НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра інформатики та програмної інженерії
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2023 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення комп'ютеризованих систем»

спеціальності «121 Інженерія програмного забезпечення»

на тему: Комп'ютерна дитяча гра для розвитку друкування та моторики рук

Виконав студент IV курсу, групи ІП
(шифр групи)

Бебех Олександр Олександрович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник ст.вик. Халус О. А. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з графічної документації доцент, к.т.н., доц., Ліщук К. І. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент доцент кафедри ОТ, доц., Волокита А.М. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ –2023

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 Інженерія програмного забезпечення

Освітньо-професійна програма – Інженерія програмного забезпечення комп'ютеризованих систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2023 р.

ЗАВДАННЯ
на дипломний проєкт студенту

Бебеху Олександру Олександровичу
(прізвище, ім'я, по батькові)

1. Тема проєкту комп'ютерна дитяча гра для розвитку друкування та моторики рук

керівник проєкту Халус Олена Андріївна, ст.вик.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «2» червня 2023 р. №2160-с

2. Термін подання студентом проєкту « 19 » червня 2023 року

3. Вихідні дані до проєкту: технічне завдання

4. Зміст пояснювальної записки
1) Загальні положення: основні визначення та терміни, опис предметного середовища, огляд ринку програмних продуктів, постановка задачі.
2) Інформаційне забезпечення: вхідні дані, вихідні дані, опис структури ігрових сцен.
3) Програмне та технічне забезпечення: засоби розробки, вимоги до технічного забезпечення, архітектура програмного забезпечення, побудова звітів.
4) Технологічний розділ: керівництво користувача, методика випробувань програмного продукту.

5. Перелік графічного матеріалу

1) Схема структурна варіантів використання _____

2) Схема структурна класів програмного забезпечення _____

3) Схема структурна станів системи _____

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «10» березня 2022 року _____

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	10.03.2023	
2	Аналіз існуючих методів розв’язання задачі	20.03.2023	
3	Постановка та формалізація задачі	01.04.2023	
4	Розробка інформаційного забезпечення	03.04.2023	
5	Алгоритмізація задачі	07.04.2023	
6	Обґрунтування вибору використаних технічних засобів	10.04.2023	
7	Розробка програмного забезпечення	24.05.2023	
8	Налагодження програми	27.05.2023	
9	Виконання графічних документів	30.05.2023	
10	Оформлення пояснювальної записки	01.06.2023	
11	Подання ДП на попередній захист	05.06.2023	
12	Подання ДП рецензенту	12.06.2023	
13	Подання ДП на основний захист	19.06.2023	

Студент _____ Олександр БЕБЕХ

(підпис) (ініціали, прізвище)

Керівник _____ Олена ХАЛУС

(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка до дипломного проекту: 27 таблиць, 20 рисунків та 10 джерел – загалом 93 сторінки.

Дипломний проєкт присвячений створенню комп’ютерної дитячої гри для розвитку навичок печаті та моторики рук, що може бути застосовано для ознайомленню із клавіатурою та навчання печаті дітей віком від 6 до 12, а також ровитку моторики рук у ігровій формі.

Об'єкт дослідження: діти віком від 6 до 12 років.

Предмет дослідження: комп’ютерна гра розроблена з допомогою ігрового рушія Unity.

У розділі аналіз вимог до програмного забезпечення було проаналізовано вебігри та комп’ютерні ігори для розвитку печаті, було виділено перваги та недоліки, та на їх основі було сформульовано вимоги до програмного забезпечення.

Розділ моделювання та конструювання програмного забезпечення присвячений опису інструментів для створення комп’ютерних ігор, архітектурі та компонентам ігрового застосунку.

У розділі аналіз якості та тестуванню програмного був описаний план тестування створеного програмного забезпечення.

У розділі впровадження та супровід програмного забезпечення були описані вимоги до систем, на яких коректно працює програмне забезпечення та описано взаємодію з розробленою програмою.

КЛЮЧОВІ СЛОВА: КОМП’ЮТЕРНА ГРА, ІГРОВИЙ РУШІЙ, UNITY, ПЕЧАТЬ.

ABSTRACT

Explanatory note to the degree project: 27 tables, 20 figure, and 10 sources – in general 93 pages.

The purpose of the diploma project is to create a computer game for children to develop typing skills and hand motility, which can be used to introduce the keyboard and teach typing to children aged 6 to 12 years old, as well as develop hand motility in a game form.

Object of study: children aged 6 to 12 years old.

Subject of study: the computer game is developed using the Unity game engine.

In the software requirements analysis section, web-games and computer games for print development were analyzed, advantages and disadvantages were highlighted, and software requirements were formulated based on them.

The section on software modeling and design is devoted to describing the tools for creating computer games, the architecture and components of a game application.

In the section on quality analysis and software testing, a test plan for the created software was described.

In the section on implementation and maintenance of the software, the requirements for the systems on which the software works correctly were described, and the interaction with the developed program was described.

KEY WORDS: COMPUTER GAME, GAME ENGINE, UNITY, TYPING.

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІП-8202.045480.01.91	Технічне завдання	13	
3	А4	КПІ.ІП-8202.045480.02.81	Пояснювальна записка	93	
4	А4	КПІ.ІП-8202.045480.03.13	Опис програми	38	
5	А4	КПІ.ІП-8202.045480.04.51	Програма та методика тестування	6	
6	А4	КПІ.ІП-8202.045480.05.34	Керівництво користувача	10	
7	А3	КПІ.ІП-8202.045480.06.99	Схема структурна варіантів	1	
			використань		
8	А3	КПІ.ІП-8202.045480.07.99	Схема структурна класів	1	
			програмного забезпечення		
9	А3	КПІ.ІП-8202.045480.08.99	Схема структурна станів системи	1	

Змін.	Арк.	№ докум.	Підп.	Дата	КПІ.ІП-8202.045480.00.90		
Розроб.		Бебех О. О.			Відомість дипломного проєкту	Літ.	Аркуш 1 / Аркушів
Перевір.		Халус О. А.					
Н.контр.		Ліщук К.І.				КПІ ім. Ігоря Сікорського ФІОТ каф. ІПІ гр. ІП-82в	
Затв.		Жаріков Е.В.					

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

Комп'ютерна дитяча гра для розвитку друкування та моторики рук

Технічне завдання

КПІ.ІП-8202.045480.01.91

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олена ХАЛУС

Нормоконтроль:
_____ Катерина ЛІЩУК

Виконавець:
_____ Олександр БЕБЕХ

Київ – 2023

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: комп'ютерна дитяча гра для розвитку друкування та моторики рук.

Галузь застосування: розважальна індустрія, навчальна газуль.

Наведене технічне завдання поширюється на розробку комп'ютерна дитяча гра для розвитку друкування та моторики рук [КПІ.ІП-8202.045480], яке може бути застосовано для ознайомлення із клавіатурою та навчання печаті дітей віком від 6 до 12, ровитку моторики рук у ігровій формі, а також у розважально-навчальниз цілях.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки комп'ютерної дитячої гри для розвитку друкування та моторики рук є завдання на дипломне проєктування, затверджене кафедрою інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для зацікавлення дітей віком від 6 до 12 років у розвитку навичок печаті та моторики рук.

Метою розробки є створення простого у встановленні та використані програмного забезпечення, яке здатне ставити перед користувачем задачі, для вирішення яких потрібно опановувати та розвивати навички печаті та розвивати моторику рук.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати виконання наступних основних функцій:

4.1.1 Користувацького інтерфейсу:

- початок нової гри, на Рисуноку 4.1 кнопка New Game, перехід до ігорової сцени 1-го рівня, Рисунок 4.2;
- продовження гри після досягнення певного прогресу, Рисунок 4.1 кнопка Continue, перехід до ігорової сцени збереженого рівня, Рисунок 4.2;
- відображення стану ігрового рівня, Рисунок 4.2;
- відображення ігрового персонажа, Рисунок 4.2;
- відображення параметрів гравція, а саме кількість очок здоров'я, кількість ігрової валюти, радіус атаки, зображені на Рисунку 4.2 позначені символом X;
- перехід до бойової сцени із стану зображеного на Рисунку 4.2 до сцени на Рисунку 4.3;
- відображення залишку слів, верхня частина Рисунку 4.3;
- відображення кількості очок здоров'я, лівий нижній кут Рисунку 4.3;
- відображення кількості допустимих помилок, правий нижній кут Рисунку 4.3;
- відображення ігрового персонажа та радіусу атаки, центр Рисунку 4.3;

Рисунок 4.1 – Початкове меню

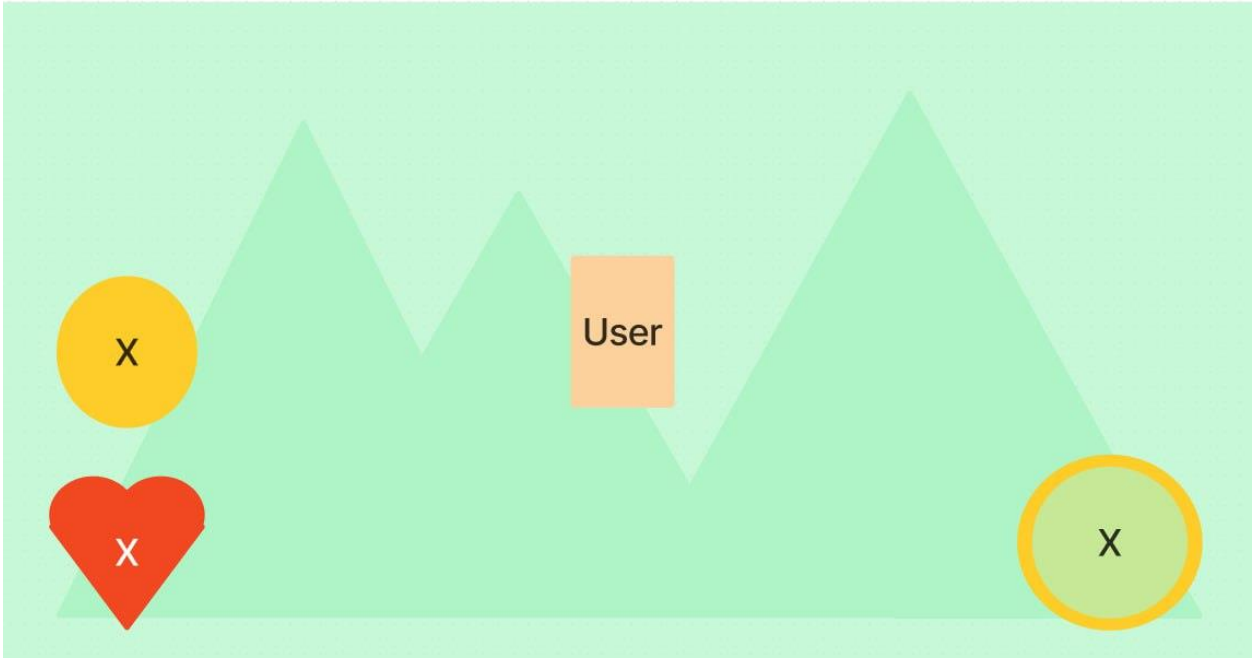


Рисунок 4.2 – Ігровий рівень 1...N

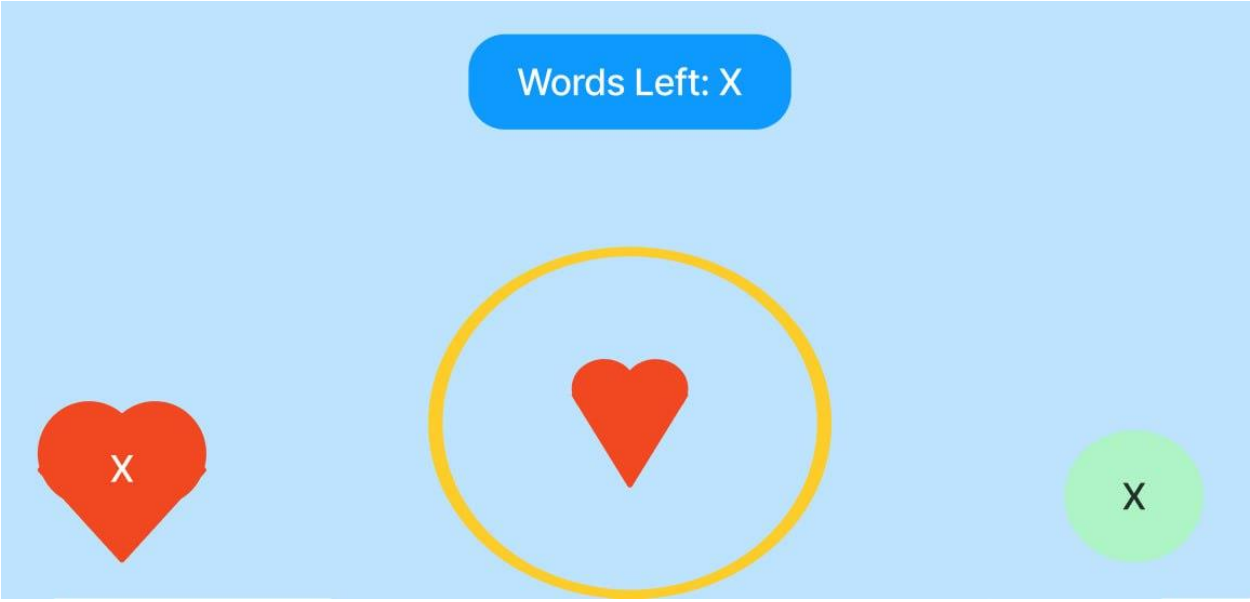


Рисунок 4.3 – Бойова сцена

4.1.2 Для користувача:

– створення та відображення ігрових сцен;
– обробка вводу користувача;
– збереження прогресу користувача;
– винагородження користувача;
– поступове збільшення складності гри;
– система діалогів для повідомлення інформації користувачу;
– система виконання завдань;
– перехід на наступні рівні;
– бойові сцени із використанням печаті;
– відображення актуальної бойової сцени із інформацією про необхідні символи для друкування, залишку слів, кількості очок здоров’я та допустимих помилок;
– зупинка гри у будь-який момент.

4.1.3 Розробку виконати на платформі .NET з використанням ігрового

рушія Unity.

4.1.4 Додаткові вимоги:

– встановлена операційна система Windows 10.

4.2 Вимоги до надійності

4.2.1 Передбачити контроль введення інформації.

4.2.2 Передбачити захист від некоректних дій користувача.

4.3 Умови експлуатації

4.3.1 Обслуговування

Умови до обслуговування не висуваються.

4.3.2 Обслуговуючий персонал

Програмне забезпечення не потребує обслуговуючого персоналу.

4.4 Вимоги до складу і параметрів технічних засобів

4.4.1 Програмне забезпечення повинно функціонувати на комп‘ютерах із операційною системою Windows.

4.4.2 Мінімальна конфігурація технічних засобів:

– тип процесору Pentium;
– об‘єм ОЗП 512 Мб;
– об‘єм вільного дискового простору 256 Мб;
– серія графічного процесора G8x або пізніші.

4.5 Вимоги до інформаційної та програмної сумісності

4.5.1 Програмне забезпечення повинно працювати під управлінням операційних систем Windows.

4.5.2 Вхідні дані повинні бути введені в екранні клавіші та із

клавіатури.

4.5.3 Результати повинні бути у вигляді змін стану комп’ютерної гри у відповідь на вхідні дані.

4.5.4 Програмне забезпечення повинно бути розроблено з використанням мови програмування C#.

4.5.5 Програмне забезпечення повинно бути розроблено з використанням системи контролю версій git.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не пред‘являються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не пред'являються.

4.8 Спеціальні вимоги

Згенерувати установчу версію програмного забезпечення.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

5.2 Програмне забезпечення повинно мати довідникову систему.

5.3 У склад супроводжувальної документації повинні входити наступні документи:

5.3.1 Пояснювальна записка не менше ніж на 50 аркушах формату А4 (без додатків 5.3.2 - 5.3.5).

5.3.2 Технічне завдання.

5.3.3 Керівництво користувача.

5.3.4 Програма та методика тестування

5.3.5 Варіанти використання

5.4 Графічна частина повинна бути виконана на 3 аркушах формату А3, котрі включаються у якості додатків до пояснювальної записки:

5.4.1 Схема структурна варіантів використання.

5.4.2 Схема структурна класів.

5.4.3 Схема функціональна послідовності.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення рекомендованої літератури	10.03	
2.	Аналіз існуючих методів розв’язання задачі	20.03	Опис існуючих рішень
3.	Постановка та формалізація задачі	01.04	Специфікації програмного забезпечення
4.	Розробка інформаційного забезпечення	03.04	Перелік вимог
5.	Алгоритмізація задачі	07.04	Схема структурна класів програмного забезпечення
6.	Обґрунтування вибору використаних технічних засобів	10.04	
7.	Розробка програмного забезпечення	24.05	Тексти програмного забезпечення
8.	Налагодження програми	27.05	Текст програми та методики тестування
9.	Виконання графічних документів	30.05	Графічні матеріали проекту
10.	Оформлення пояснювальної записки	01.06	Текст пояснювальної записки
11.	Подання ДП на попередній захист	05.06	
12.	Подання ДП рецензенту	12.06	
13.	Подання ДП на основний захист	19.06	

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Пояснювальна записка
до дипломного проєкту

на тему: Комп'ютерна дитяча гра для розвитку друкування та моторики рук

КПІ.ІП-8202.045480.02.81

Київ – 2023

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

Гравець – людина, яка є активним учасником комп’ютерної гри.

Ігровий рушій – є програмною платформою, призначеною в першу чергу для розробки відеоігор і, як правило, включає відповідні бібліотеки та програми підтримки.

Геймплей – це специфічний спосіб взаємодії гравців із грою, зокрема з відеоіграми. Геймплей — це шаблон, визначений правилами гри, зв’язком між гравцем та грою, викликами та їх подоланням, сюжетом та зв’язком гравця з нею.

Ігрова механіка – це правила, які керують і спрямовують дії гравця, а також реакцію гри на них. Таким чином, механіка гри ефективно визначає, як гра працюватиме для людей, які в неї грають.

Рівень – певна область віртуального(ігрового) світу.

Моб – скорочення від словосполучення “мобільний об’єкт”, персонаж, яким керує комп’ютер.

UI – (від англ. User Interface) спосіб взаємодії користувача із застосунком за допомогою різних графічних елементів.

ВСТУП

У сучасному світі багато людей проводить чимало часу за різними пристроями з вводом за допомогою клавіатури, такими як персональні комп'ютери, телефони, планшети, ігрові приставки та інше. Починаючи із спілкування через пошту і закінчуючи набором звітів, навик друкування є необхідним для комфортного користування пристроями. Чимало людей відчувають незручності від того, що повільно друкують, через брак досвіду та відповідних моторних навичок. Багато людей у достатньо пізньому віці стикаються із тим, що потрібно друкувати у достатньо великих об'ємах, щоб покращити свої результати. З іншого боку діти проводять достатньо багато часу, опановуючи навички набору в ігровій формі, що можна використати для набування корисних умінь для подальшого життя.

Отже, бажано заохотити дітей до навчання та набування корисних навичок, а також розвивати ці навички для подальшого використання. Немаловажливим є і підтримка вже набутих навичок, та закріплення їх у довготривалій перспективі.

Зараз існує чимало різних ігор для розвитку дітей, починаючи із ознайомлення з алфавітом і закінчуючи різними пізнавальними іграми.
Для навчання швидкої печаті є ігри на сайтах, де слова падають і потрібно їх набрати до моменту їх падіння, але вони мало як можуть зацікавити дітей.
Існують також різні курси по друкуванню, де людей спеціаль змушують друкувати великі тексти і навчають різним технікам, але вони теж не підходять дітям, адже потребують зосередженості та усидливості. Спеціальні насадки на клавіатуру, які закривають букви, щоб запам’ятовувати де вони знаходяться, також варто застосовувати у подібних справах. Є сайти та програми для розвитку сліпої печаті та швидкого друку.

