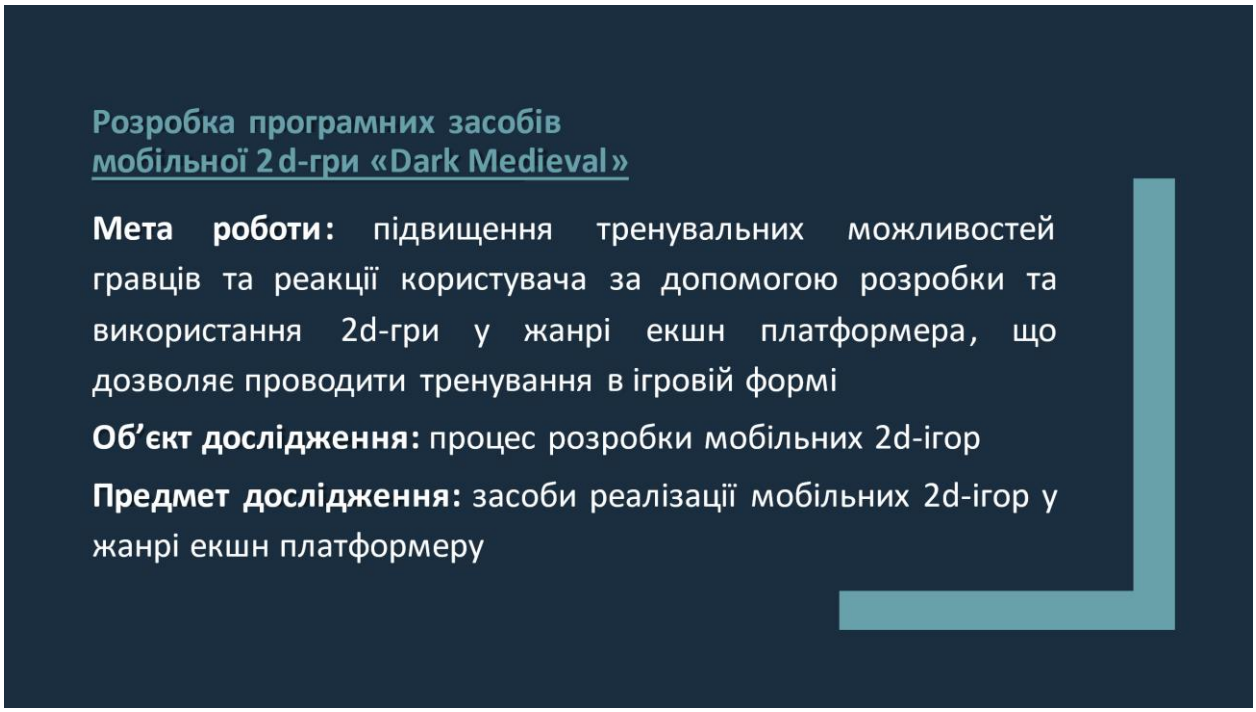

Рисунок Г.2 – Мета, об'єкт і предмет дослідження

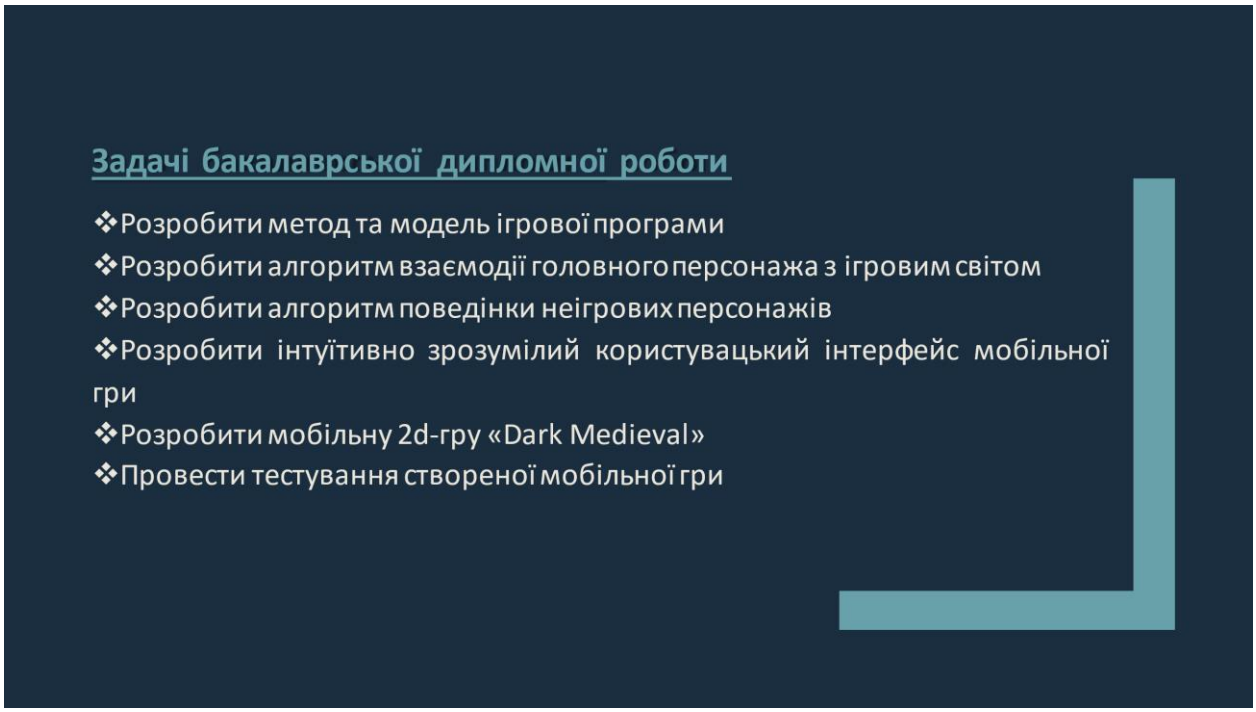

Рисунок Г.3 – Задачі бакалаврської дипломної роботи

Рисунок Г.4 – Наукова новизна

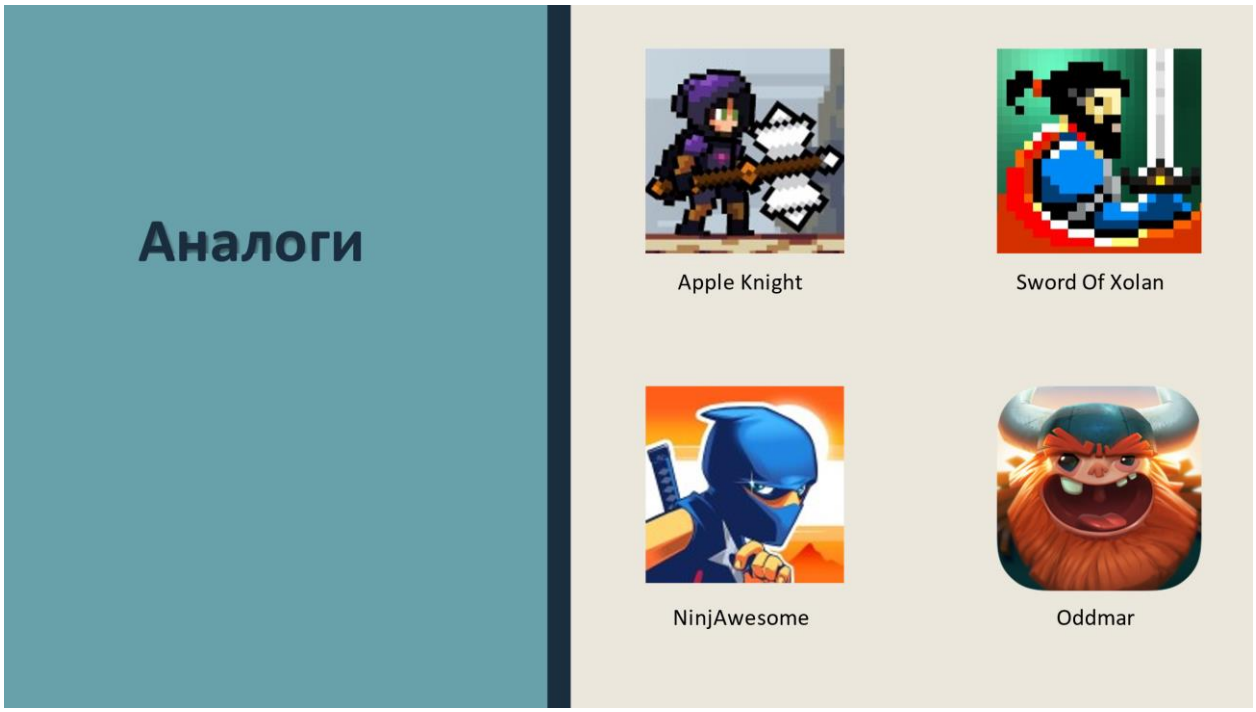

Рисунок Г.5 – Аналоги

Порівняння аналогів

Функції	Apple Knight	Sword Of Xolan	NinjAwesome	Oddmar	Dark Medieval
Присутність різноманітних пасток	-	+	+	-	+
Вороги, що переслідують головного персонажа	+	-	+	+	+
Не потрібно використовувати Інтернет з'єднання	+	+	+	+	+
Відсутність внутрішніх покупок	-	-	-	-	+
Гра є безкоштовною	+	+	+	+	+

Рисунок Г.6 – Порівняння аналогів

Метод поведінки ворогів у ігровому середовищі

1. **Патрулювання.** Ворог патрулює задану йому місцевість ігрового середовища, із заданою швидкістю рухаючись з однієї точки до іншої. Рух здійснюється за допомогою компонента «Transform», який дозволяє рухати об'єкти за допомогою двовимірного вектору. Під час ініціалізації ворога задаються параметри швидкості, радіус видимості, радіус атаки, відстань руху від точки та сама точка, яка є окремим об'єктом на сцені. Одна й та сама точка може бути вказана в декількох ворогів.
2. **Переслідування.** За допомогою методу «Distance» аналізується відстань до головного персонажу, якщо відстань менше заданої видимості, то ворог розпочинає переслідування і рухається в сторону персонажа. Під час руху ворог за допомогою методу «OverlapCircleAll» записує у масив список усіх об'єктів, що знаходяться в радіусі його атаки, далі за допомогою циклу «foreach» відбувається проходження по масиву об'єктів та перевіряється назва, якщо назва об'єкту «Player», то ворог наносить удар. Далі щоб ворог не міг нескінченно наносити удари, за допомогою класу «Time» відбувається перезарядка.
3. **Повернення на позицію.** Якщо під час аналізу відстані за допомогою методу «Distance», головний персонаж вийде з радіусу видимості ворога, то ворог розпочне рух в сторону точки патрулювання.

Рисунок Г.7 – Метод поведінки ворогів у ігровому середовищі

Блок-схема алгоритму методу поведінки ворогів у ігровому середовищі

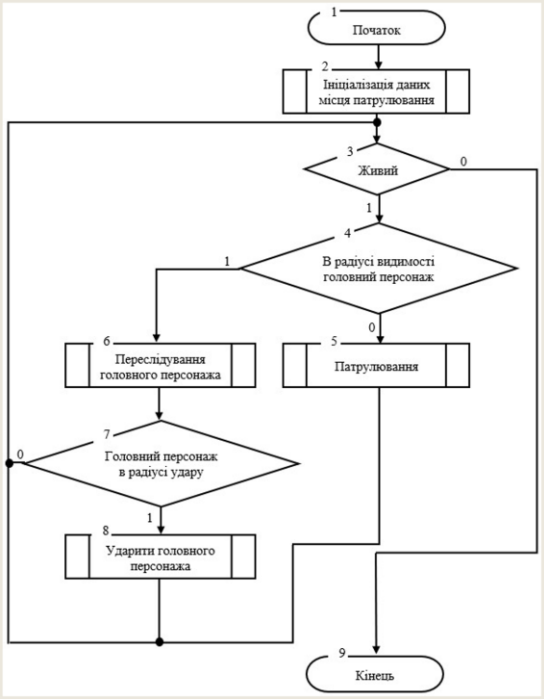

Рисунок Г.8 – Блок-схема алгоритму методу поведінки ворогів у ігровому середовищі