Тому, для покращення уже існуючих рішень, було вирішено створити ігровий застосунок, який міг би зацікавити та розвивати навички друкування користувачів.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Загальні положення

Із розвитком науки та технологій, сучасні пристрої, такі як персональний комп’ютер, є просто необхідністю для спілкування, роботи, навчання і навіть просто побуту. Згідно даним зібраним на сайті Statista кількість домашніх комп'ютерів у світі в період з 2005 по 2019 [1] щорічно зростає, а у Сполучених Штатах цей відстоток станом на 2016 рік складає 89% [2]. Такий ріст пов’язаний із розповсюдженням технологій та доступу до них, а також збільшенням кількості спеціалістів. У той же час найбільше користувачів персональних комп’ютерів серед людей середнього віку[3]. Відповідно кількість людей, які ознайомлюються із пристроями для роботи із комп’ютерами також росте. Цей процес потребує часу, практики та мотивації, для того щоб на комфортному рівні володіти пристроями вводу.

Через зростання доступності персональних комп’ютерів і пристроїв на їх основі, популярності набувають різні програми, застосунки та комп’ютерні ігри. Ігрова індустрія за останні десятиліття набула великої популярності. Настільки швидкі темпи росту застали зненацька багатьох менеджерів та аналітиків. «Чи коли-небудь відбувалися зміни в культурному морі, такі невидимі, як та, яку представляє зростання інтерактивних розваг? Для всіх, хто досяг повноліття, скажімо, після появи першої Sony Playstation у 1995 році, відеоігри є такими ж центральними у всесвіті поп-розваг, як фільми чи музика… Нікому не прийде в голову заперечувати, що відеоігри великі, але небагато дорослих людей за межами бізнесу розуміють, наскільки великими вони стали». (Ді, New York Times Magazine, 21 грудня 2003 р.).

Як наслідок кількість коштів, які витрачаються користувачами на ігри теж зростає, до прикладу з 2012 по 2019 загальний дохід збільшився більш, ніж у 2 рази[4].

На початку свого існування комп’ютерні ігри були складними у створенні та обслуговуванні. Зараз існує набагато більша кількість доступних інструментів для розробки ігор, що надає багато можливостей для розробників. Наразі головним таким інструментом є ігрові рушії. Це сприяє наявності великої кількості не тільки комерційних компаній, а й ентузіастів, які займаються розробкою.

Ігрові застосунки мають багато видів, в залежності від платформи, технології, жанру, вікової категорії і особливо кількості коштів, витрачених на виготовлення. Саме завдяки великій різноманітності та доступності ігрова індустрія має значну аудиторію. Якщо говорити про різноманітність детальніше, в першу чергу потрібно виділити ігрові жанри: шутери, стратегії, симулятори, пригодницькі, азартні, рольові, платформери, навчальні, казуальні, спортивні та інше. Кожен ігровий жанр має унікальні характеристики та технології для їх реалізації, що дозволяє знайти підхід до різноманітних категорій людей.

Сучасні відеоігри розраховані на всі вікові категорії, починаючи від малолітніх дітей і закінчуючи літніми людьми. Ігрове різноманіття дозволяє користувачам за бажанням обрати застосунки базуючись на основній тематиці, комп’ютерній графіці, сюжету та тривалості гри. Але ігри, так само як і фільми підпадають під вікові обмеження та класифікацію[5].

Ігри розробляються для усіх платформ та пристроїв. Найбільш популярними є Windows та Android. В залежності від платформи виникають різні особливості розробки, починаючи з особливостей програмного забезпечення і закінчуючи принципами роботи апаратного забезпечення. Також, варто зазначити про існування веб-ігор, які можна грати на будь-яких платформах, де можна встановити браузер. Дуже часто ігри створені для однієї платформи, не можуть запускатись на іншій, тому існує процес переведення на іншу платформу. А у деяких випадках це є неможливим, через відсутність достатніх характеристик пристроїв або програмного забезпечення.

Для ігрових застосунків існує багато видів монетизації, що дозволяє не тільки підтримувати уже існуючі продукти(виправлення помилок, додавання нового контенту, підтримка роботи серверів), а й фінансувати створення нових проектів. Такими видами отримання прибутку із застосунку є продаж власне самого застосунку та доповнень до нього, внутрішньоігровий магазин, реклама, пожертвування та підписка на продукт. Для принесення більшого прибутку потрібен персонал, який просуває застосунок в маси.

Таким чином, поставлена задача, яка полягає у створенні застосунку розрахована на дитячу аудиторію для розвитку навичок печаті, дуже добре комбінується із особливостями ігор, ціллю яких є якомога довше утримувати увагу користувача.

1.2 Виділення проблеми та визначення предмета дослідження

Сьогодні існує багато способів видів та способів навчання, особливо за допомогою персональних комп’ютерів. Цю задачу можна вирішити створенням веб-гри або ігрового застосунку. В обох випадках програмний продукт, дозволяє користувачеві виконувати основні задачі, а саме навчатись та вдосконалювати свої вміння.

Веб-гра для розробки та підтримування потребує наступне: база даних, де будуть зберігатись ігрові дані(налаштування противників, словник, дані користувачів); розгорнути сайт на системі хостингу та отримання сертифікату; кабінет адміністратора для підтримки сайту та реєстрація клієнта для збереження прогресу. Серед недоліків веб-ігор є необхідність у доступі до Інтернету для використання продукту. Також дані клієнта потрібно зберігати або в базі даних, або в кеші браузера. На телефонах або планшетах у браузерах часто виникають проблеми зі зручністю використання. Також виникає залежність від версії браузера та того, що він може підтримувати.

Щоб уникнути ці проблеми було вирішено створити саме ігровий застосунок. Для його постійної роботи не потрібен інтернет та прогрес гравця

зберігає застосунок на пристрої та не видаляється з часом. Також ігровий застосунок розробляється під конкретну платформу, тобто у ньому не буде конфліктів в залежності від пристрою на якому він запускається, оскільки на платформі для якої він не розрахований гра просто не запуститься. Як зазначалось вище кількість комп'ютерів, як і їх потужність, у домогосподарствах зростає, тож кожен бажаючий зможе завантажити та користуватись додатком.

1.3 Характеристики комп'ютерних ігор та їх особливості

У цьому розділі детальніше розглянемо види, жанри, вікові обмеження, сеттинг та платформи для яких розробляються ігри та інше.

Варто розпочати з того, без чого ігрові застосунки неможливо запустити - платформи. Ігрова платформа, для якої розробляється продукт, визначає інструменти, а відповідно і складність розробки. Кожна платформа має як свої переваги так і недоліки, до прикладу для ПК перевагою є розповсюдженість платформи а недоліком велика кількість різноманітних комбінацій комплектуючих обладнання. Не менш важливим є програмне забезпечення, адже саме завдяки цьому продукт може запускати та функціонувати. Загалом існують наступні платформи[6]:

- персональні комп'ютери – є універсальною платформою для будь-яких дій. Починаючи із розробки продуктів і закінчуючи їх використанням. Є достатньо громіздкими і потребують постійного живлення, але є ноутбуки, у яких цей недолік є незначним. Потребують багато додаткових пристроїв, а саме клавіатура, комп'ютера миша, монітори, динаміки, мікрофон та інше. Можна виділити перевагу у тому, що користувач може сам вибирати та замінювати комплектуючі на власний розсуд чи за необхідності. Для персональних комп'ютерів є багато варіантів операційної системи(ОС), серед них найпопулярніші Windows, Linux;

- мобільні пристрої - наразі є найрозповсюдженішими видами пристроїв, через свої розміри та багатофункціональність. З точки зору апаратного забезпечення вони є найбільш обмеженими, тому ігрові застосунки із деталізованою графікою або великою кількістю процесів не запустяться. Більшість ігор для цієї платформи є безкоштовними, а розробники отримують прибуток за рахунок внутрішньо ігрової реклами. Варто відмітити, що на цій платформі часто відбуваються помилки ніж на інших платформах;
- ігрові приставка(консоль) - спеціально створена для відеоігор платформа, кожна модель цієї платформи має визначені апаратні компоненти та програмне забезпечення спеціально під цю модель. Для консолей не варто враховувати можливість заміни компонентів на інший тип, тому через це не може бути конфліктів. Але це може бути і недоліком, оскільки при виході із ладу компонента заміна є ускладненою. Для ігрових консолей кожен розробник створює спеціальні контролери, розраховані виключно для ігор. Для цієї платформи є характерним висока ціна у порівнянні з іншими не тільки для пристрою а й для ігрових застосунків. До недоліків можна віднести відсутність зворотної сумісності, коли клієнт придбав консоль та гру для неї, але при купівлі наступного покоління консолі цю ж гру доводиться купувати знову, але це відноситься не до всіх розробників.

Прибуток від створеної продукції грає велику роль, оскільки завдяки цьому можна найняти більш кваліфікованих працівників, покращити робочі потужності та створити кращий продукт для подальшого отримання доходів. У випадку комп’ютерних ігор при розробці варто враховувати яку модель отримання прибутку розробники обирають. Адже в залежності від цього деякі аспекти гри можуть відрізнятись. Розрізняють два види моделей: модель авансового доходу та модель постійного доходу[4].

Авансова модель розрахована на дохід від авансових покупок гри, а пізніше і від покупок доповнень для цієї ж гри. Прикладами таких ігор є Sekiro: Shadow Die Twice, Half-Life та Omori.

Моделі постійного доходу націлені на отримання доходу протягом життєвого циклу гри. Виділяють наступні підтипи постійного доходу:

- Free-to-play (безкоштовна гра) – це підтип, за якого клієнт отримує продукцію безкоштовно та може її завантажити, але щоб отримати особливий контент або пришвидшити прогресс йому необхідно скористатись мікротранзакціями, або за допомогою платної сезонної підписки. Серед таких ігор є Genshin Impact та World Of Tanks;
- Buy-to-play (придбати для гри) – така модель передбачає покупку продукту за фіксовану плату, однак для отримання нових можливостей потрібно з певною періодичністю докуповувати новий контент. Як правило новий контент розробляється після виходу гри, заради підтримки інтересу користувачів. Ця модель застосовується для The Elder Scrolls Online;
- Pay-to-play (модель підписки) – у цій підмоделі гравцю потрібно сплачувати визначену ціну постійно для можливості користуватись продуктом та накопичувати прогрес. Яскравим прикладом є World of Warcraft;
- Freemium – підтип передбачає надання користувачеві певної частини продукту, так званої демо-версії, після її проходження пропонується купити повну версію. Freemium популярний серед мобільних ігор та невеликих проектів;
- основані на рекламі – підтип у якому продукт поширюється безкоштовно, але під час гри або для отримання певного контенту показують рекламу. Розповсюджено серед мобільних ігор.

Багато що в розробці комп’ютерних ігор визначає розробник та видавець. Від них залежить для якої аудиторії, платформи будуть

розроблятися ігрові застосунки, їх спосіб монетизації та скільки фінансів виділиться. Виділяють наступні види ігрових розробників:

- розробники, що виготовляють ігрові консолі та ексклюзиви для них. В ігровій індустрії не є рідкістю створення так званих ексклюзивів, комп'ютерних ігор призначених виключно для однієї або кількох платформ чи навіть пристроїв;
- розробники, які беруть контракти на розробку для певної платформи, або компанії;
- сторонні розробники, займаються розробкою ігор, але мають контракт з видавцем, який вносить корективи;
- інді-розробники, не мають фінансування від великих видавництв. Вони самі обирають що і як їм розробляти, але обмежені в ресурсах.

При розробці ігрового застосунку важливу роль відіграє сетинг(від англійського settings - антураж, обстановка) - сукупність компонентів, правил, подій та факторів, які формують та визначають світ у якому відбуваються дії. Термін як правило використовується для відеоігор. Серед елементів сетингу можна виділити час, події та місце. При чому одні і ті ж ігрові застосунки можуть бути реалізовані в різних сетінгах.

Щоб зацікавити гравця та надати йому мотивацію, гра повинна надавати виклик. Це може бути у вигляді заздалегідь визначених завдань, які є абсолютно однаковими під час гри. Він може виходити з попередньо визначеної структури, яка змінюється математичною випадковістю. Нарешті, опозиція може виникнути від ігрових агентів, чиї дії є автономними[9].

Цілі описують, чи є у грі точні й незмінні умови перемоги, чи цілі є суб'єктивними щодо унікальних випадків у конкретній грі чи інтерпретацій гравців[9].

Відеоігри розрізняють за взаємодією з іншими гравцями. Існують такі взаємодії[9]:

- одиночна гра передбачає відсутність взаємодії чи втручання інших гравців;
- гра з елементами кооперативу надає можливість проходити гру разом з іншими гравцями формуючи певну команду;
- ігри де гравець грає проти іншого гравця або команда проти команди часто базуються на суперництві та складаються з невеликих за тривалістю матчів;
- мультиплеєрні ігри розраховані на велику кількість людей, які знаходяться на одному просторі. Передбачаються різні взаємодії, а саме утворення команд, протистояння або просто відсутність взаємодій.

У іграх можна виділити те як плине внутрішньо ігровий час та як він залежить від реального часу. Існують ігри у яких із плином реального часу стан гри змінюється або ні. Далі чи можуть ігрові агенти діяти одночасно, чи по черзі. Контроль ігрового циклу може належати гравцеві або сама гра.

Варто відмітити, що у відеоігор є варіації в залежності від їх тривалості. Ігри можуть бути як із кінцевою ціллю, після чого гра закінчиться, або безкінечні, де гравець може продовжувати гру доки йому заманеться.

Ігрові застосунки піддаються віковим обмеженням так саме як і фільми. Наявність у іграх сцен або поведінки, яка включає, наприклад, ненормативну лексику, сцени сексуального характеру, вживання наркотичних речовин або реалістичного насильства, може негативно вплинути на психічний стан гравця, незалежно від його віку, тому важливо охарактеризувати рейтинг обмежень. При чому при оцінці обмежень варто враховувати всі можливі сценарії розвитку комп’ютерної гри, адже свобода дій під час ігрового процесу може надавати можливість пройти гру без вище перерахованих сцен. За основу було обрано класифікацію BBFC - є неурядовою організацією, заснованою кіноіндустрією в 1912 році і відповідальною за національну класифікацію та

цензуру фільмів, що демонструються в кінотеатрах та відео роботах. За віковим обмеженням можна виділити наступні класи[5]:

- «U» насильство буде м'яким і зазвичай знайоме з дитячих мультфільмів. Може бути особливе фантастичне довкілля з використанням не реалістичної зброї, і без зображень реалістичних поранень. Мова буде дуже м'якою, і не буде зосереджено на сексуальній поведінці, крім таких дій, як поцілунки чи обійми. Не буде сцен із небезпечними діями, які маленькі діти можуть копіювати, випробовуючи їх у реальному житті. Буде значний елемент вирішення проблем (наприклад, головоломки), який компенсує обмеження дій у грі. Прикладами таких ігор є Lego Indiana Jones (2008) та Bratz The Movie (2008);
- «PG» насильство може мати трохи більший вплив, але не буде жодних деталей у тому, як завдано травми чи поранення, і обстановка віддалена від реального світу яскравими елементами фантазії. Висловлювання є м'якими. Як і на «U», відсутні зображення небезпечних дій, які можна скопіювати після того, як їх показано у ігровому застосунку. Відсутність особливої уваги на реалістичній зброї(холода чи вогнепальна зброя). Приклади включають Kung Fu Panda (2008) і Harry Potter and the Order of the Phoenix (2007);
- ігри з рейтингом «12» підходять лише для дітей віком від 12 років, і протизаконно продавати або здавати їх в оренду дітям віком до 12 років. У «12» — помірне насильство, але без акценту на реалістичних деталях травм, ймовірно, і знову ж таки, фантастичні налаштування, які віддаляють дію від реального світу, будуть сильним помякшувальним фактором. Жорстка мова навряд чи буде прийнятною, оскільки природа відеоігор дозволяє нескінченне повторення такої мови. Можуть бути посилання на секс, які будуть знайомі більшості підлітків, і стримані зображення сексуальної

активності. Прикладом є Tomb Raider: Anniversary (2007) і Iron Man (2008);

- у відеоіграх «15» може бути зосереджено на більш серйозних проблемах, як це роблять фільми з рейтингом «15». Це може включати акцент на сильному насильстві з кривавими деталями, сцени жахів, нецензурну лексику, сексуальні сцени або згадки про наркотики. Приклади ігри Resident Evil 4 ("15", 2007) і Halo ("15", 2007);
- у «18» проблеми можуть бути ще сильніше і можуть включати акцент на сильному насильстві з реалістичними деталями травм, сильний жах, дуже різкі висловлювання, дуже сильні посилання на стать або навіть зображення сексуальної активності чи вживання наркотиків. Приклади включають серію Grand Theft Auto (‘18’, 1997–2008) і Unreal Tournament 3 (‘18’, 2007).

Ігровий жанр – це специфічна категорія ігор об’єднаних певними геймплейними особливостями. З часом кількість жанрів зростала, нові технологіїх та підвищення обчислювальних можливостей дозволяли вводити інновації та реалізовувати раніше складні ідеї. Ігрові жанри визначають які механіки та особливості будуть притаманні грі. Не є обов’язковим належність лише до одного жанру. Яскравим прикладом є MineCraft – поєднання компонентів із жанрів екшн, стратегії, пригоди та виживання в цій грі позначає MineCraft майже всіма ярликами жанрів, доступними для ігор, як-от наступний список із основних сайтів оглядів ігор: Пісочниця, виживання (Вікіпедія); Симулятор (Allgame, GameStop), Екшн (Mobygames); Пригодницький бойовик (Giantbomba); Пригоди (1UP, звичайні ЗМІ); Пригоди>Від першої особи>Фентезі (Gamefaqs); Дія від першої особи (IGN); фентезі (Metacritic), Симулятор міста (Neoseeker); Стратегія (PSN); Екшн і пригоди, стратегії та симулятори (Xbox Live); 3D, Пригоди, Від першої особи (Gamespot)[7].

Можна виділити наступні ігрові жанри[8]:

- екшин(від анг. Action – дія);
- екшн-пригодницький жанр;
- пригодницькі жанр;
- головоломки;
- рольовий жанр(RPG - role-played game);
- симулятор;
- стратегія;
- спортивний жанр;
- масова багатокористувацька онлайн-гра (так звана MMO та MMOG);
- казино або азартні ігри;
- казуальні ігри;
- цифрові колекційні карткові ігри;
- гра для печаті.

1.4 Аналіз існуючих програмних продуктів

Проаналізуємо відоме на сьогодні алгоритмічне забезпечення у данійобласті та технічні рішення, що допоможуть у реалізації комп'ютерної дитячої гри для розвитку друкування та моторики рук. Далі будуть розглянуті допоміжні програмні засоби, засоби розробки та готові програмні рішення.

1.4.1 Аналіз відомих алгоритмічних та технічних рішень

На основі аналізу існуючих ігрових продуктів для розвитку печаті та моторики рук було виділено основний алгоритм, який використовується, а саме валідація того, що ввів гравець та порівняння із очікуваним результатом. Цей алгорит є простим і не потребує доробки, тому доцільним є його використання.

Технічні рішення для головної механіки гри, а саме печаті, виділено наступні:

- гравцю потрібно ввести певні символи або набори символі за короткий проміжок часу, якщо він не встигне тоді до нього застосовується певний штраф, при накопиченні штрафів гравець програє;
- гравець повинен коректно вводити послідовності літер, але за помилковий ввід отримує штраф , при накопиченні штрафів гравець програє;
- гравцю потрібно корректно ввести символ або їх послідовність із обмеженням у часі, якщо гравець не встигає цього зробити або робить це не коректно до нього застосовується штраф, а згодом і програш.

Серед цих технічних рішень останнє є поєднанням перших двох, тому вирішено використовувати його, оскільки це рішення одночасно вимагає від гравця спритності та правильності вводу.

1.4.2 Аналіз допоміжних програмних засобів та засобів розробки

Для розробки ігрових застосунків невід'ємною частиною є ігровий рушій. На даний момент існує багато ігрових рушіїв, а саме:

- 4A Engine – ігровий рушій від українських розробників із компанії 4A Games, розрахований для шутерів, розробелий на C++;
- Unity – міжплатформенний 2D та 3D ігровий рушій для розробки різних ігрових жанрів, простий у вивченні та використанні, розробелий на C++, але скрипти на C#;
- Unreal Engine – це 3D графічний ігровий рушій, початково розроблений для шутерів, але згодом і для решти жанрів, розробелий на C++.

Оскільки Unity є 2D рушієм та розрахований для різноманітних ігрових жанрів, а також легкий у вивченні та використанні, було обрано його для розробки. Також скрипти, які пишуться на C# є простішими у вивченні ніж на C++, як у інших рушіїв.