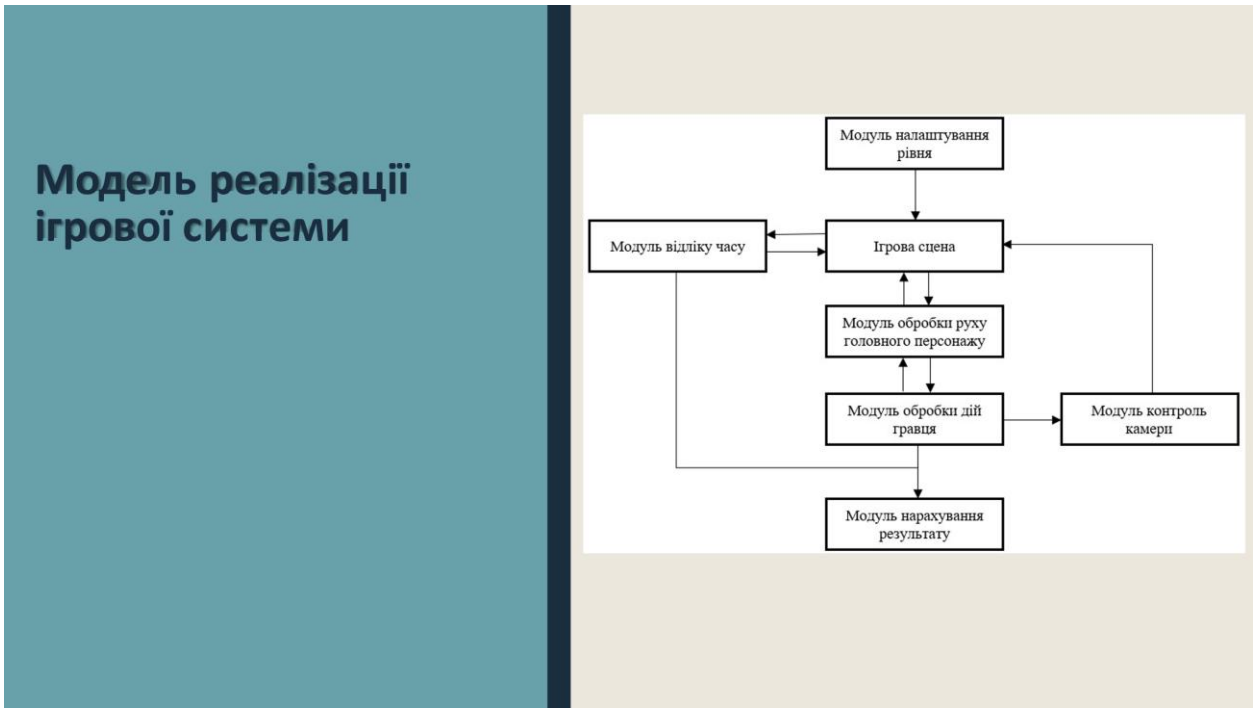

Рисунок Г.9 – Модель реалізації ігрової системи

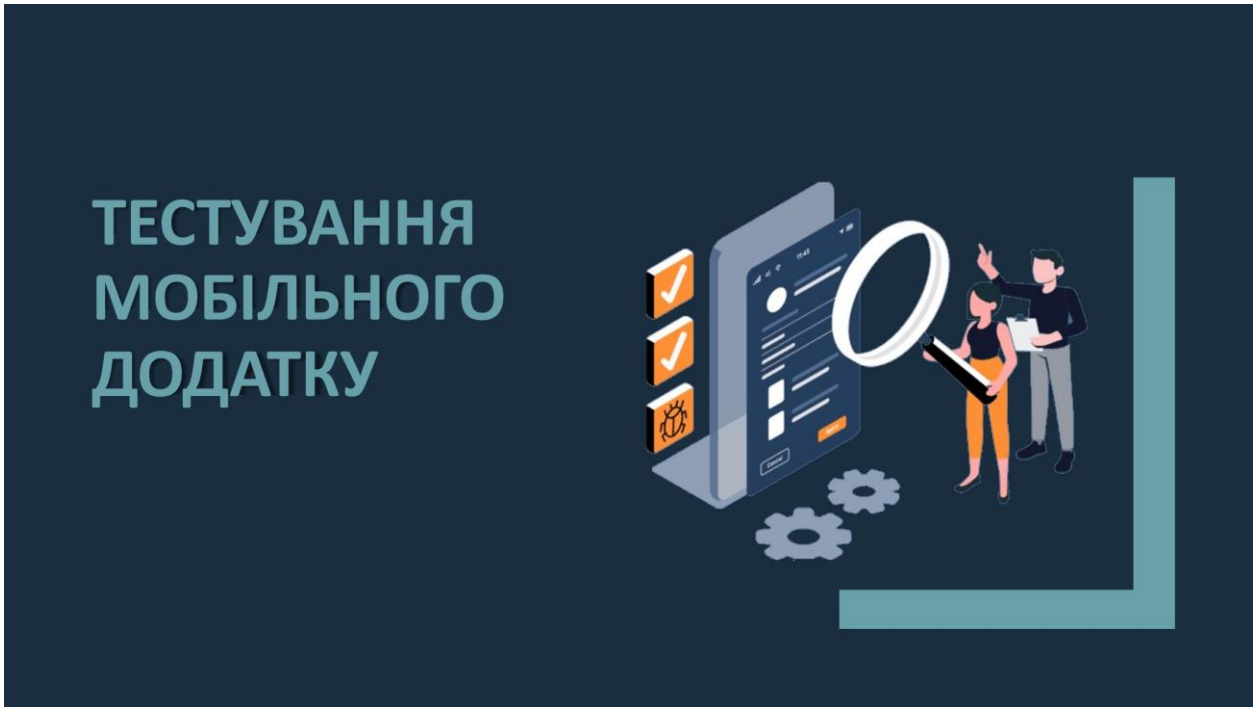

Рисунок Г.10 – Тестування мобільного додатку

Рисунок Г.11 – Тестування: головне меню гри

Рисунок Г.12 – Тестування: запуск рівня