1.4.3 Аналіз відомих програмних продуктів

Нині існує небагато ігрових продуктів механіки яких зав'язані на використанні вміння печаті та їх розвитку, в основному це веб-ігри.

Веб-гра "Keyboard Jump" за своїм ігровим процесом полягає у тому, що наш персонаж стрибає вгору на сходинки, і для вдалого стрибка потрібно повністю надрукувати слово, яке там написано. В ході друку слова ми можемо помилятися, за що гра карає, а саме віднімає здоровья гравця, при досягненні кількості здоров'я в 0 гра закінчується. Проте після помилково надрукованої літери її вже не потрібно натиска. За кожне правильне надруковане слово, а у деяких випадках випадкова послідовність символів, дається певна кількість балів. Гравець може обрати кількість очків здоров'я та які слова будуть використовуватись при грі. Гра націлена на акуратність та точність при печаті груп літер.

Веб-гра "Keyboard Ninja" потребую від гравця своєчасного натискання літери, яка знаходиться над "фруктом", який з'являється на невеликий проміжок часу на екрані. Якщо не встигнути натиснути літеру до зникнення "фрукту", користувачеві зараховується помилка, три помилки і гра завершується. Також можуть з'являтись "бомби" з літерами над ними, при натисканні цієї літери гра завершується. За кожну вірно натиснуту букву нараховуються бали, а за невірну літеру бали знімаються. Ціль гри розвинути швидкість та точність печаті літер.

Гра "Penguin Crossing" мета якої полягає в тому, щоб набрати літери на крижинах, перш ніж вони вріжуться в скелі. Як тільки пінгвін стрибне на крижину, якомога швидше необхідно ввести символи на крижині. Кожного

разу, коли правильно вводиться рядок символів, пінгвін стрибає на іншу крижину. Якщо зробити помилку, ви повинні виправити її, перш ніж продовжити. Якщо ви робите занадто багато помилок або друкуєте занадто повільно, гра закінчується. Допомагає розвинути вашу швидкість і точність.

З вище перечислених прикладів зрозуміло, що гра для розвитку навиків печаті, розрахованої на дитячу аудиторію, є актуальною.

Для порівняння розробки з дипломного проєкту “Let’s Type” з аналогами можна скористатись таблицею 1.1.

Таблиця 1.1 – Порівняння з аналогом

Функціонал	Let’s Type	Keyboard Jump	Keyboard Ninja	Penguin Crossing	Пояснення
Гра для печаті	Так	Так	Так	Так	Головною цілю гри є друкувати літери або слова заради перемоги
Різні рівні складності	Ні	Так	Ні	Так	Можна грати у більш складні версії гри та налаштовувати їх під свій рівень
Безкінечна	Так	Так	Так	Так	Чи може гра потенційно закінчитися
Збереження прогресу	Так	Ні	Ні	Ні	Зберігання стану гри для відновлення у майбутньому

Продовження таблиці 1.1

Поступове наростання складності	Так	Ні	Так	Ні	Чим більше граєш одну ігрову сесію тим складніше

1.5 Аналіз вимог до програмного забезпечення

Головною функцією програмного забезпечення є розвиток навичок друкування та моторики рук у ігровій формі, більше функцій можна побачити на Рисунку 1.1, де зображена схема структурна варіанти використання.

Рисунок 1.1 – Схема структурна варіантів використання

У наступних таблицях 1.2 – 1.11 наведено детальний опис варіантів використань:

Таблиця 1.2 – Варіант використання UC1

Параметр	UC1
Назва	Початок нової гри
Опис	Гравець має змогу розпочати гру із першого рівня та базовими характеристиками персонажа
Учасник	Користувач
Передумова	Відкрите стартове ігрове меню
Післяумова	Завантаження сцени першого ігрового рівня із базовими характеристиками персонажа

Таблиця 1.3 – Варіант використання UC2

Параметр	UC2
Назва	Продовження гри
Опис	Гравець має змогу продовжити гру із збереженого рівня та характеристиками персонажа
Учасник	Користувач
Передумова	Відкрите стартове ігрове меню та існують збереження гравця
Післяумова	Відкриття сцени ігрового рівня із параметрами заданими у збереженні

Таблиця 1.4 – Варіант використання UC3

Параметр	UC3
Назва	Вихід із гри
Опис	Гравець має змогу вийти із гри
Учасник	Користувач
Передумова	Відкрите стартове ігрове меню, або ігрове меню під час сцени ігровго рівня, або ігрове меню під час бойової сцени
Післяумова	Припинення роботи ігрового застосунку із збереженням останнього стану програми

Таблиця 1.5 – Варіант використання UC4

Параметр	UC4
Назва	Дослідження ігрового рівня
Опис	Гравець має змогу досліджувати ігровий рівень переміщуючись по ньому та взаємодіючи із різними об'єктами
Учасник	Користувач
Передумова	Завантажений ігровий рівень
Післяумова	Використані ігрові об'єкти

Таблиця 1.6 – Варіант використання UC5

Параметр	UC5
Назва	Викорситання ігрової валюти
Опис	Гравець має змогу використовувати ігрову валюту для покращення характеристик або переходу на наступний рівень
Учасник	Користувач
Передумова	Наявний магазин на активному рівні або табличка переходу на рівні
Післяумова	Збільшення характеристик гравця або перехід на наступний рівень в обмін на ігрову валюту

Таблиця 1.7 – Варіант використання UC6

Параметр	UC6
Назва	Відкриття ігрового меню
Опис	Гравець має змогу відкрити ігрове меню
Учасник	Користувач
Передумова	Завантажена сцена бою або рівня
Післяумова	Зупинка всіх ігрових процесів та відобрадення ігрового меню

Таблиця 1.8 – Варіант використання UC7

Параметр	UC7
Назва	Вступ у бій із противником
Опис	Гравець має змогу розпочати бій із противником
Учасник	Користувач
Передумова	Ігровий рівень із противниками на ній
Післяумова	Початок бойової сцени

Таблиця 1.9 – Варіант використання UC8

Параметр	UC8
Назва	Печать необхідних літер
Опис	Гравець має змогу вводити літери, які відображаються на об'єктах у ігровій сцені
Учасник	Користувач
Передумова	Завантажена бойова сцена із ворожими об'єктами
Післяумова	Знищення ворожого об'єкту на сцені

Таблиця 1.10 – Варіант використання UC9

Параметр	UC9
Назва	Отримання ігрової валюти за виграш
Опис	Гравець має змогу оттримати ігрову валюту, якщо виграє бойову сцену
Учасник	Користувач
Передумова	Завершення ігрової сцени із виграшним результатом
Післяумова	Повернення до ігрового рівня із винагородою у вигляді ігрової валюти

Таблиця 1.11 – Варіант використання UC10

Параметр	UC10
Назва	Перегляд діалогів
Опис	Гравець має змогу переглянути діалог та отримати винагороду за деякі із них

Продовження таблиці 1.11

Учасник	Користувач
Передумова	Ігровий рівень із об'єктом діалогу
Післяумова	Припинення діалогу та надання винагороди

1.5.1 Розроблення функціональних вимог

У якості вимог до комп'ютерної дитячої гри було обрано створення гри з противниками різної складності та бойовою системою, в основі якої розвиток навиків печаті та моторики рук. Функціональні вимоги до гри наведені у таблиці 1.11.

Таблиця 1.12 – Опис функціональних вимог

Ідентифікатор	Назва	Опис
FR-1	Початок нової гри	Система надає можливість розпочати нову гру із першого рівня
FR-2	Продовження гри	Система надає можливість продовжити вже існуючу гру із збереження
FR-3	Пересування по рівню	Гра надає можливість пересуватись у межах ігрової карти
FR-4	Вступ до діалогу	Система надає можливість вступати до діалогу із іншими персонажами
FR-5	Покращення характеристик	Гра надає можливість покращувати характеристики гравця, такі як кількість очок здоров’я та радіус атаки
FR-6	Отримання винагороди	Система надає можливість отримання винагороду за результатами бою із противником

Продовження таблиці 1.12

FR-7	Відкриття ігрового меню	Система надає можливість можливість зупинити гру у будь який момент відкривши ігрове меню та відновити нинішній стан, закривши його
FR-8	Захист персонажа за допомогою печаті	Система надає можливість під час бойової сцени захищати свого персонажа натискаючи літери, які знаходяться на ворожих об’єктах для їх знешкодження
FR-9	Перехід на наступні рівні	Система надає можливість переходити на наступні ігрові рівні за ігрову валюту
FR-10	Початок бойової сцени	Система надає можливість розпочинати бойову сцену із будь-яким противником на полі
FR-11	Перегляд характеристик гравця	Система надає можливість у будь-який момент переглядати актуальні характеристики гравця
FR-12	Припинення гри	Система надає можливість припинити гру у будь-який момент
FR-13	Участь у діалозі	Система надає можливість вступати до діалогу заради отримання інформації

Детальну відповідність між варіантами використання та функціональними вимогами наведено на Рисунку 1.2.

	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9	UC10
FR-1	■									
FR-2		■								
FR-3										
FR-4				■						
FR-5					■					
FR-6							■		■	
FR-7						■				
FR-8							■	■		
FR-9				■	■					
FR-10							■			
FR-11										
FR-12			■							
FR-13										■

Рисунок 1.2 – Матриця трасування

1.5.2 Розроблення нефункціональних вимог

Для ігрового щастосунку повинні бути наступні нефункціональні вимоги:

- коректне відображення ігрового рівня;
- коректне відображення характеристик гравця;
- збереження ігрового прогресу гравця при переході на наступний рівень чи виходу із гри;
- швидкий відгук на ввід користувача;
- обробка усього вводу користувача;
- зупинка усіх ігрових процесів при переході до меню;
- підвищення складності гри із кожним наступним рівнем;
- різні типи ворогів;
- залежність ціни від ігрового рівня та нинішніх характеристик гравця;

- відображення залишкової кількості слів, які ще потрібно надрукувати, у бойовій сцені;
- у бойовій сцені, при коректному вводі знищувати не більше одної літери.

1.6 Постановка задачі

Головною метою даної роботи є покращення методів розвитку друкування та моторики рук, за рахунок зацікавлення користувача, навчання у ігровій формі та поступового підвищення складності.

Для досягнення даної мети необхідно вирішити наступні задачі:

- розробити застосунок із простою для розширення архітектурою;
- розробити механіку дослідження ігрового світу та переходу між рівнями:
- розробити основні механіки гри, у яких користувачу потрібно використовувати друкування та спритність рук;
- розробка та наповнення сцен ігрових рівнів;
- розробити систему отримування та виконання завдань;
- створити механізм збереження прогресу гравця на його пристрої.

Висновки до розділу

У розділі 1 було описано загальні положення предметної області, висвітлено проблему та поставлено задачу для вирішення.

Було проаналізовано характеристики відеоігор, а саме:

- для яких платформ вони розробляються;
- види монетизації для ігор;
- типи розробників та способи випуску ігор;
- за кількістю гравців;
- вікові обмеження ігор.

На основі проаналізованих особливостей ігор, було зроблено висновки щодо характеристик дипломного проекту.

Розглянуто різноманітні ігрові жанри, на їх основі було вирішено, які особливості та механіки краще обрати та розвивати.

Було розглянуто кілька існуючих аналогів ігрових застосунків та порівняно із дипломним проектом. Виділено переваги та недоліки.

Після чого були розроблено схему структурну варіантів використання та на її основі описані варіанти використань у таблицях. Далі було описано функціональні вимоги до програмного застосунку. На основі варіантів використань та функціональних вимог було побудовано матрицю трасування.

Поставлені нефункціональні вимоги до застосунку щодо стійкості, обробки даних та прогресу гравця та стану гри.

На завершення була поставлена мета та задачі до програмного забезпечення.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Мова програмування C#

Створити комп'ютерну гру можливо на будь-якій мові програмування, як низького, так і високого рівня. Але від вибору мови програмування в першу чергу залежить кількість часу та ресурсів, які будуть витрачені на розробку, адже не для всіх мов є інструменти, які дозволяють швидко опанувати мову та створити застосунок.

Варто пам'ятати і те, шо для мов низького рівня потрібна висока кваліфікація, адже вони працюють із близькими для машинної мови сутностями, а саме байти, адреси, інструкції. З іншого боку мови високого рівня працюють із більш звичними для людини абстракціями: об'єктами, функції та змінні. Отже перевага для розуміння та зручності стоїть на боці мов високого рівня.

Створення ігрового застосунку для консольного клієнту та ігрового застосунку із графічним інтерфейсом, це дві абсолютно різні задачі. Для консольного застосунку достатньо вводити дані через аргументи, стрічку консолі чи зчитувати їх із файлів та після виводити результати роботи у файл чи вікно консолі. А для графічним інтерфейса потрібно підтримувати відчинене вікно у якому відображається ігрова ситуація, із даними, які обчислила програма у даний момент.

Для створення ігрового застосунку із графічним інтерфейсом потрібно працювати із великою кількістю змінних та одночасних процесів. Окрім цього, в залежності від складності гри, кількість процессів може кардинально збільшитись, і особисто створювати ігровий рушій для розробки гри дуже клопітлива та тривала робота. Тому варто використовувати мову із існуючими ігровими рушіями та бібліотеками для них.

Одною із найпопулярніших мов програмування, в тому числі і для створення ігрових застосунків для всіх платформ є C#.

C# – це мова програмування, розроблена для створення широкого спектру корпоративних програм, які працюють на платформі .NET Framework. Еволюція C# є простою, сучасною, безпечною для типів і об'єктно-орієнтованою. Код C# скомпільовано як керований код, що означає, що він отримує переваги від служб загальномовного середовища виконання. Ці послуги включають сумісність мов, збирання сміття, покращену безпеку та покращену підтримку керування версіями.

C# належить до сімейства мов програмування C. Це означає, що опановування C# полегшить вивчення C, C++ або Java, оскільки ці мови використовують подібний синтаксис і принципи програмування.

Щодо продуктивності мови, її можна виміряти за часом компіляції та фактичною продуктивністю програми. Порівняно зі своїм найближчим аналогом Java, C# має подібний час компіляції. Але тести продуктивності програми показують трохи швидші результати для C#. Але у порівнянні C++ та C#, виграє перша. Потрібно мати на увазі, що існує невелика різниця між тим, як компілюється код C++ і C#. Зазвичай C# проходить дві фази компіляції: перша – це коли код компілюється в проміжну мову на етапі збірки, а друга – це компіляція в машинний код, коли він виконується під час виконання. З огляду на тести, існує значна різниця між мовами, у яких C++ випереджає C#.

C# активно розробляється та підтримується Microsoft. За останні кілька років було випущено 3 основні випуски разом із .NET 5, які значно покращили досвід розробки.

Для побудови програми C# поділяються на дві різні категорії: програми командного рядка або консолі та програми Windows. Для обох категорій існують різні шаблони необхідного для окреслення проекту. Таким чином

можна легко розпочати створення повноцінної об'єктно-орієнтованої програми Windows.

Варто зазначити, використовуючи середовище виконання Mono, ті самі програми можна оптимізувати та перенести для систем на базі Unix та macOS. У багатьох випадках Mono показує результати продуктивності, які можна порівняти з .NET runtime. Це означає, що немає істотної різниці в тому, як програми C# працюють на різних платформах.

У C# наявні низькорівневі функції, такі як масиви вказівників (які забезпечують прямий доступ до пам’яті), також може бути недоліком з точки зору освноєння мови, але дуже корисно, якщо потрібна детальна робота із пам’ятю. Хоча C# є типово безпечною мовою, функції вказівників пропускаються під час перевірки виконання, що може стати причиною низької продуктивності або поломок програми.

2.2 Ігровий рушій Unity

Unity — це ігровий рушій, який використовується розробниками для створення різноманітних 2D або 3D кросплатформних ігор, та підтримує скрипти на мові програмування C#. Його використовували для розробки моделювання та відеоігор для консолей, комп’ютерів і мобільних пристроїв на початку свого існування. Unity має у своєму розпорядженні багато бібліотек та інструментів розробки. За допомогою цих інструментів можна виконати величезну кількість задач при чому різними способами.

Основою будь-якого проекта в Unity є сцена. Сцена - це файл, в якому розміщені окремі ігрові ситуації з певного набору об’єктів, налаштувань. Одночасно можна завантажити декілька сцен. На сцені може бути розташований об'єкт, до якого може бути прикріплена модель для відображення, або це може бути просто пустий об’єкт. До об’єкта можно прикріпити компонент, з якими взаємодіють скрипти.

Також є система наслідування, дочірні об’єкти повторюють усі можливі характеристики базового об’єкта. Також є локальна система координат.

Рушій підтримує безліч популярних форматів даних. Моделі, звуки, текстури, матеріали, скрипти можна запакувати у формат .unitypackage та ділитись із іншими розробниками або викладати у вільний доступ. Цей формат використовується у внутрішньому магазині Unity Asset Store, в якому розробники можуть безкоштовно і за гроші викладати в загальний доступ різні елементи, потрібні при створенні ігор. Щоб використовувати Unity Asset Store, необхідно мати обліковий запис розробника Unity.

На відміну від багатьох ігрових рушіїв, у Unity є дві основні переваги: наявність візуального середовища розробки та кросплатформеності.

Як недоліки можна виділити обмеження візуального редактора під час роботи з багатокомпонентними схемами, коли у складних сценах візуальна робота не ускладнюється. Другим недоліком є відсутність підтримки в Unity посилань на зовнішні бібліотеки, роботу з якими програмістам доводиться налаштовувати самостійно, і це ускладнює командну роботу. Ще один недолік пов'язаний із використанням шаблонів екземплярів (англ. prefabs). З одного боку, ця концепція Unity пропонує гнучкий підхід візуального редагування об'єктів, але з іншого боку, редагування таких шаблонів є складним. Також, WebGL-версія рушія, в силу специфіки своєї архітектури (трансляція коду з C# в C++ і далі JavaScript), має ряд невирішених проблем з продуктивністю, споживанням пам'яті і працездатністю на мобільних пристроях.

Вікно редактору сцени Unity 3D складається з невеликих окремих вікон, які можна переставляти, групувати в набори, від’єднувати від однієї позиції та знову прикріпляти назад у головному вікні. Така гнучкість надає змогу змінювати вікно редактору в залежності від потреб проекту, або звичок розробника. Серед окремих вікон головного вікна редактору можна виділити:

- вікно проекту, знаходиться зліва внизу Рисунку 2.1, показує бібліотеки ресурсів доступних для використання в тому числі файли

зображень, музичні файли, скрипти та інші файли проекту. У вікні проекту з’являються нові ресурси, після того як їх імпортують до проекту. До секції Favourites можна додати ресурси, які часто використовуються розробников;

- вікно перегляду сцени, знаходиться у центрі Рисунку 2.1, це дозволяє розробникам мати візуальну навігацію та можливість редагування ігрової сцени. Це вікно має можливість відображати як 2D, так і 3D-вид, залежно від типу проекту, над яким працює програміст працює. Також є можливість переміщувати об’єкти в цьому поданні, щоб розташувати їх для побудови сцени та створення певних ігрових ситуацій;
- ієрархічне вікно, зліва на Рисунку 2.1, показує ієрархічну структуру усіх об’єктів на сцені, тобто показує як об’єкти відносяться один до одного;
- вікно інспектора, зправа на Рисунку 2.1, дозволяє розробнику перевіряти та аналізувати всі скрипти об’єкта. Різні типи об’єктів мають різні набори скриптів, а скрипти різний набір параметрів, які можна модифікувати за допомогою цього вікна;
- вікно панелі інструментів, зверзу на Рисунку 2.1, найважливіше вікно, в першу чергу необхідне для запуску ігрової сцени. Зліва міститить основні інструменти для керування переглядом сцени разом із об’єктами, які містяться в ній. Ви також можете отримати доступ до хмарних служб Unity за допомогою кнопок праворуч, а також до облікового запису Unity, окрім цього до меню видимості та меню макета редактора, яке забезпечить альтернативний макет для вікон редактора.

Деякі об’єкти можуть багато разів зустрічатись у сцені або у кількох сценах. Для уникнення повторів у ігровому рушію Unity є можливість створювати префаби(від англ. Prefabs). Префаб – це об’єкт у якому визначені

всі компоненти, проініціалізовані змінні та розташовані всі необхідні об’єкти. Він виступає у ролі шаблона об’єкту. Зміна у префабі розповсюджується на всі сцени де він використовується.

2.3 Моделювання та аналіз програмного забезпечення

Для програмного забезпечення було виділено наступні основні бізнес-процеси:

- завантаження сцени ігрового рівня;
- проходження ігрового рівня;
- процес бойової сцени.

Завантаження ігрового рівня є основою для гри, адже на цій сцені знаходяться всі об’єкти із якими зможе взаємодіяти гравець. Окрім сцени ігрового рівня є присутні сцени стартового меню, завантажувача та бойова сцена. Проте завантаження саме сцени ігрового рівня є найбільш складним за умовами, та на відмінну від решти сцен ігровий рівень має багато різних реалізацій. На Рисунку 2.1 зображено бізнес-процес завантаження ігровго рівня.

Рисунок 2.1 – Бізнес-процес завантаження сцени ігрового рівня

Проходження ігрового рівня складний бізнес-процес, який має багато умов. Суть проходження полягає у переміщенні по рівню та взаємодії із об’єктами. На сцені ігрового рівня розташовано багато об’єктів в тому числі з деякими із них можна взаємодіяти. До прикладу об’єкти ворогів, діалогу,

магазин, межі ігрового світу. З усіма цими об’єктами можливо зіткнутись та розпочати взаємодію. Деякі із них відображають певну інформацію, та надають клавіші для вибору. А деякі розпочинають інші сцени. Не можливо взаємодіяти лише із об’єктами декору, у яких немає колайдера. Рисунок 2.2 відображає бізнес-процес проходження ігрового рівня.

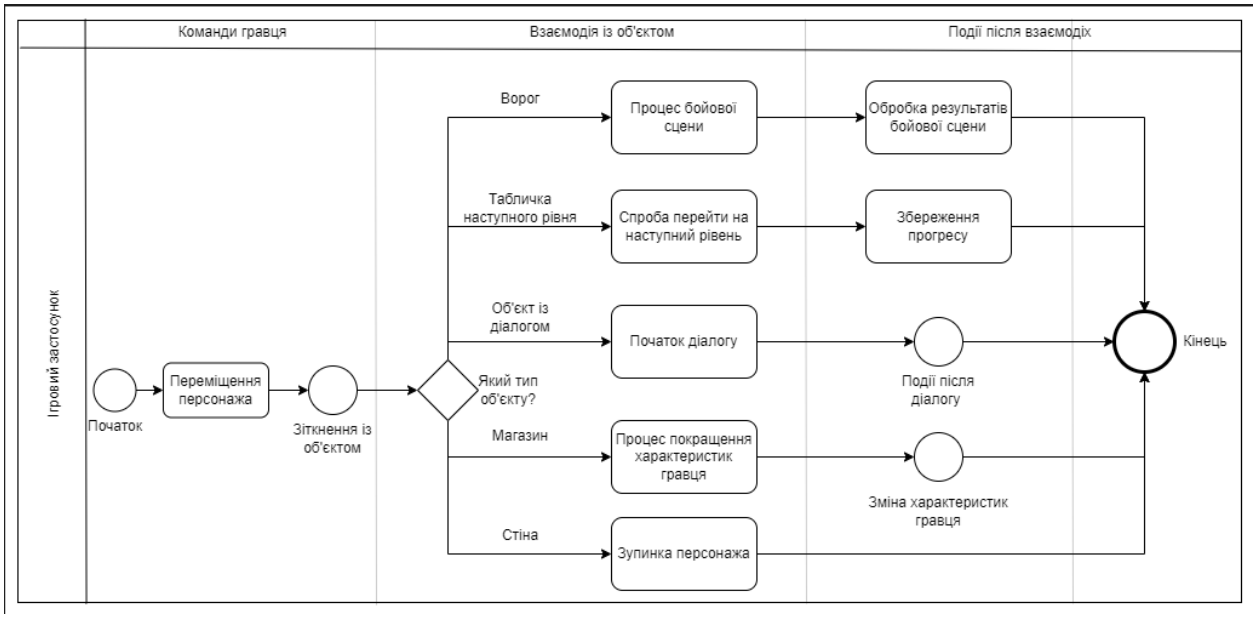


Рисунок 2.2 – Бізнес-процес проходження ігрового рівня

Бойова сцена має багато умов під час свого бізнес-процесу. Постійно іде перевірка на те чи є у гравця здоров’я та чи знаходяться ворожі об’єкти на бойовій сцені. В залежності від результатів процесу стан ігрового рівня та характеристики гравця можуть змінитись ьа повпливати на інші сцени. Вигляд бізнес-процесу бойової сцени показано на Рисунку 2.3.

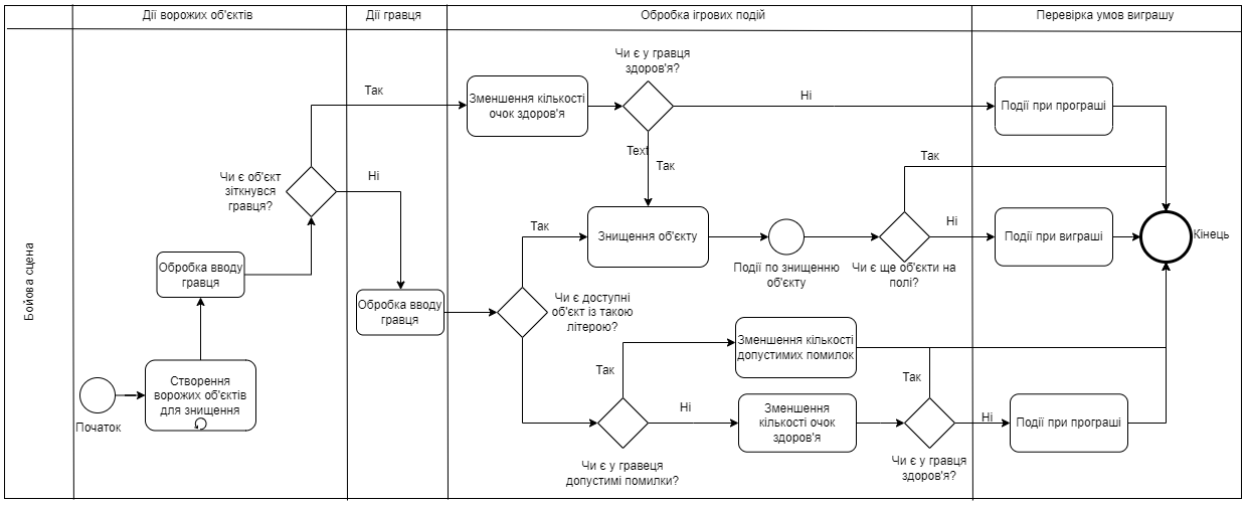


Рисунок 2.3 – Бізнес-процес бойової сцени

2.4 Архітектура програмного забезпечення

Враховуючи те, що ігровий застосунок не має зовнішніх компонентів та є повністю монолітним була обрана архітектурна модель MVC. Згідно цьому принципу додаток повинен бути розбитий на три основні логічні компоненти: Model , View та Controller. Компонент Model відповідає всій логіці, пов’язаної з даними, з якими працює користувач. Це може представляти дані, які передаються між компонентами View і Controller, або будь-які інші дані, пов’язані з бізнес-логікою. Компонент View використовується для всієї логіки інтерфейсу користувача програми. Контролери діють як інтерфейс між компонентами Model і View для обробки всієї бізнес-логіки та вхідних запитів, маніпулювання даними за допомогою компонента Model і взаємодії з Views для відтворення кінцевого результату.

Але у ігровому рушію Unity усе базується на скриптах. Ці скрипти виступають у ролі окремих компонентів та прикріплюються до об’єктів, для того щоб виконуватись при створенні об’єкту, при якійсь події, чи постійно з певним інтервалом часу. Для того щоб прикріпити скрипт до об’єкту, потрібно щоб він наслідувався від класу MonoBeahaviour, який певним чином виконує функції компонентів Model, View та Controller. Тобто MonoBeahaviour приймає дані на вхід, виконує певну логіку, оновлюється, та показує вивід.

Саме тому, щоб мінімізувати пересічення логіки створюється багато класів контролерів, які реагують на дії гравця та повідомляють іншим класам, що потрібно показати та як реагувати на дії гравця. На Рисунку 2.4 показано архітектуру програми із урахуванням MVC.

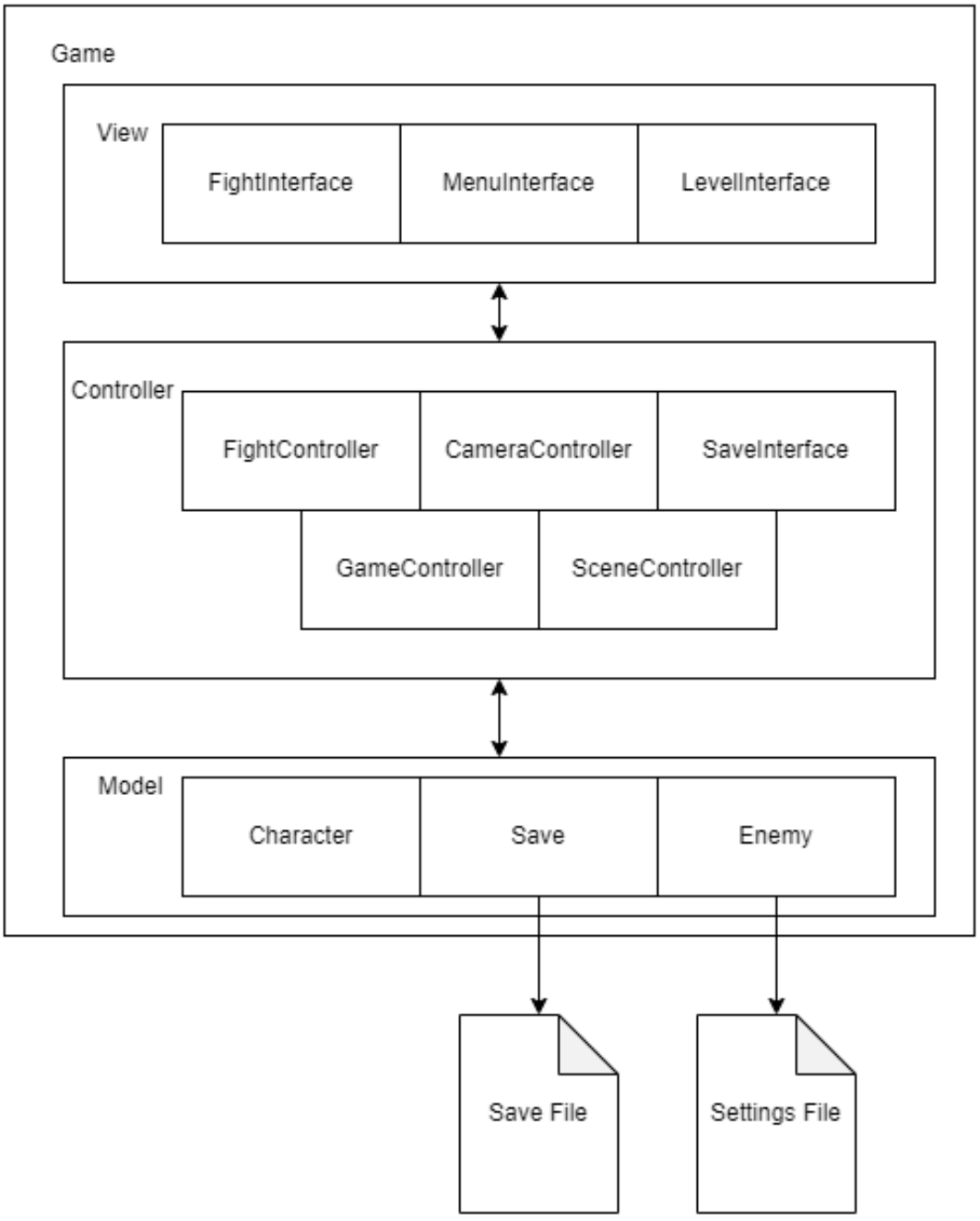


Рисунок 2.4 – Архітектура програмного забезпечення

Під час розробки програмного забезпечення виникає потреба у розширенні ігрової логіки, додавання нових структур та взаємодії із новими компонентами. Для полегшення розвитку та підтримки застосунку потрібно

розділити його на модулі, які залежать від абстракцій. Це є принципом інверсії залежностей (англ. dependency inversion principle, DIP) – один із ринципів SOLID.

Щодо структури відеогри варто відмітити, що було обрано варіант з розділенням ігрових механік. Ігровий світ поділяється на підлокації - рівні. Кожен рівень має свій рівень складності, своїх противників, об’єкти дизайну та свою ціну для переходу до наступного рівня, тобто рівні є незалежними один від одного. Таким чином кожен рівень може мати своє компоненти, які не пересікаються із іншими, та їх зміна не повпливає на інші рівні. Таке розділення на локації є зручним з точки зору економії ресурсів та мотивації гравця рухатись далі. Гравець не може перейти на наступний рівень доки не отримає достатню кількість монет, відповідно доки він не переможе певну кількість противників та не покращить свої вміння.

Для сутички із ворогом існує багато різних механік та способів їх реалізацій. При чому бойова механіка не обов’язково повинна бути центральною у грі, більша увага може приділятись сюжету, менеджменту ресурсів або уникненню конфліктів. Можна виділити активну та автоматичну сутички.

Активна сутичка, у якій гравець бере безпосередню участь. Гравецю потрібно реагувати на дії ворога заради перемоги. Частіше за все гравець обмежений у часі та можливих рішеннях. Завдяки обмеженням користувач повинен спостерігати та адаптуватись до ситуації, шукати оптимальне рішення. З архітектурної точки зору активна сутичка є більш складною, оскільки потребує багато класів, які взаємодіють та реагують на дії гравця, але вона була обрана, оскільки потрібна розвивати моторні навички гравця. Тому потрібен модуль, який відповідає за бойову механіку.

Бойова механіка базується на печаті слів у момент коли букви підсвічуються та у певному порядку. Для того щоб розділити бої з

противниками, було вирішено перенести їх на окрему сцену, де враховується нинішній стан персонажа гравця та противника з яким він змагається. Рішення із розділення бойових механік та локацій, які досліджує гравець, було прийнято оскільки бій є повністю незалежним від того що відбувається на рівні та бій відбувається один на один у стилі захисту від ворожих об'єктів. Ще не мало важливим фактором є те, що бойова сцена виконується у реальному часі і гравцю потрібно підлаштовуватись під ігровий стан, а в ігровій локації вороги є статичними об'єктами та гравець сам вирішує з ким розпочати протистояння. У випадку перемоги над противником, об'єкт зникає та гравець може переходити до наступного противника або до наступного рівня. Отже гра розділена на етапи і сам гравець вирішує який із них починається та коли, а сам етап не залежить від інших та може існувати окремо.

Для гри було виділені сутності забражені на Рисунку 2.5.

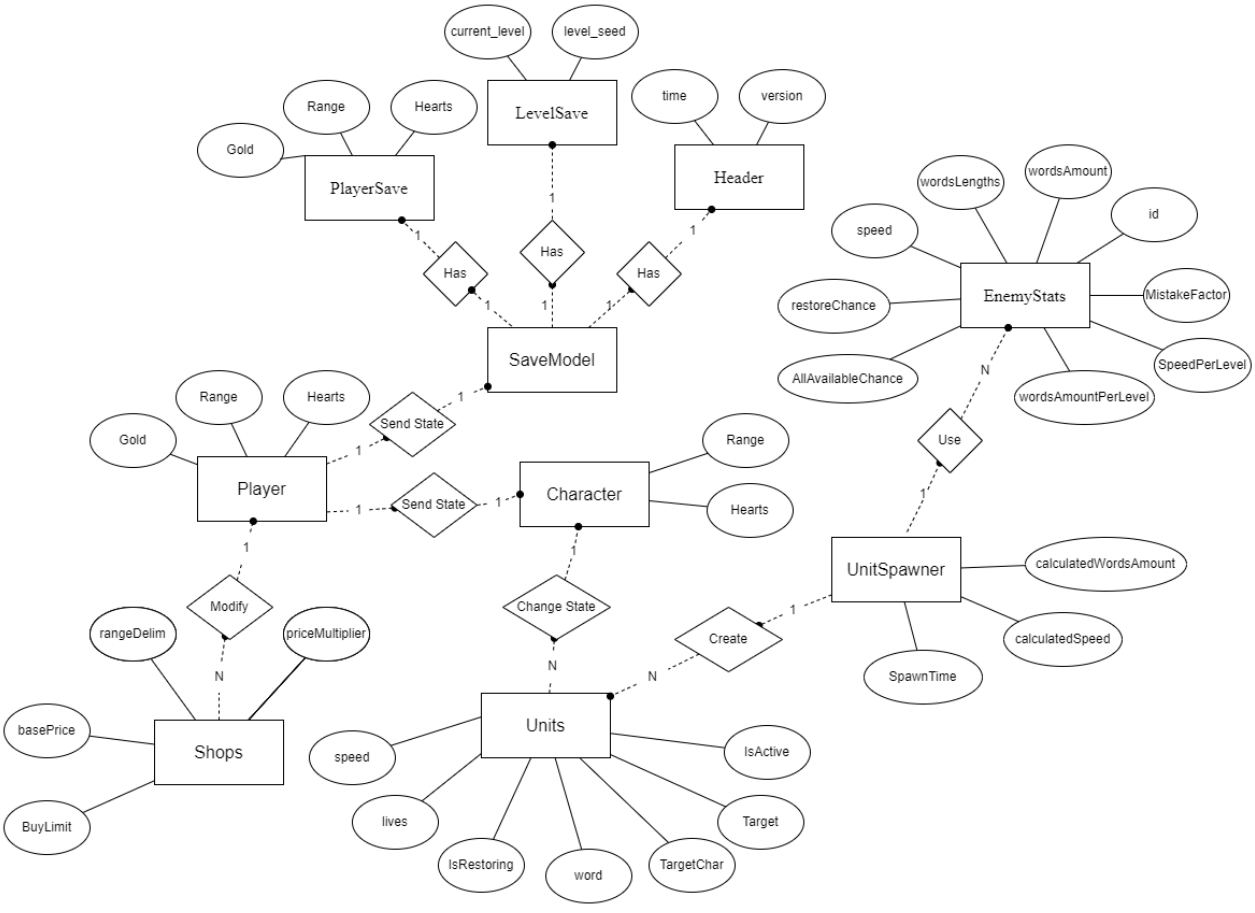


Рисунок 2.5 – ER діаграма сутностей

Сутності зображені на Рисунку 2.5 можна поділити на 3 групи:

- LevelSave, PlayerSace, Header, SaveModel – сутності для збереження стану гри та його відновлення;
- Shops та Player – сутності, які існують у межах ігрового рівня, при чому стан Player може модифікуватись та зберігатись між різними ігровими рівнями;
- Character, Units, UnitSpawner та EnemyStats – сутності бойової сцени, існують та оновлюються лише в її межах. Дані для ініціалізації отримують із ігровго рівня при старті бойової сцени.

2.5 Конструювання програмного забезпечення

Як було сказано у розділі 2.2 у ігровому рушію Unity усе базується на сценах. Для ігрового застосунку було виділено 4 основні види сцен:

- Loader – цей компонент завжди на сцені та відповідає за завантаження та видалення інших сцен, у цьому компоненті знаходяться контроллери камери та решти сцен, у тому числі і контроллер для роботи із збереженням та завантаження прогресу гравця;
- Menu - в цій сцені користувачеві надається вибір дій, можна продовжити уже розпочату гру, почати нову або вийти з застосунку;
- Level – у цій сцені відображається певна область із мобами, до якої дійшов гравець. На цій області є ігровий персонаж та вороже налаштовані моби, з якими гравець може розпочати бойову сцену. Також є різні декоративні об’єкти, перехід на наступний рівень та міні магазин;
- Fight – сцена у якій гравець повинен захищати свого персонажа від об’єктів які йдуть на нього вчасно натискаючи відповідні літери. Сцена припиняється у двох випадках: перемога гравця, після того як усі об’єкти знищено, або програш після того, як очки здоров’я гравця досягли відмітки в 0.