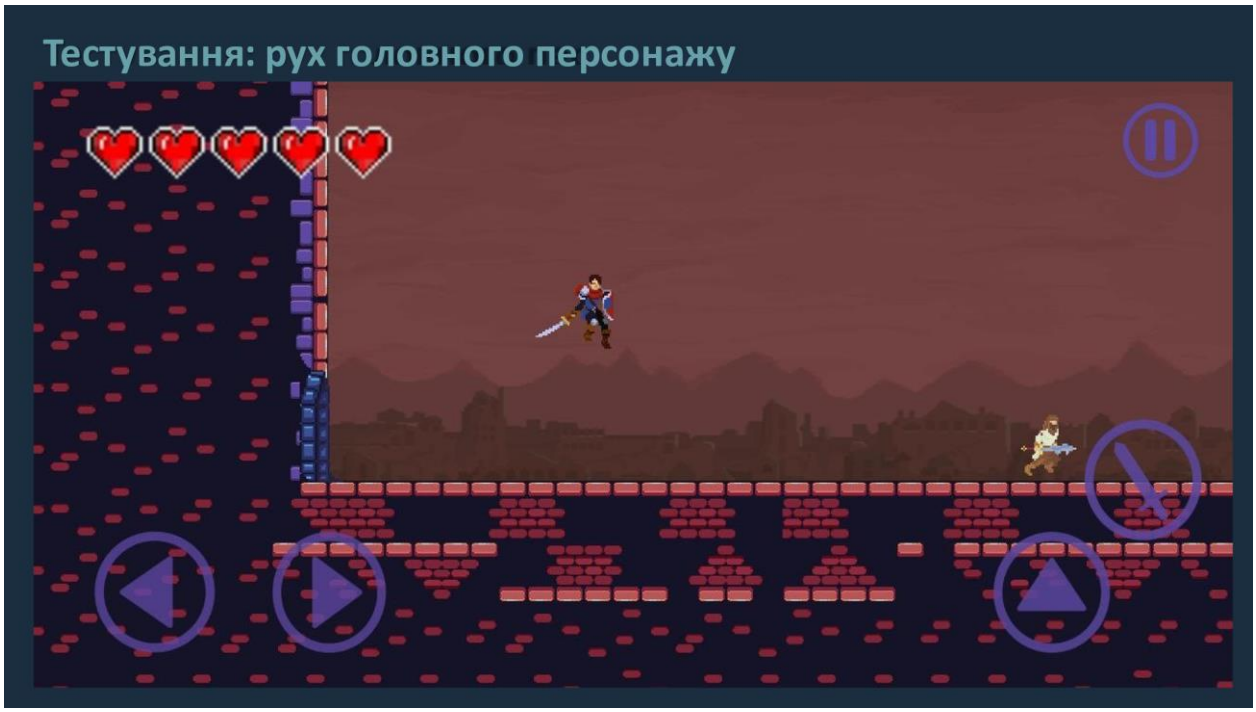

Рисунок Г.13 – Тестування: рух головного персонажу

Рисунок Г.14 – Тестування: атака

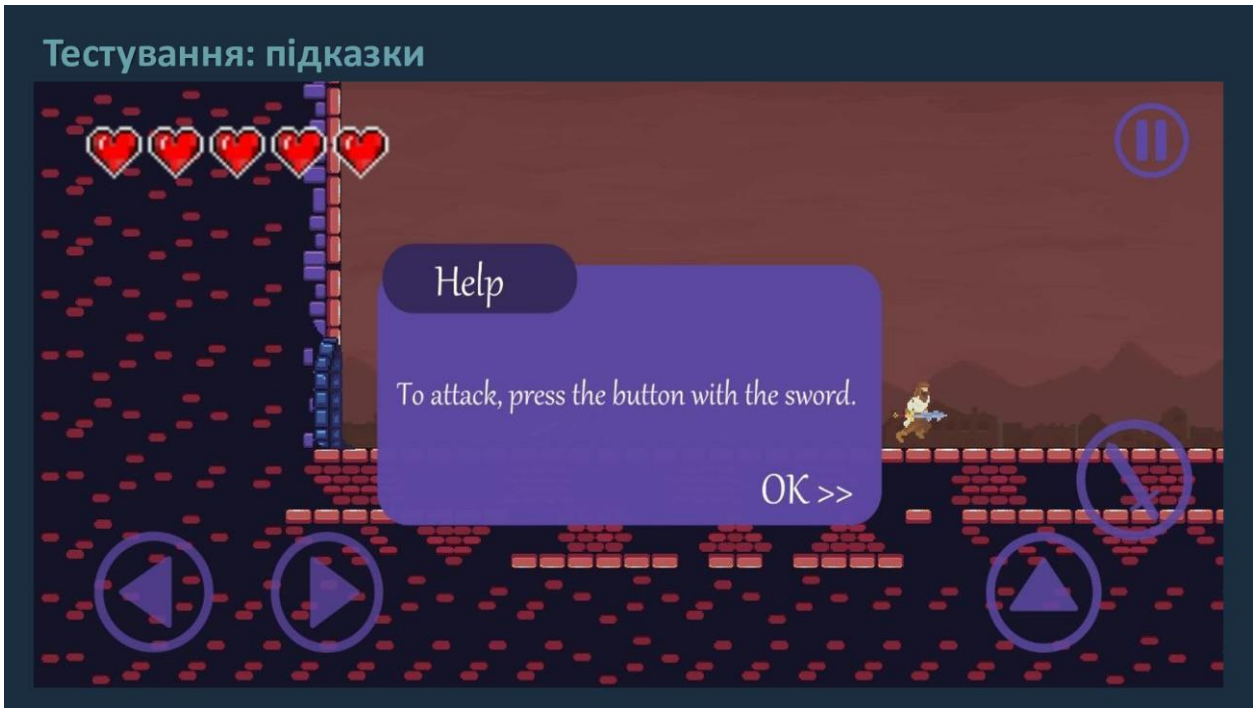

Рисунок Г.15 – Тестування: підказки

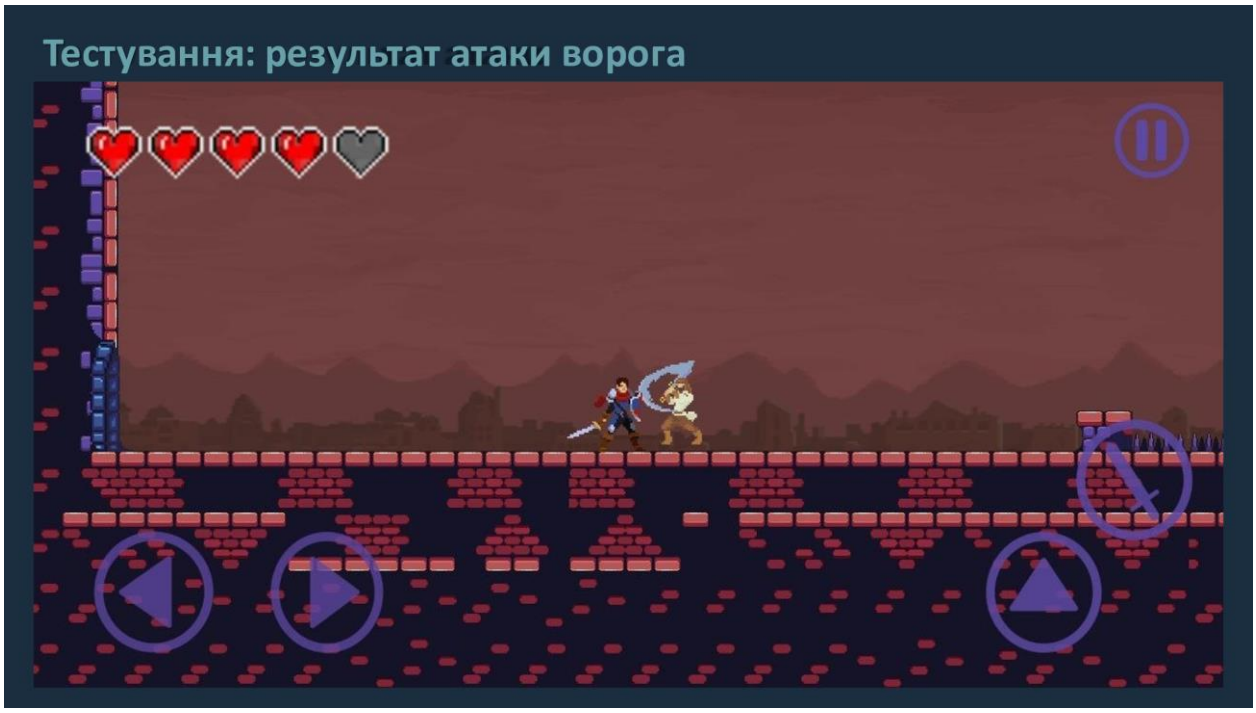