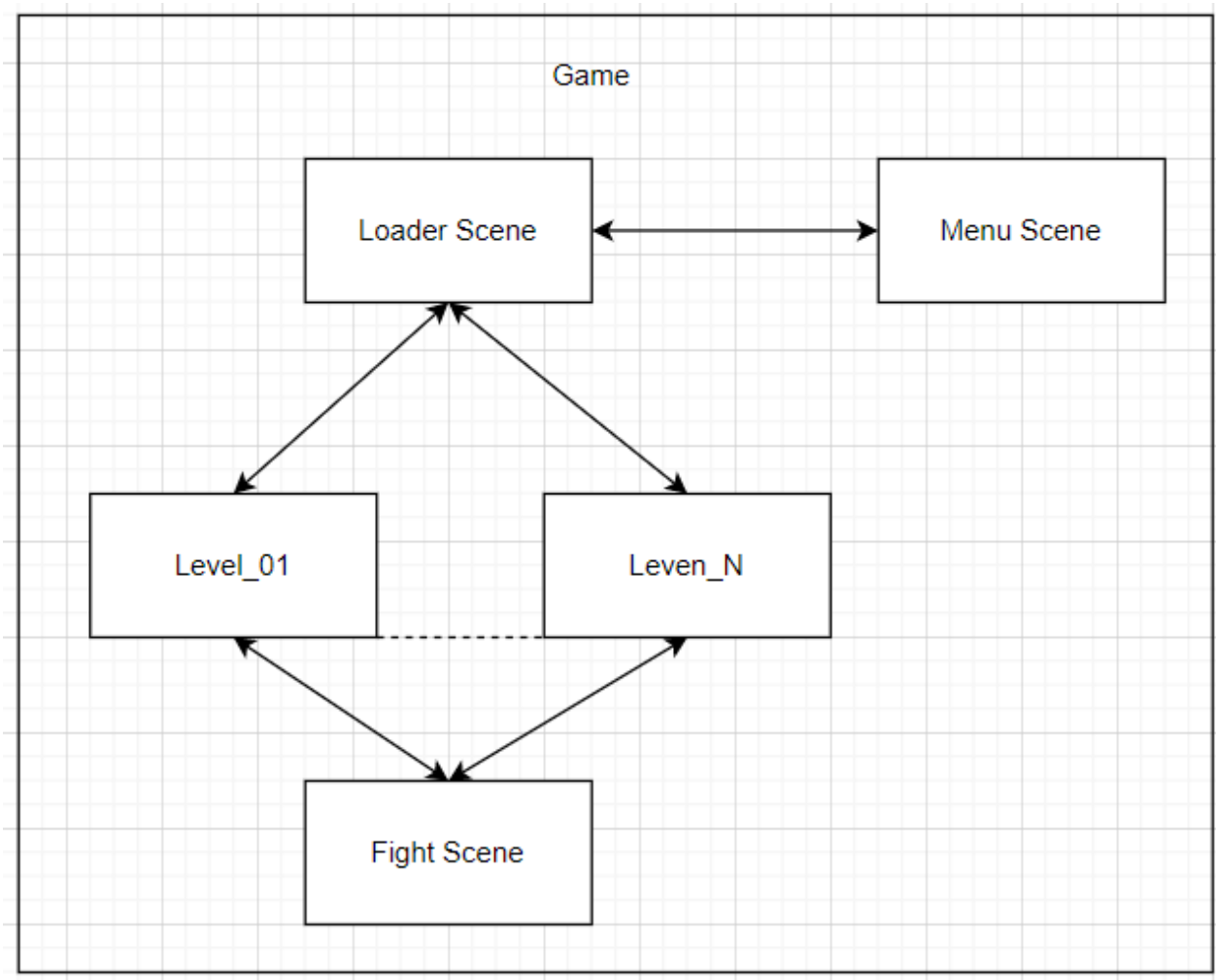


Рисунок 2.6 – Схема переходів із різних сцен

В роботі ігрового застосунку є багато процесів які потрібно контролювати. Через що виникає багато класів-контролерів, задача яких реагувати на дії гравця та запускати чи видаляти відповідні процеси. Можна виділити наступні компоненти:

- ігровий контролер, реагує на зміни ігрового стану, переходи між рівнями, меню, бойовими сценами. Також відслідковує зміни у стані гравця та зберігає або завантажує параметри гравця;
- контроллер сцен, відвантажує або завантажує необхідні сцени у нинішньому стані;
- контроллер меню, який реагує на дії гравця при виборі пунктів меню;
- контроллер рівнів, який завантажує наступний рівень у відповідь на дії гравця;

- контроллер камери, слідкує за тим на, що зараз дивиться ігрова камера, та слідкує за її перемиканням між меню, рівнями та бойовими сценами. У Unity є вбудований механізм, щоб камера слідкувала за гравцем;
- спостерігач, відслідковує зіткнення гравця із ворожими мобами, та розпочинає бойову сцену;
- контроллер бою, відслідковує умови для програшу чи перемоги гравця, та нарахування нагород;
- контроллер об'єкти-літер, створює та налаштовує об'єкти, які атакують гравця у бойовій сцені, відслідковує натискання літер та знищує об’єкти, якщо досягнуті умови;
- контроллер збереджень, зберігає або завантажує стан гри.

У деякиз випадках, для виконання завдання певного компонента необхідно створення декількох класів, для розширення можливостей.

Важливим компонентом є також префаб. Для персонажа, різних видів ворогів та елементів декору, щоб багаторазового використовувати об’єкти створено різні шаблони об’єктів - префаби. У одному префабі може використовуватись кілька класів.

Схема структурна класів винесена у графічний додаток із усіма класами. Після виділення основних компонентів, ігрового застосунок був створений із структурою класів наведеній у Таблиця 2.1. Опис класів наведено у таблиці 2.1. Опис вбудованих класів Unity, що використовувались для розробки, наведенйи у таблиці 2.2. Детальний опис методів класів наведено у таблиці 2.3.

Таблиця 2.1 – Опис класів

Клас	Опис
SceneController	Головний клас для завантаження та вивантаження ігрових сцен

Продовження таблиці 2.1

StartMenu	Клас, що описує поведінку клавіш стартового меню, а саме продовження гри, початок нової гри та вихід
GameMenu	Клас для зупинки та продовження гри, також для переходу між ігровими станами
Watcher	Клас, що прибирає глобальний інтерфес під час зміни сцени на бойову чи діалогу
ClearLocationWatcher	Клас, що розширяє логіку класу Watcher та додає відстеження подій пов’язаним із знищенням певних ворогів
PlayerParameters	Клас, що зберігає дані гравця у нинішньому ігровому стані для відображення користовачу
Player	Клас який дозволяє користувачеві керувати ігровим персонажем та регулювати його швидкість
NextLevelController	Клас для відображення та відслідковування умов переходу до наступного ігрового рівня
LevelStarter	Клас, що перенаправляє камеру на ігрового персонажа при створенні рівня
LevelStarterWithGenerator	Клас, що перенаправляє камеру на ігрового персонажа при створенні рівня та розпочинає створення не статичних об’єктів на рівні
FightController	Клас у якому відбувається підготовка та перехід до бойової сцени, а також обробка результатів
Shop	Клас, який реалізовує бізнес логіку для покращення ігрових характеристи персонажа в обмін на внутрішньо-ігрову валюту

Продовження таблиці 2.1

PlayerInterface	Клас оновлення даних інтерфейсу про характеристики гравця на карті рівня
Enemy	Базовий абстрактний клас для ворожого моба на карті рівня
EnemySpawner	Клас, у якому реалізована логіка заповнення випадковими ворогами ігрового рівня
BaseEnemy	Клас, що описує звичайного ворожого моба без ніяких модифікацій
ImmortalEnemy	Клас, що описує ворога, який не зникає після програшу
PlayerDefaultSettings	Клас, для зчитування, збереження та серіалізації початкових даних гравця
PlayerDefault	Клас, що зчитує та зберігає базові характеристики персонажа
EnemySettings	Клас, для зчитування, збереження та серіалізації характеристик ворогів
EnemyStats	Клас, що зчитує та зберігає характеристики конкретного ворога
SaveWorker	Клас, що зберігає та завантажує стани різних ігрових об’єктів, необхідних для збереження прогресу користувача
SaveModel	Базовий клас для збереження ігрової інформації про конкретний стан
Header	Клас, який містить метадані ігрової моделі, яка зберігається
SaveController	Клас, що контролює процес згортання даних для збереження та розгортання при завантаженні

Продовження таблиці 2.1

SaveComponent	Клас, який є каркасом для усіх класів які можуть зберігатись, вони повинні наслідуватись від нього
SavableComponent	Абстрактний клас необхідний для вимог серіалізації, десеріалізації та надання початкового стану від класу нащадка
PlayerSave	Клас, що описує дані для збереження персонажа
PlayerSavable	Клас, який серіалізує, десеріалізує та наданнє початковий стан даних персонажа
LevelSave	Клас для зберігання даних про нинішній рівень користувача
LevelSavable	Клас, який серіалізує, десеріалізує та наданнє початковий стан даних рівень користувача
PlayerProvider	Клас, який гарантує наявність одного екземпляру гравця та надає доступ до нього
GameState	Клас перечислення, для позначення станів гри
GameEvents	Клас, що зберігає та відслідковує ігрових подій для обміну даних між сценами
GameController	Клас, що контролює переходи між ігровими станами, в тому числі завантаженням та вивантажнням сцен
CameraController	Клас, для логіки контролю камери, а саме вімкнення та вимнення необхідних камер, переключення на щойно створені камери

Продовження таблиці 2.1

AppStarter	Клас, що реалізує логіку запуску GameController та SaveController у порядку, необхідному для коректної роботи
UnitSpawner	Клас, у якому реалізована логіка створення та контролю життєвого циклу об’єктів бойової сцени
UnitMovement	Клас для забезпечення пересування об’єкта у визначеному об’єктом напрямку
UnitModificator	Клас, що надає властивості об’єкту, в залежності від вказаних параметрів
Unit	Клас, що описує властивості та параметри об’єкту, який потрібно вчасно знищити під час бойової сцени
ServiceLocator	Клас, у якому реалізована логіка зберігання та надання об’єктів різних типів
RangeChecker	Клас, що реалізує логіку перевірки об’єктів на входження до зони доступу граця
InputProvider	Клас, який реалізує логіку відслідковування та валідації даних, введених користувачем, та відправки подій із даними вводу
FightWatcher	Клас, що реалізовує логіку ініціалізації та налаштування об’єктів бойової сцени
Character	Клас, що реалізує поведінку гравця, у відповід на ігрові події
FightInterface	Клас, що реалізовує логіку відслідковування та оновлення даних про стан гравця та бойової сцени

Продовження таблиці 2.1

WordsGenerator	Клас, що реалізовує логіку генерації слів для печаті під час бойової сцени
DialogTrigger	Клас, що реалізовує логіку перевірки умов для початку діалогу
Dialog	Клас, що описує дані діалогу та постувово відображає їх
DialogWithReward	Клас, що наслідується від класу Dialog та реалізує логіку тригреру події по завершенню відображання тексту

Таблиця 2.2 – Опис вбудованих класів Unity

Клас	Опис
Transform	Клас, що описує дані про розташування ігрового об’єкту, його масштаб
CinemacineVirtualCamera	Клас, що реалізовує логіку віртуальної камери на ігровій сцені
SpriteRenderer	Клас, що ркалізовує логіку рендеру спрайта і контролює його візуальний вигляд у сцені або 2D чи 3D проектах
Rigidbody2D	Клас, який реалізовує логіку передачі об’єкту під контроль фізичного рушія, але логіка обмежується пересуванням по осям абсцис та ординат
CircleCollider2D	Клас, який описує колайдер круглої форми із радіусом та положенням, таклж описує взаємодіє з системою 2D фізики
RectTransform	Клас, який реалізує логіку двовимірного аналогу класу Transform, але для елементів UI

Продовження таблиці 2.2

Canvas	Клас, що описує абстрактний простір для розміщення елементів інтерфейсу користувача
CanvasScaler	Клас, який реалізовує логіку керування загальним масштабом і щільністю пікселів елементів інтерфейсу у певному об’єкті із компонентом Canvas
GraphicRaycaster	Клас, який реалізує логіку трасування променів у компоненті Canvas
CanvasRenderer	Клас, що описує графічний об’єкт інтерфейсу користувача, що міститься в Canvas
TextMeshPro	Клас, що описує логіку відображення тексту, як елементу інтерфейсі користувача
Image	Клас, який реалізує відображення нестатичного зображення користувачеві
Button	Клас, що реалізує логіку виконання функції у відповідь на натискання кнопки користувачем
Tilemap	Клас, що описує систему, яка зберігає та обробляє ресурси для створення 2D-рівнів
Camera	Клас, що реалізовує логіку відображення ігрової сцени користувачу
AudioListener	Клас, що реалізовує логіку девайса, який захоплює звуки сцени
CinemachineBrain	Клас, який реалізовує бізнес логіку моніторингу усіх камер, розташованих на сцені

Таблиця 2.3 – Опис методів класів

Клас	Назва	Вхідні параметри	Опис
SceneController	LoadMenu		Завантажує сцену головного меню
SceneController	UnLoadMenu		Вивантажує сцену головного меню
SceneController	SwitchToFightScene		Завантажує бойову сцену
SceneController	UnloadFightScene		Вивантажує бойову сцену
SceneController	SwitchToLevelScene	int level	Завантажує сцену рівня за відповідним номером
SceneController	UnloadLevelScene	int level	Вивантажує сцену рівня за відповідним номером
SceneController	LoadScene	string sceneName	Завантажує сцену за назвою
SceneController	UnloadScene	string sceneName	Вивантажує сцену за назвою
SaveWorker	Save	SaveModel saveModel	Приймає ігрову модель та зберігає її до файлу збереження у текстовому форматі
SaveWorker	Load		Завантажує дані із файлу збереження та ініціалізує модель даних
SaveWorker	Prepare		Перевіряє файла наявність збережень та повертає результат

Продовження таблиці 2.3

SaveController	LoadSave		Отримує дані збережень та розподіляє дані між об’єктами
SaveController	SaveState		Формує модель із даних ігрового стану для подальшого збереження
SaveController	PrepareSave	bool delete = false	За умовами підготовлює дефолтний стан для збережння
PlayerSavable	PrepareInitial		Створює початковий стан параметрів гравця
PlayerSavable	Serialize		Серіалізує дані гравця у текстовий формат
PlayerSavable	Deserialize	PlayerSave save	Десеріалізує збережені дані гравця
LevelSavable	PrepareInitial		Створює початковий стан даних про рівень
LevelSavable	Serialize		Серіалізує дані про рівень у текстовий формат
LevelSavable	Deserialize	PlayerSave save	Десеріалізує збережені дані про рівень
StartMenu	ContinueGame		Запускає ігрову сцену із збереження
StartMenu	StartGame		Запускає сцену рівня із ігрового меню
StartMenu	StartNewGame		Видаляє збереження та запускає гру із дефолтного стану

Продовження таблиці 2.3

StartMenu	Exit		Закриває ігровий застосунок
GameMenu	Awake		Ініціалізує стан об’єкту
GameMenu	FixedUpdate		Перевіряє умови для активації ігрового об’єкту із відображенням функцій меню
GameMenu	Resume		Деактивує об’єкт меню та продовжує ігровий процес
GameMenu	Pause		Активує об’єкт меню та зупиняє ігровий процес
GameMenu	ExitToMenu		Ралізує логіку виходу із активної сцени
GameMenu	Exit		Припиняє роботу застосунку
Watcher	FixedUpdate		Перевіряє умови для відобрадення глобального інтерфейсу рівня
ClearLocationWatcher	Awake		Ініціалізує свій стан та підписується на подію перемоги над ворогом
ClearLocationWatcher	DoOtherStuff	Enemy enemy	У разів виконання умов завдання змінює діалог завдання на діалог для вингороди
ClearLocationWatcher	IsMissionComplete		Повертає результат перевірки на виконання умов завдання
Player	Awake		Ініціалізує змінні об’єкту

Продовження таблиці 2.3

Player	Update		Оновлює напрямок руху в залежності від даних введених користувачем
Player	FixedUpdate		У разі виконання умов, рухає персонажа
Player	Move		Переміщує персонажа у відповідності з наданим напрямком та швидкістю
Player	OnTriggerEnter2D	Collider2D other	У разі зітнення із інщим об’єктом, перевіряє його тип, та подає сигнал про початок бою
NextLevelController	FixedUpdate		Оновлює значення ціни рівня на елементі інтерфейсу
NextLevelController	OnTriggerEnter2D	Collider2D other	У випадку зіткнення із гравцем, робить активним клавішу для переходу до наступного рівня
NextLevelController	OnTriggerExit2D	Collider2D other	Під час виходу гравця із колайдера об’єкту, деактивує плашку із переходом на наступний рівень
NextLevelController	GoToNextLevel		Реалізує перехід на наступнйи рівень за певну плату
LevelStarter	Awake		Фіксує камеру на гравцеві та робить її активною

Продовження таблиці 2.3

LevelStarterWithGenerator	Awake		Фіксує камеру на гравцеві та робить її активною, після чого подає команду на генерацію об’єктів
EnemySpawner	Awake		Ініціалізує генератор випадкових чисел
EnemySpawner	SpawnEnemies		Розпочинає процес створення та розстановки ігрових інтерактивних об’єктів
EnemySpawner	GetRandomVector3		Створює та надає випадкову точку у певній зоні
EnemySpawner	CheckForCollision	GameObject unit	Перевіряє чи перетинається об’єкт із будь-яким об’єктом на сцені
FightController	PrepareFight	Enemy enemy	Запам’ятовує дані ворога та починає слідкувати за результатами поєдинку
FightController	GetEnemyName		Надає назву ворога, із яким розпочато бій
FightController	FightResult	FightResults res	Оброблює результати бою, виконує після бойові операції
Shop	Awake		Ініціалізує об’єкт та робить його неактивним
Shop	Update		Оновлює відображення ціни на товар

Продовження таблиці 2.3

Shop	OnTriggerEnter2D	Collider2D other	Коли гравець перетинає колайдер, активує Canvas із відображенням цін
Shop	OnTriggerExit2D	Collider2D other	Коли гравець виходить із колайдеру, деактивує Canvas із відображенням цін
Shop	BuyRange		Збільшує радіус ураження ворогів в обмін на плату
Shop	BuyHeart		Збільшує кількість здоров’я в обмін на плату
Shop	CalculatePrice	int value	Рахує ціну на покращення зарактеристик
PlayerInterface	FixedUpdate		Оновлює інформацію про дані гравця на елементах інтерфейсу
Enemy	DestroySelf		Знищує об’єкт у якого є цей компонент
ImmortalEnemy	DestroySelf		Не знищує об’єкт
GameEvents	StartFightEndEvent	FightResults res	Сповіщає усі об’єкти, що підписані на івент, про результати бою
GameController	GetState		Надає нинішній ігровий стан
GameController	LoadMenu		Подає команду на завантаження сцени головного меню та виставляє відповідний ігровий стан

Продовження таблиці 2.3

GameController	StartFight	Enemy enemy	Подає команду на завантаження та підготовку бойової сцени, підготовку виставляє відповідний ігровий стан
GameController	StartGame		Подає команду на вивантаження головного меню та завантаження збереженого рівня
GameController	StartNextLevel		Подає команду на вивантаження нинішнього рівня, завантаження наступного рівня, збереження ігрового процесу
GameController	Exit		В залежності від нинішнього ігрового стану вивантажує сцену із стеку
GameController	ExitFight		Виконує вихід із бойової сцени та активує камеру для ігрового рівня
GameController	IsDialog	bool inDialog	В залежності від аргумента inDialog виставляє відповідний стан гри
CameraController	Start		Викликає метод для деактивації камери, яку контролює

Продовження таблиці 2.3

CameraController	FocusWorldCamera	Transform transform	Фокусує підконтрольну камеру на компоненті transform
CameraController	EnableWorldCamera		Активує підконтрольну камеру
CameraController	DisableWorldCamera		Деактивує підконтрольну камеру
AppStarter	Awake		Є відправною точкою застосунку, надає команду про завантаження збережень та запускає методи для завантаження меню
AppStarter	PrepareSaves		Підготовлює збереження під час старту
UnitSpawner	SpawnRoutine		Розпочинає процес формування слів, створення об'єктів Unit із буквами, їх ініціалізація та збереження до списку об’єктів
UnitSpawner	OnUnitDeath	Unit unit	Обробляє події знищення об’єкту класу Unit, а саме очищує ресурси та перевіряє умови перемоги
UnitSpawner	GetWord		За допомогою словника, отримує слово певної довжини, в залежності від змінних об’єкту

Продовження таблиці 2.3

UnitSpawner	GetRandomPosition		Створює випадкову позицію неподалік від ігрового персонажа у певній зоні
UnitSpawner	GetWordsLeft		Надає кількість слів, які необхідно ще створити, у вигляді об’єктів класу Unit
UnitSpawner	IsFinishSpawning		Перевіряє умови для припинення процесу формування слів
UnitSpawner	Init	EnemyStats stats, int level	Ініціалізує внутрішні змінні та об’єкти в залежності від параметрів, переданих у об’єкті EnemyStats
UnitSpawner	StartSpawn		Розпочинає відкладену функцію для процесу формування слів
UnitSpawner	OnInput	char inputChar	Реалізує алгоритм перевірки наявності літери, яку ввів користувач, серед слів із залишком літер, обробляє сиуацію при якій літера введена не коректно
UnitSpawner	MakeAllAvailable		Переводить усі об’єкти-літери Unit у стан, за якого можливо їх знищити

Продовження таблиці 2.3

UnitSpawner	GetCharactersLeft		Надає кількість спроб, за яких користувач може щеввести літеру, якої немає в доступних
UnitSpawner	OnDestroy		У разі преривання роботи генератора, відправляє подію із результатом програшу
UnitMovement	Update		Виконує пересування об’єкту у відповідність до його характеристик
UnitModificator	Init	EnemyStats settings, int level	Ініціалізує внутрішні змінні об’єкту в залежності від EnemyStats
UnitModificator	ModifyUnit	Unit unit	В залежності від внутрішніх змінних, надає об’єкту Unit одну властивість
Unit	MakeAvailable		Робить цей об’єкт доступним для знищення
Unit	MakeAllAvailable		Позначає цей об’єкт жовтим кольором, при його знищенні гравцем він робить решту об’єктів доступними для знищення
Unit	MakeRestore		Позначає цей об’єкт червоним кольором, при його знищенні гравцем він відновлює гравцеві одне очко здоров’я

Продовження таблиці 2.3

Unit	OnInput	char inputChar	Приймає літеру та якщо вона відповідає літері на цьому об’єкті, знищує цей об’єкт
Unit	Die		Надсилає подію про знищення цього об’єкту, після чого знищує об’єкт
Unit	GetTarget		Надає точку, до якої прямує цей об’єкт
RangeChecker	ChangeState	Collider2D other	При зіткненні помічає усі об’єкти-літери, як доступні для знищення
RangeChecker	Init		Ініціалізує розміри цього об’єкту
InputProvider	Update		Приймає символи, що ввів користувач та валідує їх, після проходження валідації формує подію із значенням літери
FightWatcher	Awake		Завантажує дані про існуючих ворогів та розпочинає бій із необхідним ворогом
FightWatcher	StartFight	string enemyId	На основі даних про ворога ініціалізує об’єкти для початку бою
Character	Init		Встановлює кількість здоров’я персонажа

Продовження таблиці 2.3

Character	TakeAHit		Зменшує здоров’я персонажа, за умов завершує бій із програним результатом
Character	TryToDestroy	Collider2D other	У випадку зіткнення із об’єктом-літерою реалізує логіку отримання шкоди здоров’ю та знищення іншого об’єкту
Character	Heal	int heal = 1	Відновлює вказану кількість очок здоров’я
FightInterface	Start		Ініціалізує поля інтерфейсу для виведення інформації про стан гри
FightInterface	FixedUpdate		Оновлює на елементах інтерфейсу дані про стан гри
WordsGenerator	WordsGenerator		Завантажує наявні слова із ресурсів гри
WordsGenerator	GetWord	int length	Надає випадкове слово відповідної довжини
DialogTrigger	OnTriggerEnter2D	Collider2D other	Розпочинає діалог із гравцем та переводить гру у стан діалога
DialogTrigger	OnTriggerExit2D	Collider2D other	Повідомляє про припинення діалогу та необхідність відновити відображення інтерфейсу

Продовження таблиці 2.3

Dialog	Start		Деактивує елементи інтерфейсу із відображенням діалога
Dialog	Update		Поступово виводить текст, або відразу при натисканні будь-якої клавіші користувачем
Dialog	StartDialog		Очищає елемент інтерфейсу від тексту та інціалізує початок виводу тексту діалогу
Dialog	TypeLine		Поступово виводить літери із стрічки діалогу
Dialog	NextLine		Переходить до виводу наступної стрічки діалогу, якщо стрічки закінчились то завершує діалог
DialogWithReward	PostDialog		Одноразово надає винагороду грацю

У наступних розділах детальніше описано створення ігрових сцен із допомогою поєднання інструменів ігрового рушія та класів створених власноруч.

2.5.1 Розробка завантажувача гри

Будь-яка гра розпочинається із розгортання сцени та об’єктів на ній, тому створюємо сцену з назвою Loader. У ігровому рушію Unity усі об’єкти створюються одночасно, тому відправна точка, з якої стартує застосунок не існує. Тобто якщо розмістити на сцені об’єкти, які потребують поступового

запуску та залежать від даних інших об’єктів, то їх старт може бути некоректним через те що, до прикладу дані гравця ще не завантажені, але інший об’єкт вже їх потребує.

Тому стровено класс AppStarter, лише у ньому при створенні об’єкту виконується функція Awake, яка стартує разом з створенням об’єкту. AppStarter підготовлює застосунок перед завантаженням сцени меню, до прикладу завантажує збереження. Після чого AppStarter з допомогою скрипта SceneController завантажує сцену меню.

Loader має у своєму розпорядженні різні Singleton класи для зручного доступу з будь якого місця програми та відсутності необхідності передавати об’єкти цього класу в класи, яким вони потрібні. Такими класами є CameraController, GameController, PlayerProvider, FightController.

У сцені Loader існує об’єкт камери, для того, щоб показувати гравцю, що відбувається на сценах. Є дві камери, перша яка показує сцену завантаження та меню, друга, яка вмикається коли гравець переходить до гри, та ця камера слідкує за персонажем гравця. Для контролю камери існує класс CameraController. При початку бойової сцени камера змінюється на ту, яка знаходиться в бойовій сцені для спостереження за боєм.

GameController має у своєму розпорядженні клас SceneController призначений для завантаження та розвантаження сцен. GameController знає про те в якому стані знаходиться застосунок та контролює в який стан перейде відеогра. До прикладу за допомогою SceneController буде завантажено меню, рівень або бойова сцена. Також контролюється вихід із різних ігрових станів.

PlayerProvider відповідає за доступ до параметрів гравця з будь якої точки виконання програми.

FightController управляє завантаженням бойової сцени, розгортанням об’єктів та їх налаштуванням. Також він відстежує події, які відбуваються під час бою для визначення переміг гравець чи програв.

2.5.2 Розробка ігрових меню

Взаємодія гравця з грою розпочинається з головного меню. Зазвичай у головному меню існує багато дій, таких як завантаження збереженого стану гри, початок нової гри, зміна налаштувань гри, за наявності вибір різних режимів гри(одиночна, онлайн, пісочниця) та вихід. Варто зазначити про використання об’єкту Canvas, який використовується для інтерфейсу.

Для створення ігрового меню створюємо нову сцену з відповідною назвою Menu. У цій сцені є три кнопки, з допомогою яких гравець розпочати нову гру, продовжувати уже існуючу гру та вийти з гри. Також є задній фон який є об’єктом типу Image. До кожної кнопки типу Button при натисканні можна додати функцію, в даному випадку використовуються функції класу StartMenu, для початку нової гри до кнопки прив'язується функція StartNewGame, для продовження - ContinueGame та для виходу - Exit, яка зупиняє роботу застосунку з допомогою функції Application.Quit(). Скрипт StartMenu знаходиться в об’єкті Menu. Функції цього скрипта прикріпленого до об’єкту Menu уже використовуються для кнопок.

При переході до ігрових локацій сцена з меню GameController розвантажує сцену та завантажує рівень та виставляє ігровий стан.

За потреби гравець може з допомогою клавіші escape увімкнути ігрове меню(Рисунок 2.7). Під час увімкнення ігрового меню ігровий стан перестає нараховуватись з допомогою виставлення наступного параметру Time.timeScale = 0f. Гравець з допомогою GameController, який знає нинішній стан гри, може вивантажити сцену бою або рівня. Для цього існує функція GameController.Exit(), яка нічого не повертає нічого та в залежності від стану переходить до попереднього. Тобто якщо гравець знаходиться в бойовій сцені при спробі перейти в меню бій буде зупинено та він повернеться до ігрового рівня.

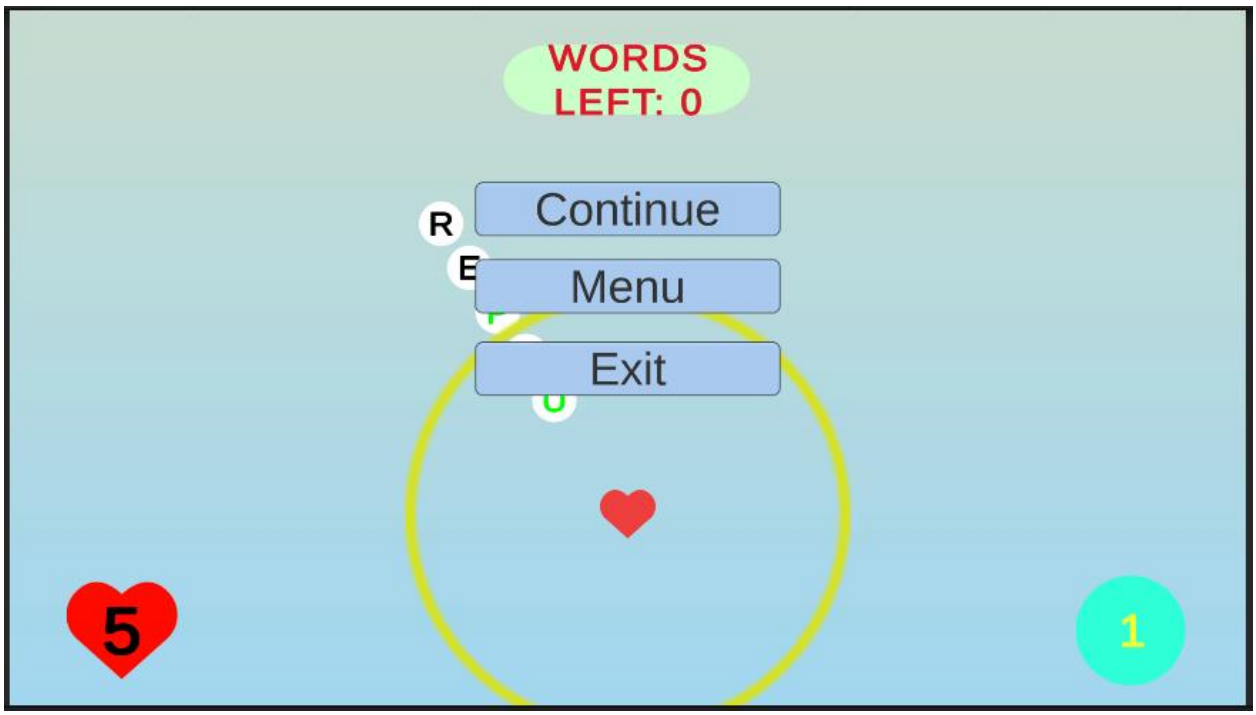


Рисунок 2.7 – Меню під час бойової сцени

2.5.3 Розробка ігрових рівнів

Для створення ігрового рівня потрібно створити сцену із назвою, яка розпочинається на Level_ та закінчується на номер рівня. Варто зазначити, що відеогра розробляється у 2D стилі тому для фона використовуэться Tilemap.

Далі на сцені розміщується об’єкт, яким буде керувати гравець - Player. Для керування об’єктом створено скрипт Player, у якого є параметри швидкості та прискорення. При натисканні клавіш напрямку, цей скрипт реагує на напрямок руху та у разі натискання клавіші shift прискорює об’єкт. На об’єкті гравця також є скрипти SpriteRenderer, Rigidbody2D, BoxCollider2D для взаємодії з іншими об'єктами.

На сцені можуть розміщуватись об’єкти декору та невидимі стіни для того, щоб гравець не виходив за межі карти. Усі ці об’єкти теж мають крипти SpriteRenderer, Rigidbody2D, BoxCollider2D. Завдяки перечисленим скриптам гравець не може проходити крізь ці об’єкти.

Наступні об’єкти, які потрібні для стандартного функціонування гри, це вороже налаштовані ігрові агенти. Від об’єкту Player вони відрізняються лише

тим, що замість скрипта Player у агента є скрипт BaseEnemy. У скрипті BaseEnemy зберігається назва ворога та нагорода за перемогу над ним. Коли гравець натикається на ворога розпочинається бойова сцена.

На сцені присутній Canvas, у якому розташовані об’єкти із параметрами гравця, а саме кількістю здоров’я, монет, радіуса атаки. Ці дані бере та оновлює скрипт PlayerInterface, який знаходиться у об’єкті Interface. При початку бойової сцени об’єкт Interface стає неактивним за допомогою функції SetActive.

Також, для переходу на наступний рівень створено скрипт NextLevelController, який має у своєму розпорядженні Canvas із текстом ціни переходу на наступний рівень та кнопкою для покупки цього переходу. Об’єкт при наближенні гравця повинен реагувати та активовувати Canvas.

На сцені рівня є об’єкти, при наближенні до яких активується діалог, це необхідно для інтеракції із гравцем за допомогою текстових повідомлень. З допомогою таких повідомлень комп’ютерна гра повідомляє та навчає гравця певним особливостям гри.

У деяких діалогах гравець може розпізнати завдання. Для виконання яких потрібно перемогти певний тип ворогів на рівні, після чого знову поговорити із мобом, який повідомив користувача про завдання. За виконання завдання гравець одноразово винагороджується.

2.5.4 Розробка бойової сцени

Бойова сцена є однією із основних сцен, стан гри змінюється у реальному часі, що вимагає від гравця дій в залежності від ситуації.

Бойова сцена заснована на активному використанні навиків печаті. Відповідно для цієї сцени потрібна обробка та валідація клавіш, які натискає гравець. Для цієї задачі створено класс InputProvider, який з допомогою функції Input.inputString отримує дані, що ввів користувач. Після отримання даних, які отримуються у вигляді послідовності символів, кожен символ

перевіряється та у разі проходження валідації відправляється івент з цим символом.

Наступним компонентом для розробки є об’єкт гравця, на який прямують об’єкти з літерами. Цей об’єкт названий Character та має стандартний набір компонентів для відображення та взаємодії з іншими об’єктами. Однак ще додається скрипт з однойменною назвою Character, призначений для відслідковування зіткнення цього об’єкта із об’єктами-літерами. Для цього є базова функція OnTriggerEnter2D, де об’єкти із компонентом колайдера можуть перетинатись та мати повну інформацію про об’єкт із яким зіткнувся.

Для об’єкту з літерою, від яких потрібно захищати об’єкт Character, має такі ж характеристики та носить назву Unit. Unit має відмінність лише у тому, що скрипт Character замінюється на скрипт Unit. Цей скрипт має у своєму розпорядженні літеру, яку необхідно натиснути для знищення цього об’єкту. Також є змінна, яка визначає чи може цей об’єкт бути знищеним, для того щоб літеру можна було натиснути лише в певний момент. Для відображення літери використовується Canvas, у якому є текстове поле.

Для підрахунку кількості слів, створення об’єктів Unit з відповідними літерами, переліку невірно натиснутих клавіш створено окремий скрипт UnitSpawner. Цей скрипт при старті отримує дані про противника, а саме кількість слів, довжини слів, швидкість руху агентів. Після обробки даних запускається функція, яка розпочинає створення груп об’єктів, які утворюють слово. При чому UnitSpawner зберігає в собі списки із усіма об’єктами, які створив, для подальшого їх знищення. UnitSpawner відслідковує івенти, які надсилає клас InputProvider, та в залежності від того вірно натиснута літера, чи ні знищує об’єкт із відповідною літерою, або декреметує кількість літер, які може натиснути гравець.

RangeChecker ще один об’єкт для якого розроблено однойменний скрипт. Його створено для того, щоб активувати об’єкти типу Unit, коли вони пересікають колайдер об’єкту RangeChecker.

Для бойової сцени існує окремий інтерфейс, для нього виділено окремий батьківський об’єкт, який у своєму розпорядженні має три дочірнії об’єкти Canvas для відображення кількості очок здоров’я гравця, кількість слів, які залишились та кількість літер, які невірно може натиснути гравець. З допомогою функції FixedUpdate, елементи текстові елементи інтерфейсу постійно оновлюються.

2.6 Аналіз безпеки даних

Ігровий застосунок виконує роботу із великим об’ємом різноманітних даних та із різними налаштуваннями стосовно лише однієї ігрової сесії. Розпочинаючи від даних об’єктів про розміщення у системі координат та завершуючи даними на елементах інтерфейсу. При такому об’ємі даних варто чітко виокремити, які дані потрібно зберігати та як саме зберігати ці дані.

Під час використання ігрового застовсунку, користувач накопичує первний прогрес. Цей прогрес виражається у номері рівня, на якому знаходиться користувач, кількість ігрової валюти, кількість очок здоров’я та розмір радіусу впливу. Ці данні однозначно необхідні між запусками гри та є обов’язковими для збереження. Після певного рівня гра починає генерувати випадкову кількість ворогів, які в свою чергу теж обрані випадковим чином із наявних. Необхідно, щоб між запусками об’єкти з’являлись у тих же місцях, що і при першому запуску рівня. Для цього до даних про рівень додається це число, яке є аргументом при створені генератора випадкових чисел. Це всі дані необхідні для збереження, для коректного продовження гри між рівнями.

Зберігати та завантажувати дані можна з допомогою бази даних або просто до файлової системи. Ігровий застосунок повинен працювати без доступу до інтернету, це означає, що мати доступ до даних потрібно на

пристрої, де встановлений застосунок. Проблема бази даних у тому, що її потрібно встановлювати, налаштовувати та підтримувати роботу. У свою чергу робота із локальними файлами потребує розробки функціоналу для роботи із файлами та збереження даних. Тому для цього застосунку було реалізовано локальне збереження даних.

Для даного програмного забезпечення відсутня база даних чи обмін даними із зовнішніми компонентами, тому забезпечувати хешування чи шифрування даних не потрібно.

Висновки до розділу

У цьому розділі було розглянуто мову програмування C# та ігровий рушій Unity, його переваги та особливості для платформи Windows.

Було розглянуто процес розробки архітектури та структури програмного забезпечення. Було описано архітектурну MVC, яку використано для розробки ігрового застосунку, та специфіку використання цієї архітектури у Unity. Також розглянуто структуру гри, в якій бойові сцени відділені від ігрових локацій.

Далі було розглянуто компоненти на які було розбито програму. Особливо варто виділити основні сцени. Було розглянуто основні процеси та контролери, призначені для відслідковування подій. Далі розглянуто класи, що були створені для роботи застосунку а також їх методи. Були розглянуті компоннти Unity, які були використані дял розробки.

Розглянуто створення головного меню з використанням базових елементів інтерфейсу Unity. Також розглянуто розробку меню, яке вмикається під час гри. Розглянуто розробку функцій меню та його положення серед ігрових станів.

Розглянуто етапи створення та елементи необхідні для ігрової локації, по якій може переміщатися гравець та стикатися з вороже налаштованими агентами. Також описано можливості для дизайну рівнів, розставлення різних

декоративних об’єктів, обмежень для гравця та розробка умов переходу на наступний рівень.

Розглянуто розробку бойової сцени, а саме об’єкти та скрипти, які необхідно створювати та контролювати під час існування сцени, а також події, на які потрібно реагувати скриптам.

Під кінець були виділені основні дані для збереження та спосіб у який відбувається збереження даних.