Рисунок Г.16 – Тестування: результат атаки ворога

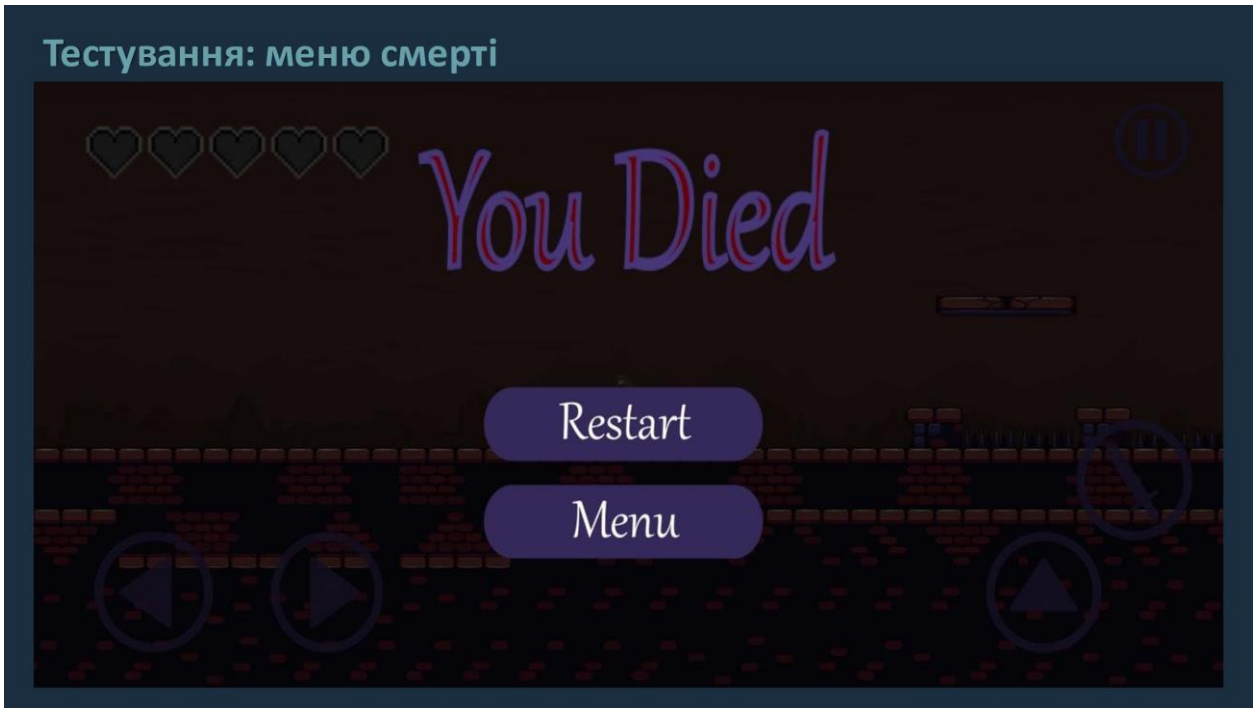

Рисунок Г.17 – Тестування: меню смерті

Рисунок Г.18 – Тестування: меню паузи

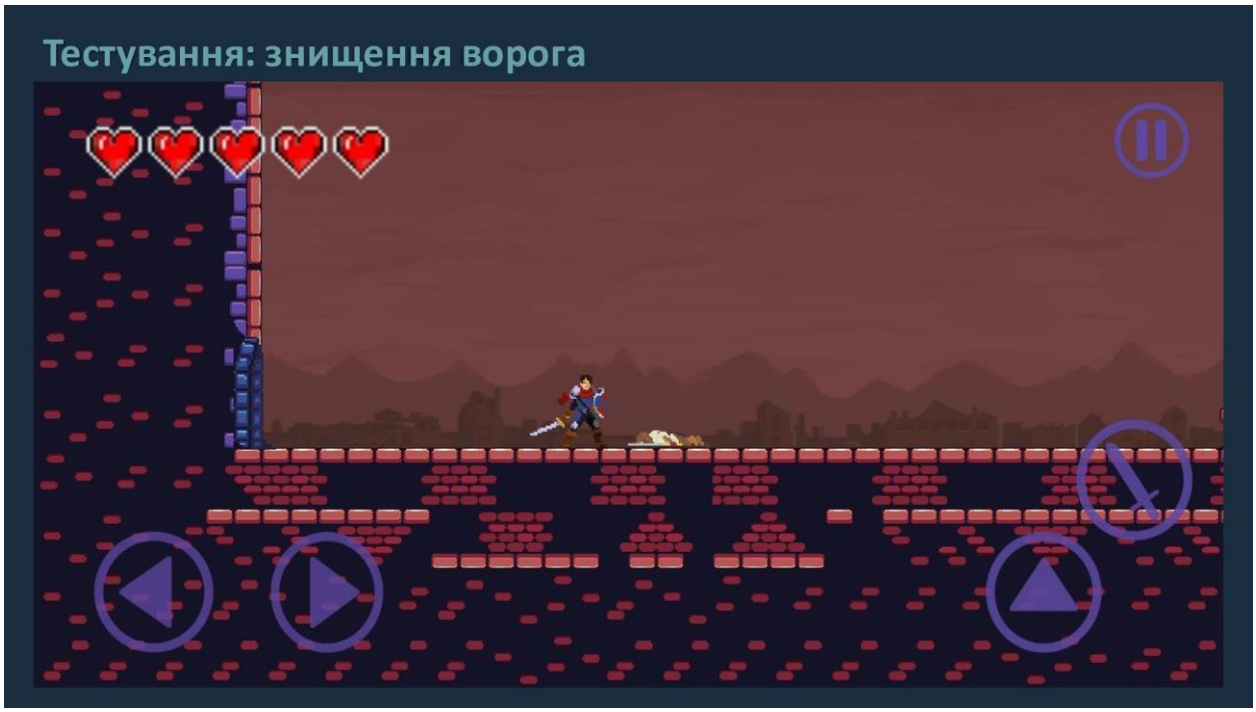