3 ТЕСТУВАННЯ КОМП'ЮТЕРНОЇ ГРИ

3.1 Тестування ігрових застосунків

Тестування комп'ютерних ігор дещо відрізняється від тестування традиційного програмного забезпечення. Комп'ютерна гра є тісним поєднанням дизайну та програмної інженерії. На відміну від традиційного програмного забезпечення, метою якого є надання користувачеві послуги, метою ігрових застосунків є розвага[10]. Тобто звичайні застосунки або додатки призначені для виконання однієї або кількох завдань, для спрощення життя користувача. У свою чергу відеоігри є більш комплексними програмними продуктами, оскільки вони повинні захоплювати увагу користувача цікавими механіками, різноманітними подіями, а не тільки добре написану програму. Як результат ігровий застосунок націлений на отриманні гарного ігрового досвіду гравцем, а не ефективності роботи. В залежності від оцінки користувачів вимоги до відеогри можуть змінюватись, і як тестувати ці вимоги також. Отож для тестування ігрових застосунків потрібні досвідчені люди, гарно відлагодженні механізми пошуку несправності та інформованість про роботу механік ігор.

Тестувальник відеоігор, для перевірки коректності роботи застосунку, повинен мати логічну узгодженість, спостережливість, прогресивне мислення, розумність і вичерпне тестування функцій, стратегії гри та функціональності. Не менш важливим є розуміння принципів, характеристик та внутрішнього контексту програми.

Відеоігри не вважаються програмним забезпеченням на ефективність, тому їх тестування більше спрямоване не на перевірку точності, а на інтенсивність досвіду користувача.

3.2 Приклади тестування ігрового застосунку

Тестування головного меню, є одним із найпростіших для виконання. Адже компоненти меню повинні виконувати виключно функції у відповідь на дії користувача.

Таблиця 3.1 - Тестування роботи функції меню “New Game”

Мета тесту	Перевірити коректність відгуку натискання клавіші “New Game”
Початковий стан гри	Відкрите головне меню
Вхідні дані	Дані з пристрою вводу. Значення, що натиснуто на необхідну клавішу
Схема проведення тесту	Обробка вводу користувача та підготовка сцени.
Очікуваний результат	Початок нового ігрового стану на початковій локації з початковими характеристиками персонажа гри
Стан гри після проведення випробувань	У програмі завантажено сцену із початковими ігровим рівнем та характеристиками гравця

Таблиця 3.2 - Тестування роботи функції меню “Exit”

Мета тесту	Перевірити коректність відгуку натискання клавіші “Exit”
Початковий стан гри	Відкрите головне меню
Вхідні дані	Дані з пристрою вводу. Значення, що натиснуто на необхідну клавішу
Схема проведення тесту	Обробка вводу користувача та підготовка сцени
Очікуваний результат	Припинення роботи ігрового застосунку

Продовження таблиці 3.2

Стан гри після проведення випробувань	Ігровий застосунк вимкнений.

Таблиця 3.3 - Тестування роботи функції меню “Continue”

Мета тесту	Перевірити коректність відгуку натискання клавіші “Continue”
Початковий стан гри	Відкрите головне меню
Вхідні дані	Дані з пристрою вводу. Значення, що натиснуто на необхідну клавішу
Схема проведення тесту	Обробка вводу користувача та підготовка сцени
Очікуваний результат	Відновлення ігрового стану відповідно із прогресом гравця
Стан гри після проведення випробувань	Завантажена ігрова сцена на рівне, який досягнув користувач та із характеристиками ігрового персонажа

Ігрові рівні вміщують у собі багато об’єктів, в тому числі багато елементів інтерфейсу. Різні види об’єктів мають різні види взаємодій між собою. Через велику кількість взаємодій часто виникають помилки, які можна перевіряти тестами.

Таблиця 3.4 - Тестування переміщення ігрового персонажа

Мета тесту	Перевірити правильність руху ігрового персонажа
Початковий стан гри	Завантажена сцена із локацією ігрового світу
Вхідні дані	Дані з пристрою вводу. Напрямок руху.
Схема проведення тесту	Обробка вводу користувача та виконання команди протягом часу постування сигналу

Продовження таблиці 3.4

Очікуваний результат	Переміщення ігрового персонажа у відповідному напрямку
Стан гри після проведення випробувань	Координати персонажа змінені у відповідності до напрямку та часу руху

Таблиця 3.5 - Тестування зіткнення ігрового персонажа із об’єктами декору

Мета тесту	Перевірити правильність зіткнення ігрового персонажа із об’єктами декору
Початковий стан гри	Завантажена сцена із локацією ігрового світу, де є ігровий персонаж та об’єкти декору
Вхідні дані	Дані з пристрою вводу
Схема проведення тесту	Обробка даних що ввів користувач, перевірка перетину меж об’єктів
Очікуваний результат	Ігровий персонаж не може пройти крізь об’єкт декору
Стан програми після проведення випробувань	Персонаж гравця та об’єкт декору залишаються на стані такими ж як і до зіткнення

Таблиця 3.6 - Тестування зіткнення ігрового персонажа із невидимою стіною

Мета тесту	Перевірити правильність зіткнення ігрового персонажа із невидимою стіною
Початковий стан гри	Завантажена сцена із локацією ігрового світу, де є ігровий персонаж та невидима стіна
Вхідні дані	Дані з пристрою вводу
Схема проведення тесту	Обробка даних що ввів користувач, перевірка перетину меж об’єктів

Продовження таблиці 3.6

Очікуваний результат	Ігровий персонаж не може пройти крізь невидиму стіну
Стан програми після проведення випробувань	Персонаж гравця та невидима стіна залишаються на стані такими ж як і до зіткнення

Таблиця 3.7 - Тестування зіткнення ігрового персонажа із ворожим агентом

Мета тесту	Перевірити запуск скрипту при зіткненні ігрового персонажа із ворожим агентом
Початковий стан гри	Завантажена сцена із локацією ігрового світу, де є ігровий персонаж та ворожий агент
Вхідні дані	Дані з пристрою вводу
Схема проведення тесту	Обробка даних що ввів користувач, перевірка перетину меж об’єктів
Очікуваний результат	Розпочинається бойова сцена, із характеристиками агента та ігрового персонажа
Стан програми після проведення випробувань	Завантажена бойова сцена із параметрами, які відповідають параметрам агента та персонажа

Таблиця 3.8 - Тестування входу до зони активації клавіші для переходу на наступний рівень

Мета тесту	Перевірити правильність входу до зони активації клавіші для переходу на наступний рівень
Початковий стан гри	Завантажена сцена із локацією ігрового світу, де є ігровий персонаж та об’єкт для переходу до наступної локації

Продовження таблиці 3.8

Вхідні дані	Дані з пристрою вводу
Схема проведення тесту	Обробка даних що ввів користувач, перевірка знаходження персонажа у зоні активації
Очікуваний результат	Поява знака із клавішею для переходу на наступну локацію
Стан програми після проведення випробувань	Над зоною активації клавіші для переходу на наступний рівень наявний знак про можливість переходу на наступний рівень

Бойова сцена є основою, з точки зору розвитку навиків печаті та моторики рук.

Таблиця 3.9 - Тестування натискання букви, яка відображається на об’єкті

Мета тесту	Перевірити роботу скрипту при натисканні букви, яка відображається на об’єкті
Початковий стан гри	Завантажена бойова сцена, розташовані об’єкти з літерами
Вхідні дані	Дані з пристрою вводу
Схема проведення тесту	Обробка літери, що ввів користувач
Очікуваний результат	Знищення об’єкта із літерою, яку введено
Стан програми після проведення випробувань	Об’єкт із літерою відсутній, наступна літера доступна

Таблиця 3.10 - Тестування перетину об’єкта із літерою та ігровим персонажем

Мета тесту	Перевірити роботу скрипту при натисканні об’єкта із літерою та ігровим персонажем
Початковий стан гри	Завантажена бойова сцена, розташовані об’єкти з літерами

Продовження таблиці 3.10

Вхідні дані	Дані з пристрою вводу
Схема проведення тесту	Обробка даних про перетин двох об’єктів
Очікуваний результат	Знищення об’єкта із літерою, зменшення кількості очок здоров’я персонажа
Стан програми після проведення випробувань	Об’єкт із літерою відсутній, кількість очок здоров’я менше ніж до початку тесту

Таблиця 3.11 - Тестування ситуації за якої очки здоров’я персонажа рівні нулю

Мета тесту	Перевірити роботу скрипту при відсутності очок здоров’я персонажа
Початковий стан гри	Завантажена бойова сцена, кількість очок здоров’я персонажа рівна нулю
Вхідні дані	Дані з пристрою вводу
Схема проведення тесту	Обробка кількості очок здоров’я персонажа
Очікуваний результат	Припинення бойової сцени, результатом якої є програш гравця
Стан програми після проведення випробувань	Ворожий агент залишається на локації, гравець не отримав винагороду

Таблиця 3.12 - Тестування натискання букви, яка відсутня на об’єкті

Мета тесту	Перевірити роботу скрипту при натисканні букви, яка відсутня на об’єкті
Початковий стан гри	Завантажена бойова сцена, розташовані об’єкти з літерами
Вхідні дані	Дані з пристрою вводу
Схема проведення тесту	Обробка літери, що ввів користувач
Очікуваний результат	Об’єкт із літерою не буде знищено, кількість літер для помилки зменшена на 1

Продовження таблиці 3.12

Стан програми після проведення випробувань	Об’єкт із літерою присутній

Таблиця 3.13 - Тестування знищення останнього об’єкту із літерою

Мета тесту	Перевірити роботу скрипту при знищенні останнього об’єкту із літерою
Початковий стан гри	Завантажена бойова сцена, розташований останній об’єкти з літерами
Вхідні дані	Дані з пристрою вводу
Схема проведення тесту	Обробка літери, що ввів користувач
Очікуваний результат	Об’єкт із літерою буде знищено, перехід до сцени з ігровою локацією
Стан програми після проведення випробувань	Ворожий агент відсутній на локації, гравець отримав винагороду

3.3 Опис контрольного прикладу

Щоб перевірити коректну роботу гри, розглянемо приклад, метою тесту є перевірити отримання та виконання завдань. Для цього підготовлена спеціальна сцена, де розміщені усі необхідні ігрові об’єкти, а саме персонаж із діалогом, ворог для знищення. Вхідними данними є напрямок руху персонажа у сцені рівня та надруковані літери у бойовій сцені. Сцена показана на Рисунку 3.1.

Рисунок 3.1 – Сцена контрольного тесту

Першим етапом проведення тесту є перетнення колайдера неігрового персонажа, через що розпочнеться діалог. Початок діалогу зображений на Рисунку 3.2. У цьому діалозі нам нададуть завдання перемогти противника, якого ми зустрінемо продовжуючи свою дорогу вліво ігрової сцени.

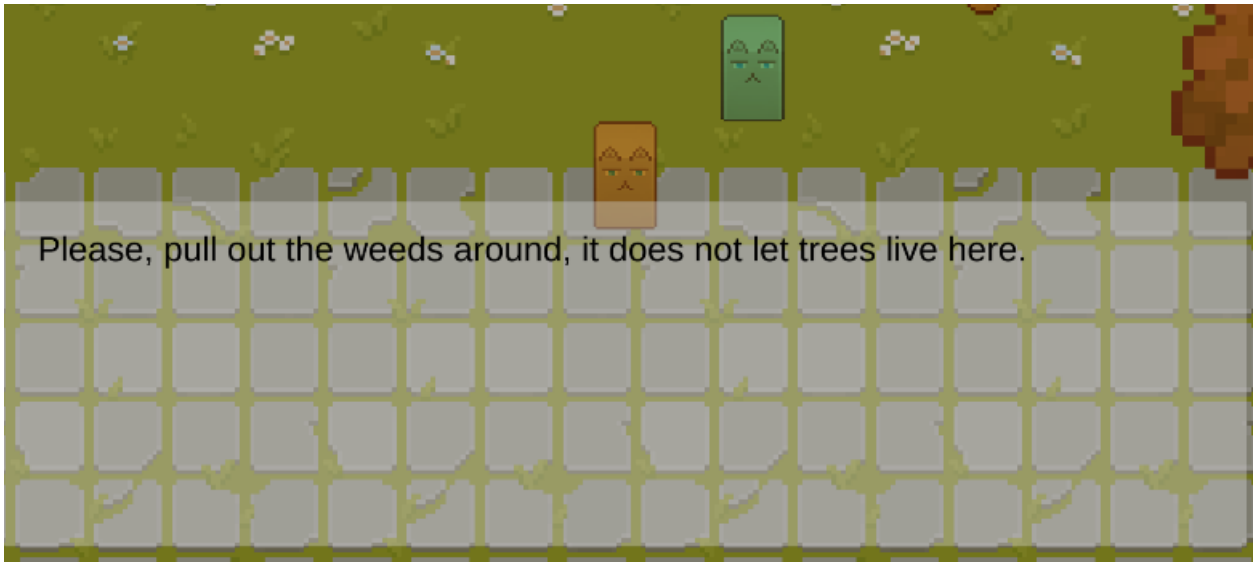


Рисунок 3.2 – Початок тесту із отримання завдання

Наступним кроком є знищення противника із яким потрібно розпочати бойову сцену зіткнувшись із ним. Після цього у разі перемоги ворожий

об’єкт буде знищено на сцені рівня, гравець отримає винагороду за перемогу, а завдання буде вважатись виконаним. Якщо гравець не впорається, та програє, ворог залишиться на сцені та буде доступний для початку нового бою поки не виграє гравець.

Очікуваним результатом є після виконання завдання, у не ігрового персонажа є новий діалог, що і показано на Рисунку 3.3. Після діалогу користувач отримує нагороду.

Рисунок 3.3 – Кінцевий діалог із нагородою

Після отримання нагороди завдання є виконаним, а гравець має винагороду.

Висновки до розділу

У даному розділі було розглянуто тестування ігрових застосунків, а саме відмінність тестування від звичайного програмного забезпечення. Описано складнощі тестування та .

Після чого були приведені приклади тестів для:

– сцени головного меню;
– сцени ігрових рівнів;
– бойової сцени.

Окрім тестування самих сцен, також було протестовано невеликі ситуації на коректність обробки даних вводу, команд користувача та взаємодії ігрових об’єктів.

У розділі розглянуто не всі ситуації для тестування комп’ютерної гри. Але завдяки проведеним тестам було підтверджено коректну роботу ігрового застосунку.

Було розглянуто контрольний приклад, спеціально для якого до гри було додано демонстраційни рівень, де є всі активності, які можна виконувати у грі. Також на основі контрольного прикладу продемонстрована ситуація із виконанням завдання, можливі варіанти розгалуження процесу виконання тесту та стан системи по завернню.

4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ІГРОВОГО ЗАСТОСУНКУ

4.1 Розгортання ігрового застосунку

Ігровий рушій Unity надає змогу зібрати готову гру для різних платформ. При чому для зібраної гри не потрібно ніяких додаткових програмних інтерфейсів чи бібліотек. Тобто для запуску потрібна лише директорія із усіма ресурсами гри.

Для збірки гри за допомогою Unity необхідно обрати пункт File, після чого із випадаючого меню натиснути на пункт меню Build Settings. З’явиться вікно із налаштуванням збірки, зображеним на Рисунку 4.1. У цьому вікні можна обрати сцени, які ви хочете включити до збірки, платформу, для якої буде виконуватись збірка, також можна налаштувати під яку архітектуру розрахована збірка. Це надає велику гнучкість для розробника, йому не потрібно окремо під кожну платформу чи операційну систему підлаштовувати відеогру. Достатньо завантажити необхідні бібліотеки для заданої платформи.

Ігровий застосунок доступний лише для операційної системи Windows. Для встановлення гри не потрібно стороннього програмного забезпечення. Для запуску потрібно завантажити ресурси гри, які є у відкритому доступі на сервісі GitHub серед релізних версій застосунку.

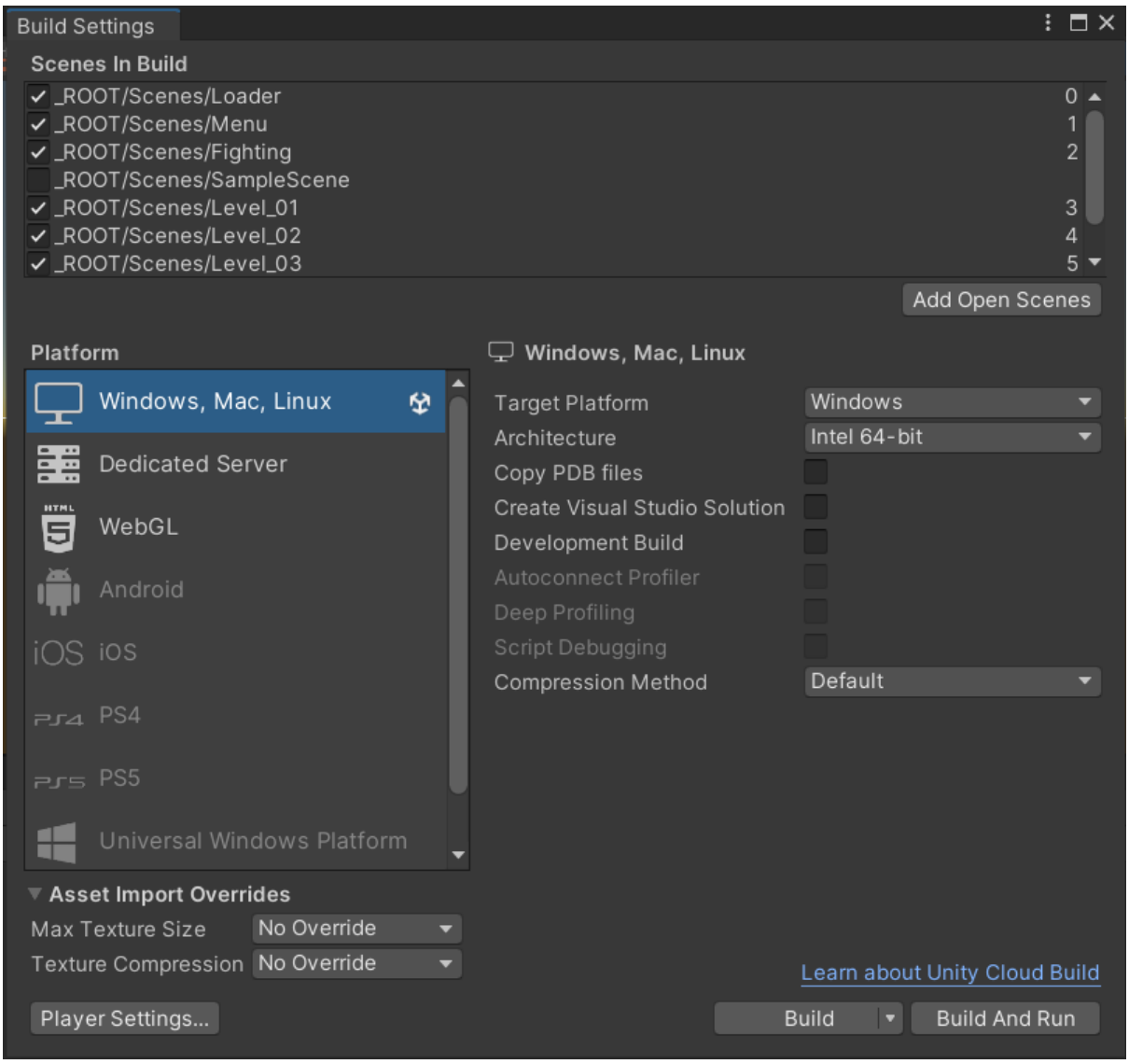


Рисунок 4.1 – Вікно Unity для налаштування збірки

4.2 Робота із ігровим застосунком

Після запуску ігрового застосунку, гравець бачить стартове меню Рисунок 4.2. Користувач з допомогою лівої клавіші комп'ютерної миші може виконувати наступні дії:

– Continue - продовжити гру із збережень гравця;

– New Game - розпочати нову гру;

– Exit - вийти із застосунку.

Рисунок 4.2 – Стартове меню

При відкритті ігрової сцени, користувача зустрічає ігроий рівень, на якому є персонаж підконтрольний гравцю. Приклад щойно завантаженого рівня є на Рисунку 4.3. Після завантаження ігрової сцени, гравець може пересуватись по ігровій локації за допомогою стрілок або клавіш WASD, де:

– W та стрілка вгору – рух вверх;

– A та стрілка вліво – рух вліво;

– S та стрілка вниз – рух ввниз;

– D та стрілка вправо – рух вправо.

Натиснувши та утискаючи клавішу SHIFT користувач може прискорювати свого персонажа.