Рисунок Г.19 – Тестування: знищення ворога

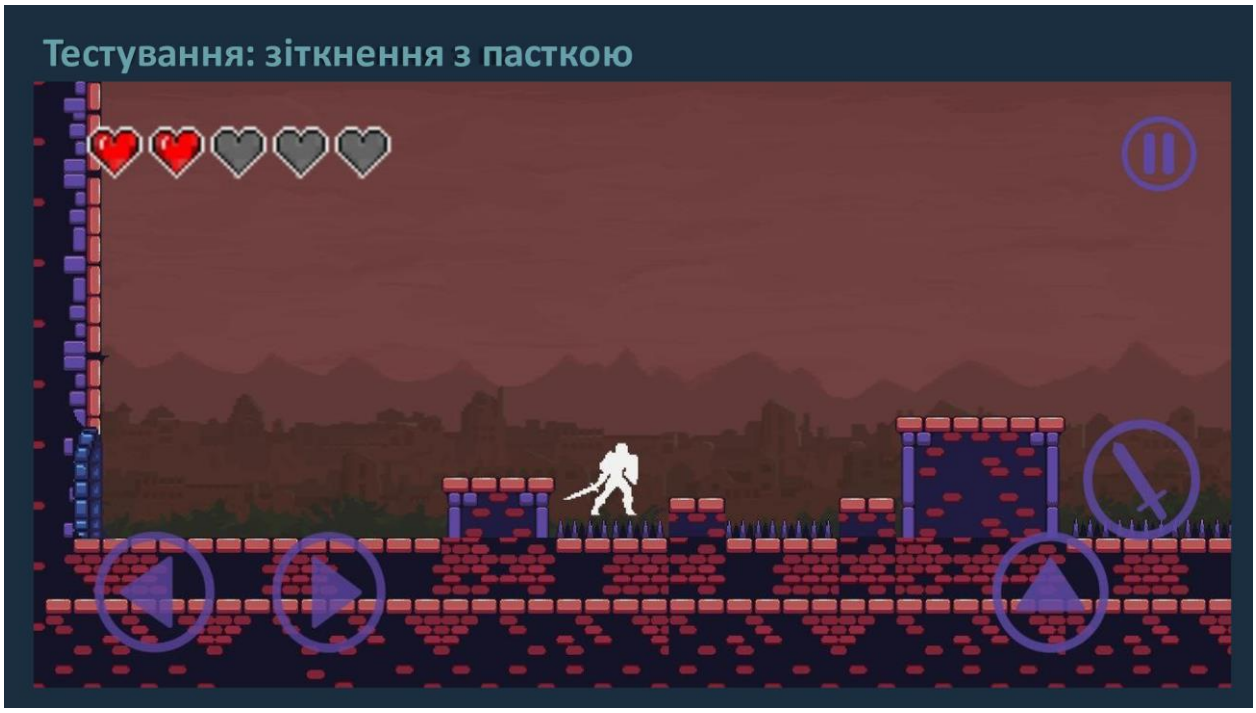

Рисунок Г.20 – Тестування: зіткнення з пасткою

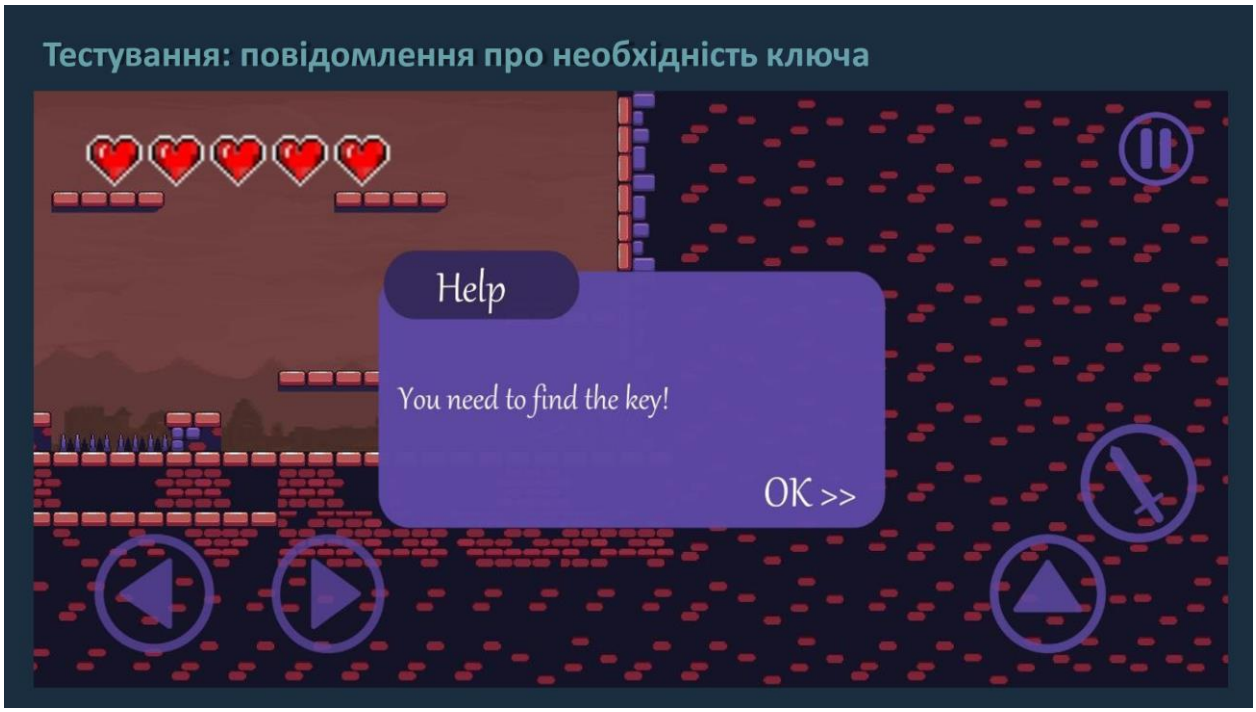

Рисунок Г.21 – Тестування: повідомлення про необхідність ключа

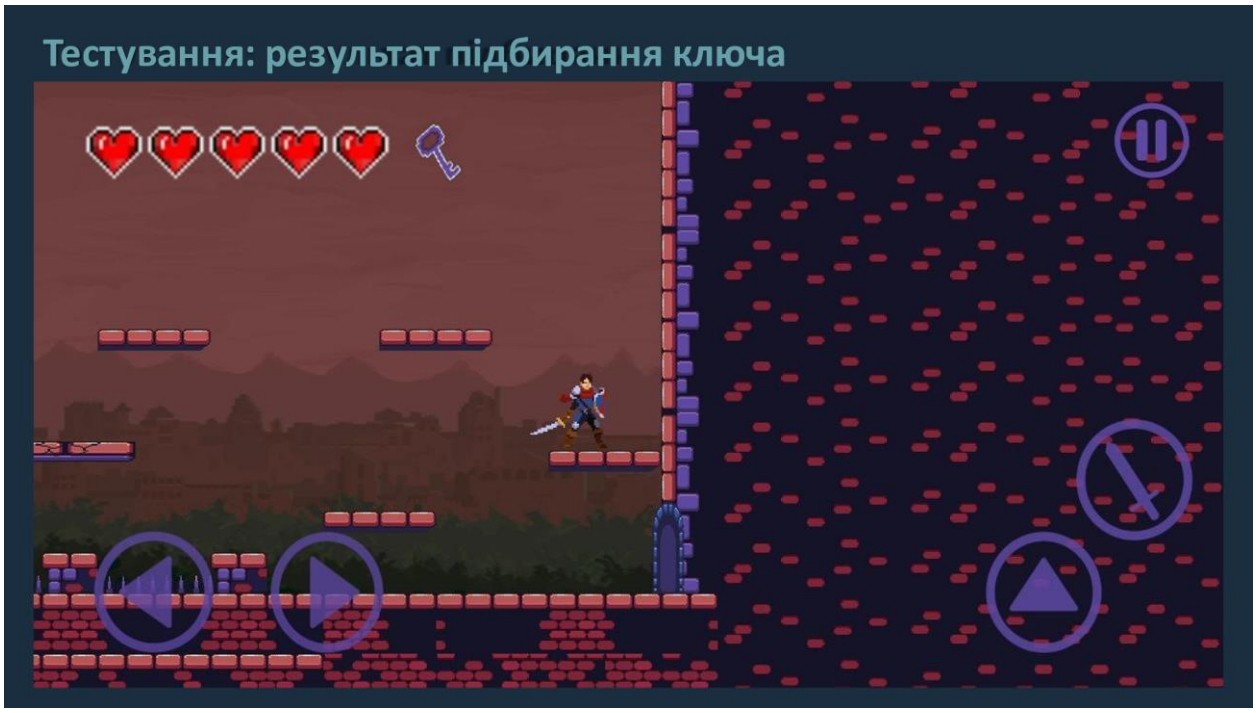

Рисунок Г.22 – Тестування: результат підбирання ключа

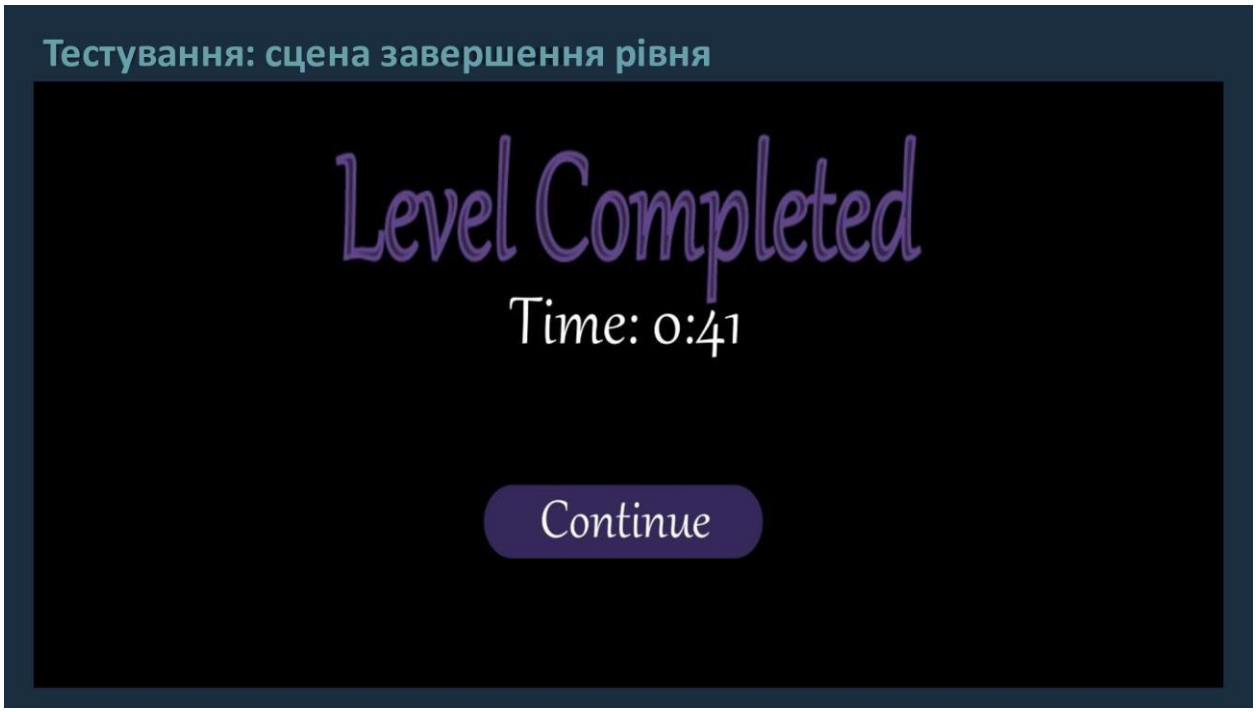

Рисунок Г.23 – Тестування: сцена завершення рівня

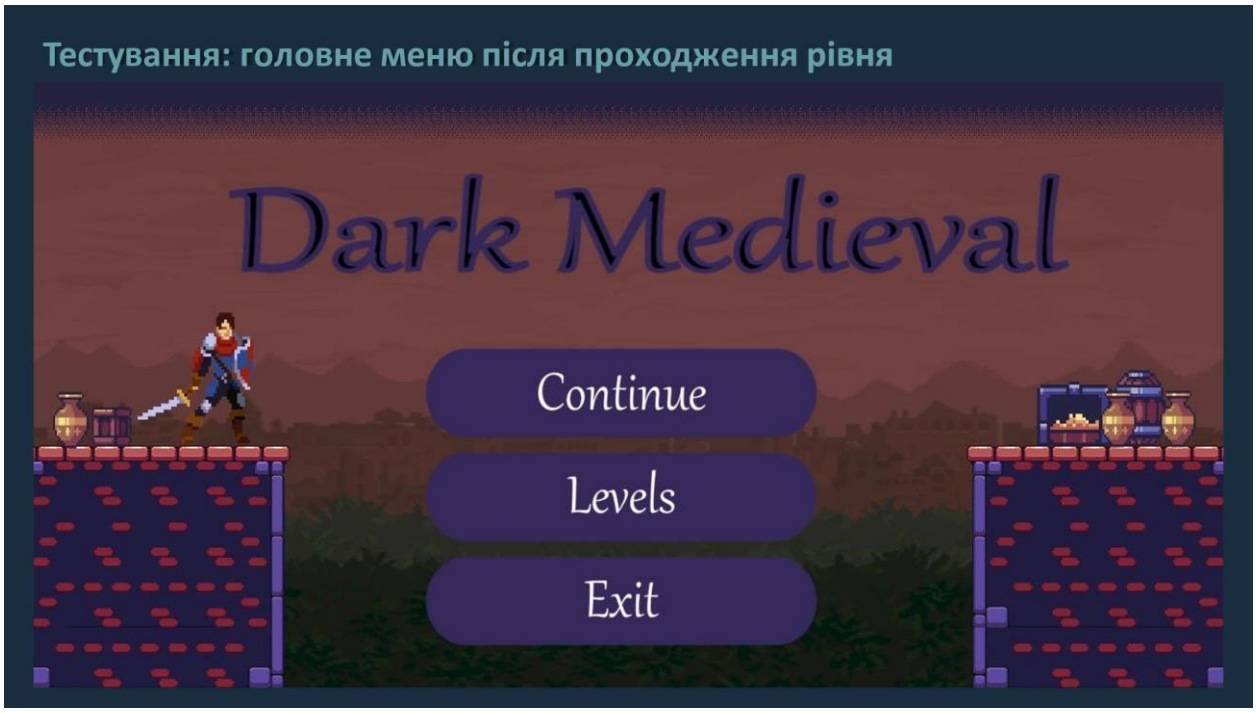

Рисунок Г.24 – Тестування: головне меню після проходження рівня

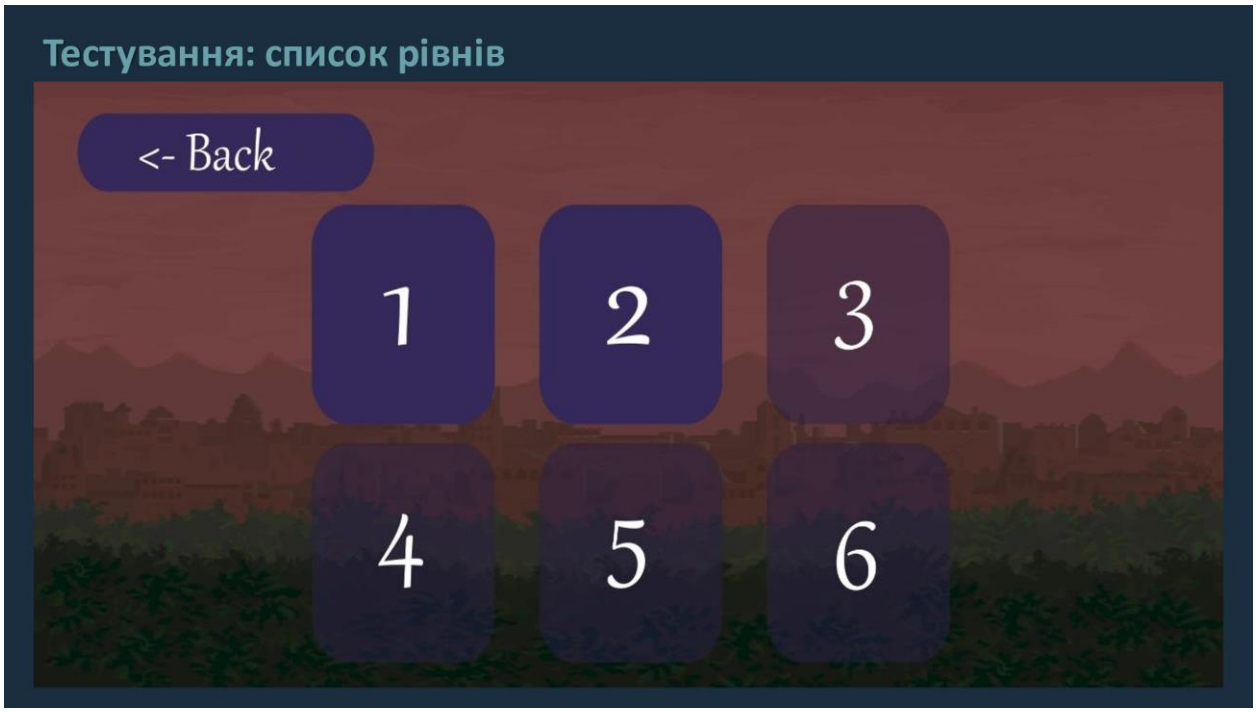

Рисунок Г.25 – Тестування: список рівнів

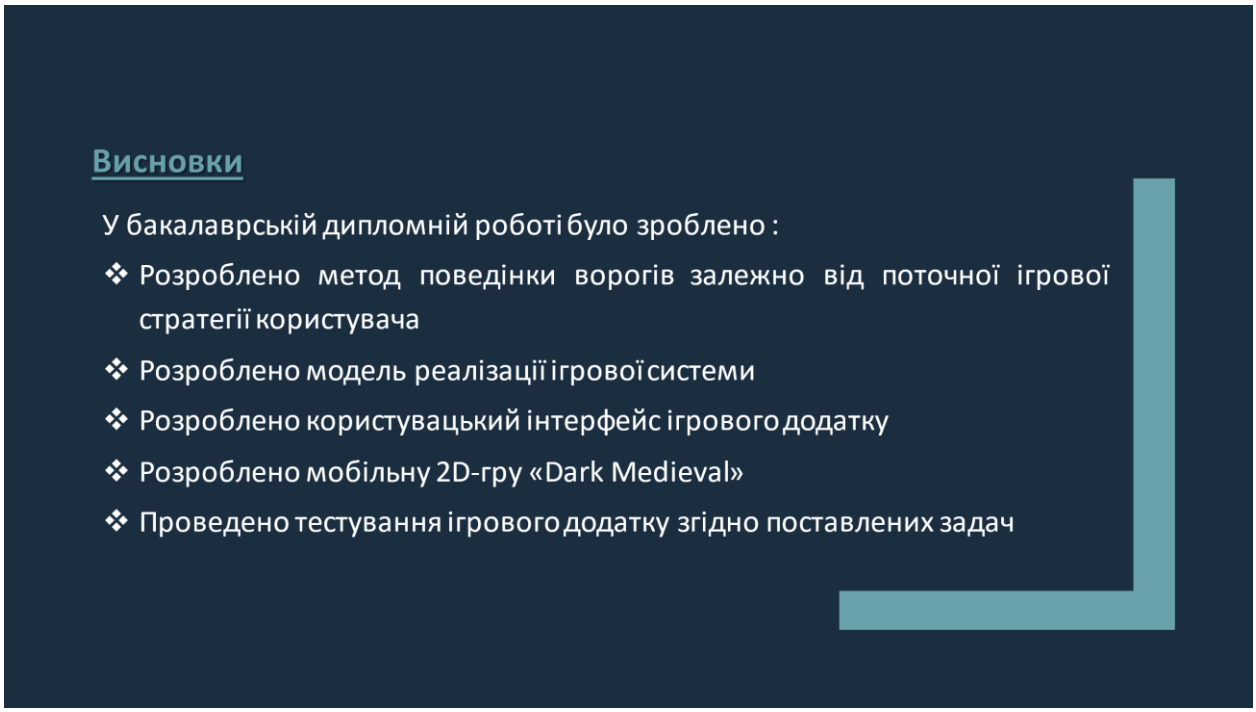

Рисунок Г.26 – Висновки

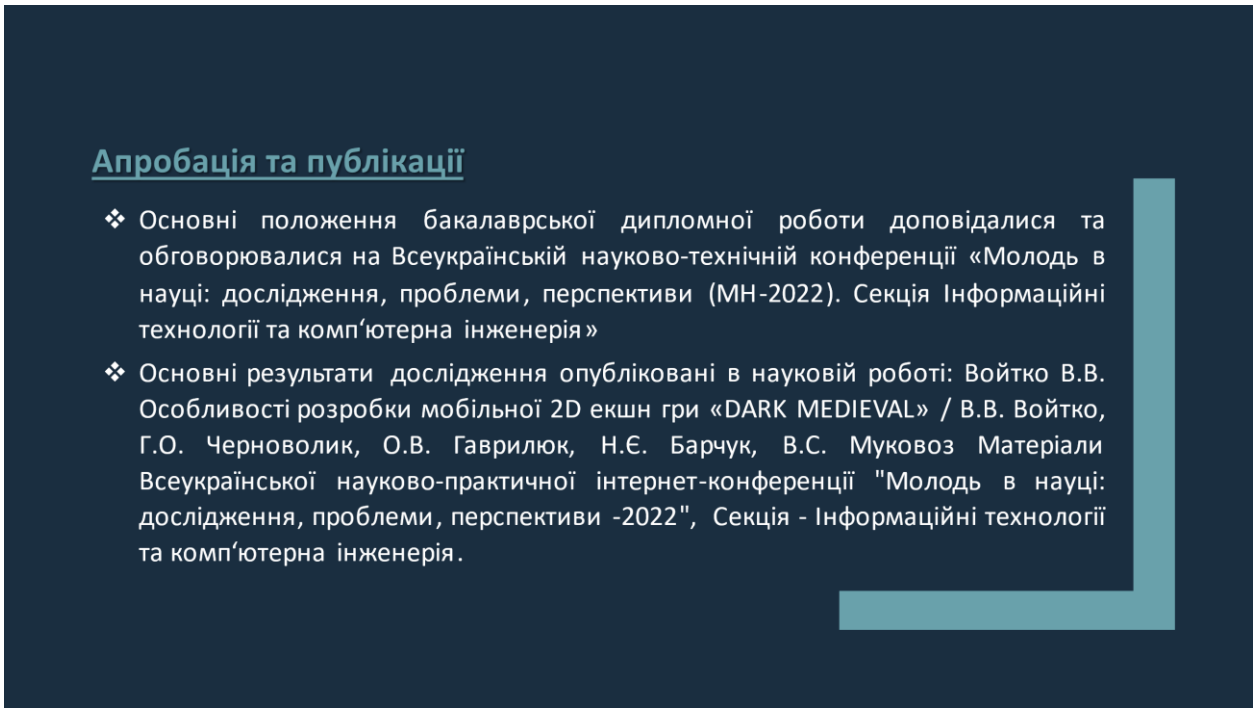

Рисунок Г.27 – Апробація та публікації