Рисунок 4.3 – Ігровий рівень

При потряпляння у зону діалогу розпочинаєтсья діалог, який складається із скінченної кількості фраз. Для переходу до настуної фрази потрібно натиснути ліву клавішу комп'ютерної миші. Текст виводиться поступово, для моментального виводу тексту користувач має можливість натиснути натиснути ліву клавішу комп'ютерної миші. Приклад на Рисунку 4.4.

Рисунок 4.4 – Діалогове вікно

У будь-який момент комп’ютерної гри, окрім стартового меню, користувач може зупинити гру та відкрити ігрове меню з допомогою клавіші ESC. Після зупинки гри, на екрані з'являється наступні клавіші, як показано на Рисунку 4.5:

– Continue - продовжити гру;

– Menu – перейти до стартового меню;

– Exit - вийти із застосунку.

Рисунок 4.5 – Ігрове меню

При зіткненні із ворогом розпочинається бойова сцена. У цій сцені користувач повинен натискати літери, які відображуютья на прямуючих до ігрового персонажа об’єктах, до того як вони з ним зіткнуться. При зіткненні об’єкта-літери із персонажем гравця, об’єкт знищується, а у гравця зменшується кількість очок здоров’я. Кількість очок здоров’я відображається зліва у низу.

При натисканні літери, користувач може помилитись або випадково натиснути неправильний варіант. У такому разі існує механізм покарання,

який зменшує кількість доступних помилок за кожну неправильно введену літеру. Лічильник доступних помилок відображається у правій нижчній частині екрану, показаний на Рисунку 4.6. Якщо лічильник досягнув 0 то за кожну помилку при введенні літери ігровий персонаж втрачає очки здоров’я.

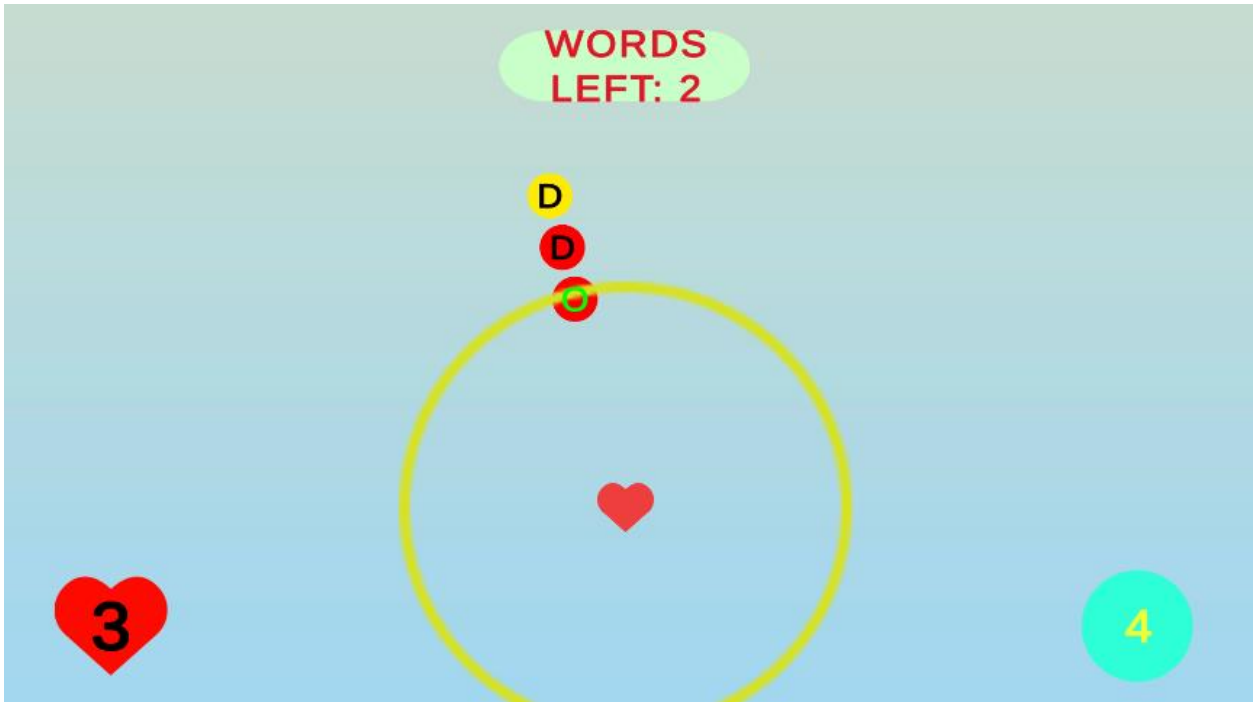


Рисунок 4.6 – Бойова сцена

Програти бойову сцену можливо лише за умови втрати усіх очок здоров’я (нижній правий кут на Рисунку 4.6). Після програшу користувач має змогу знову вступити у бій із противником.

При дослідженні ігрового світу, гравець може зіткнутись із магазином, де він має дві опції, а саме збільшити кількість здоров’я або збільшити радіус атаки, як показано на Рисунку 4.7. Виконати сказані дії гравець може обмежену кількість разів та в обмін на ігрову валюту. Покупка покращення виконується з допомогою натискання лівої клавіші комп'ютерної миші. Кожна наступна покупка коштує більше та кількість покупок вказані та обмежені.

Рисунок 4.7 – Магазин

Гравець може перейти до наступного рівня за певну кількість ігрової валюти, потрапивши у певну зону, яка позначена табличкою, як на Рисунку 4.8. Перехід виконується з допомогою натискання лівої клавіші комп'ютерної миші.

Рисунок 4.8 – Табличка переходу на наступний рівень

Висновки до розділу

У даному розділі було розглянуто спосіб збірки ігровго застосунку використовуючи інструменти, які надає ігровий рушій Unity. Описано вимоги, необхідні для ігрового застосунку, та спосіб завантаження найновішої версії гри.

Описано дії необхідні для запуску відеогри. Далі було описана робота користувача із ігровим застосунком та можливі дії користувача під час поботи відеогри. А саме:

- користування ігровим стартовим меню;
- пересування по ігровому рівню;
- перехід до ігорового меню та використання його функцій;
- використання діалогової системи та пропуски діалогів;
- позначення під час бойової сцени;
- умови перемоги або програшу в бойовій сцені;
- використання магазину;
- використання таблички переходу до іншого рівня.

ВИСНОВКИ

В рамках цього дипломного проекту була реалізована комп'ютерна дитяча гра для розвитку друкування та моторики рук. Створена гра дозволяє користувачеві розвивати навички печаті у ігровій формі та досліджувати ігровий світ розділений на невеликі локації.

В процесі створення комп’ютерної гри, було проаналізовано предметну область, а саме ігрові жанри, розподіл на вікові категорії, класифікація ігор, види монетизації, види ігор. Також було розглянуто уже існуючі рішення.

Для створення ігрового застосунку було проаналізовано можливі інструменти розробки. У результаті аналізу для розробки було обрано мову розробки C# та ігровий рушій Unity, на його основі C#. Було розглянуто можливості, переваги та недоліки ігрового рушія Unity.

В результаті виконання дипломного проєкту було спроєктовано архітектуру, виділено основні компоненти, визначено сцени із якими буде взаємодіяти користувач. Після чого було описано класи та їх методи, також базові класи ігрового рушія Unity використані для роботи ігрового застосунку.

Кожна сцена, взаємодія сцен та компоненти сцен було розглянуто у окремих розділах. У описі створення сцен розглянуто як використовуються класи створені у результаті розробки програмного забезпечення, а також розглянуто використання інструментів, що надає ігровий рушій Unity. Після чого було розглянуто задачу із збереженням прогресу користувача та можливі рішення.

Комп’ютерна гра була протестована згідно із розглянутими методиками текстування ігрових застосунків. Були перевірені різні функції та коректність роботи застосунку.

Розглянуто, як з допомогою ігрового рушія Unity було зібрано ігровий застосунок, завдяки чому для використання гри не потрібно встановлення ніяких бібліотек чи сторонніх програмних засобів.

Описано дії необхідні для запуску комп'ютерної дитячої гри. Варіанти використання та можливості користувача під час роботи ігрового застосунку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Частка домогосподарств з комп'ютером вдома у всьому світі з 2005 по 2019 рік[Електронний ресурс] – Режим доступу до ресурсу: https://www.statista.com/statistics/748551/worldwide-households-with-computer/
2) Відсоток домогосподарств у Сполучених Штатах із комп'ютером вдома з 1986 по 2016 рік[Електронний ресурс] – Режим доступу до ресурсу: https://www.statista.com/statistics/214641/household-adoption-rate-of-computer-in-the-us-since-1997/
3) Розподілення віку користувачів персональних комп'ютерів[Електронний ресурс] – Режим доступу до ресурсу: https://www.statista.com/statistics/748072/number-pc-gamers-world-platform/#:~:text=The%20statistic%20presents%20the%20number,between%2039%20and%2050%20worldwide
4) Аналіз динаміки розвитку ринку відеоігор, джерел його фінансування та особливостей монетизації продукції в даній сфері [Електронний ресурс] – Режим доступу до ресурсу: http://www.economy.nayka.com.ua/pdf/5_2021/4.pdf
5) GUIDE TO GAMES CLASSIFICATION [Електронний ресурс] – Режим доступу до ресурсу: https://pathfinder.academy/newearswick/wp-content/uploads/sites/7/2019/06/pbbfcGuidetogameclassification.pdf
6) Platform Comparison Between Games Console, Mobile Games And PC Games [Електронний ресурс] – Режим доступу до ресурсу: https://www.researchgate.net/publication/317811826_Platform_Comparison_Between_Games_Console_Mobile_Games_And_PC_Games
7) Why Video Game Genres Fail: A Classificatory Analysis [Електронний ресурс] – Режим доступу до ресурсу: https://surface.syr.edu/cgi/viewcontent.cgi?article=1167&context=istpub

8) Video Game Genre, Evolution and Innovation[Електронний ресурс]. Eludamos. Journal for Computer Game Culture, 2009 – с. 149 – Режим доступу до ресурсу: https://www.academia.edu/224767/_2009_Video_Game_Genre_Evolution_and_Innovation_journal_

9) Game Classification as Game Design: Construction Through Critical Analysis[Електронний ресурс] – Режим доступу до ресурсу: https://www.researchgate.net/publication/221217526_Game_Classification_as_Game_Design_Construction_Through_Critical_Analysis

10) J. Kasurinen, M. Palacin-Silva, and E. Vanhala, “What Concerns Game Developers? A Study on Game Development Processes, Sustainability and Metrics,” International Workshop on Emerging Trends in Soft-ware Metrics, WETSoM, С. 15–21, 2017 – Режим доступу до ресурсу: https://www.researchgate.net/publication/318294460_What_Concerns_Game_Developers_A_Study_on_Game_Development_Processes_Sustainability_and_Metrics

ДОДАТКИ

Додаток А Звіт подібності

UNICHECK

Ім'я користувача:
Лісовиченко Олег Іванович

ID перевірки:
1015404811

Дата перевірки:
03.06.2023 11:46:03 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
03.06.2023 11:47:35 EEST

ID користувача:
76913

Назва документа: **ІП-82_Бебех_ПЗ**

Кількість сторінок: **82** Кількість слів: **15094** Кількість символів: **114068** Розмір файлу: **1.19 MB** ID файлу: **1015068517**

4.01%
Схожість

Найбільша схожість: **0.9%** з джерелом з Бібліотеки (ID файлу: **1012700424**)

2.52% Джерела з Інтернету	80	Сторінка 84
4% Джерела з Бібліотеки	287	Сторінка 85

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

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

Замінені символи	1

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

Комп'ютерна дитяча гра для розвитку друкування та моторики рук

Текст програми

КПІ.ІП-8202.045480.03.12

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олена ХАЛУС

Нормоконтроль:
_____ Катерина ЛІЩУК

Виконавець:
_____ Олександр БЕБЕХ

Київ – 2023

ТЕКСТ ПРОГРАМНОГО КОДУ

Тексти програмного коду

Комп'ютерна дитяча гра для розвитку друкування та моторики рук

(Найменування програми (документа))

CD-R

(Вид носія даних)

38 арк, 34.4 MB

(Обсяг програми, арк., Кб)

Київ – 2023

Файл Dialog.cs

```
namespace _ROOT.Scripts.DialogSystem
{
    using System;
    using System.Collections;
    using TMPro;
    using UnityEngine;

    ///<summary>
    /// Клас, для поступового виводу тексту для користувача у певне поле
    ///</summary>
    public class Dialog : MonoBehaviour
    {
        ///<summary>
        /// Поле, куди воводиться текст
        ///</summary>
        [SerializeField] private TextMeshProUGUI textMeshProUGUI;

        ///<summary>
        /// Увесь текст, поділений на частини
        ///</summary>
        [SerializeField] private string[] lines;

        ///<summary>
        /// Швидкість виводу тексту
        ///</summary>
        [SerializeField] private float textSpeed;

        ///<summary>
        /// Функція, яка викликається по завершенню діалогу
        ///</summary>
        public event Action<bool> OnDialogEnd;

        ///<summary>
        /// Індекс нинішньої частини тексту, що виводиться
        ///</summary>
        private int index;

        ///<summary>
        /// При створенні об'єкту із цим скриптом, він виставляється неактивним
        ///</summary>
        private void Start()
        {
            gameObject.SetActive(false);
        }

        ///<summary>
        /// Поступовий вивід тексту та реакція на ввід користувача
        ///</summary>
        private void Update()
        {
            if (Input.GetMouseButtonDown(0))
            {
                if (textMeshProUGUI.text == lines[index])
                {
                    NextLine();
                }
                else
                {
                    StopAllCoroutines();
                    textMeshProUGUI.text = lines[index];
```

```
            }
        }
    }

    ///<summary>
    /// Початок поступового виводу тексту
    ///</summary>
    public void StartDialog()
    {
        textMeshProUGUI.text = String.Empty;
        index = 0;
        gameObject.SetActive(true);
        StartCoroutine(TypeLine());
    }

    ///<summary>
    /// Друк тексту до поля
    ///</summary>
    IEnumerator TypeLine()
    {
        foreach (var c in lines[index].ToCharArray())
        {
            textMeshProUGUI.text += c;
            yield return new WaitForSeconds(textSpeed);
        }
    }

    ///<summary>
    /// Перевірка та перехід до друку наступної частини тексту
    ///</summary>
    void NextLine()
    {
        if (index < lines.Length - 1)
        {
            index++;
            textMeshProUGUI.text = String.Empty;
            StartCoroutine(TypeLine());
        }
        else
        {
            PostDialog();
            OnDialogEnd?.Invoke(false);
            gameObject.SetActive(false);
        }
    }

    public virtual void PostDialog()
    {
    }
  }
}
```

Файл DialogTrigger.cs

```
namespace _ROOT.Scripts.DialogSystem
{
  using Game;
  using GlobalWorld;
  using UnityEngine;

  ///<summary>
  /// Клас, який реагує на вхід до колайдеру, щоб запустити діалог
  ///</summary>
```

```
public class DialogTrigger : MonoBehaviour
{
    ///<summary>
    /// Діалог для показу
    ///</summary>
    [SerializeField] private Dialog dialog;

    ///<summary>
    /// При перетині гравцем колайдеру розпочинається діалог
    ///</summary>
    private void OnTriggerEnter2D(Collider2D other)
    {
        var player = other.gameObject.GetComponentInParent<Player>();
        if (player)
        {
            player.DisableMove();
            dialog.OnDialogEnd += player.SubscribeForMove;
            dialog.OnDialogEnd += GameController.Instance.IsDialog;

            GameController.Instance.IsDialog(true);
            dialog.StartDialog();
        }
    }

    ///<summary>
    /// Припинення відслідковування діалогу при визоді із колайдеру
    ///</summary>
    private void OnTriggerExit2D(Collider2D other)
    {
        var player = other.gameObject.GetComponentInParent<Player>();
        if (player)
        {
            GameController.Instance.IsDialog();
            dialog.OnDialogEnd -= player.SubscribeForMove;
            dialog.OnDialogEnd -= GameController.Instance.IsDialog;
        }
    }

    ///<summary>
    /// Метод для задання діалога
    ///</summary>
    public void SetDialog(Dialog newDialog)
    {
        dialog = newDialog;
    }
}
```

Файл DialogWithReward.cs

```
namespace _ROOT.Scripts.DialogSystem
{
    using Saves.Player;
    using UnityEngine;
    public class DialogWithReward : Dialog
    {
        [SerializeField] private int reward = 20;

        private bool rewardTaken = false;
        public override void PostDialog()
        {
            if (!rewardTaken)
            {
```

```
            rewardTaken = true;
            PlayerSavable.Instance.Gold += reward;
        }
    }
  }
}
```

Файл WordsGenerator.cs

```
namespace _ROOT.Scripts.Dictionary
{
  using System.Collections.Generic;
  using UnityEngine;
  public class WordsGenerator
  {
    private Dictionary<int, List<string>> dictionary = new ();

    public WordsGenerator()
    {
      var file = Resources.Load<TextAsset>("words");

      IEnumerable<string> lines = file.text.Split("\r\n");

      foreach (var line in lines)
      {
        if (!dictionary.ContainsKey(line.Length))
        {
          dictionary.Add(line.Length, new List<string>());
        }
        dictionary[line.Length].Add(line.ToLower());
      }
    }

    public string GetWord(int length)
    {
      return dictionary[length][(int)(Random.value * dictionary[length].Count) % dictionary[length].Count];
    }

  }
}
```

Файл FightInterface.cs

```
namespace _ROOT.Scripts.Fight.Interface
{
  using TMPro;
  using UnityEngine;
  public class FightInterface : MonoBehaviour
  {
    [SerializeField] private TMP_Text HeartLabel;

    [SerializeField] private TMP_Text WordsLeftPanel;

    [SerializeField] private TMP_Text CharactersLeftPanel;

    [SerializeField] private Character character;

    [SerializeField] private UnitSpawner unitSpawner;

    private string WordsLeftFraze;
    private void Start()
    {
      WordsLeftFraze = "Words left: ";
      HeartLabel.SetText(character.hearts.ToString());
      WordsLeftPanel.SetText(WordsLeftFraze + unitSpawner.GetWordsLeft());
```

```
            CharactersLeftPanel.SetText(unitSpawner.GetCharactersLeft().ToString());
        }

        private void FixedUpdate()
        {
            HeartLabel.SetText(character.hearts.ToString());
            WordsLeftPanel.SetText(WordsLeftFraze + unitSpawner.GetWordsLeft());
            CharactersLeftPanel.SetText(unitSpawner.GetCharactersLeft().ToString());
        }

    }
}
```

Файл Character.cs

```
namespace _ROOT.Scripts.Fight
{
    using System;
    using Game;
    using GlobalWorld;
    using Saves.Player;
    using UnityEngine;
    public class Character : MonoBehaviour
    {

        [SerializeField] public int MaxHearts;

        [SerializeField] public int hearts;
        private void OnTriggerEnter2D(Collider2D other)
        {
            TryToDestroy(other);
        }

        public void Init()
        {
            hearts = MaxHearts = PlayerSavable.Instance.Hearts;
        }
        public void TakeAHit()
        {
            hearts--;
            if (hearts == 0)
            {
                GameEvents.StartFightEndEvent(FightResults.Lose);
                Destroy(gameObject);
            }
        }

        private void TryToDestroy(Collider2D other)
        {
            var unit = other.GetComponentInParent<Unit>();
            if (unit is not null)
            {
                unit.Destroy();
                TakeAHit();
            }
        }

        public void Heal(int heal = 1)
        {
            hearts += heal;

            if (hearts > MaxHearts)
            {
```

```
            hearts = MaxHearts;
        }
    }
  }
}
```

Файл FightWatcher.cs

```
namespace _ROOT.Scripts.Fight
{
  using System;
  using Game;
  using GlobalWorld;
  using Saves.Player;
  using UnityEngine;
  public class Character : MonoBehaviour
  {

    [SerializeField] public int MaxHearts;

    [SerializeField] public int hearts;
    private void OnTriggerEnter2D(Collider2D other)
    {
      TryToDestroy(other);
    }

    public void Init()
    {
      hearts = MaxHearts = PlayerSavable.Instance.Hearts;
    }
    public void TakeAHit()
    {
      hearts--;
      if (hearts == 0)
      {
        GameEvents.StartFightEndEvent(FightResults.Lose);
        Destroy(gameObject);
      }
    }

    private void TryToDestroy(Collider2D other)
    {
      var unit = other.GetComponentInParent<Unit>();
      if (unit is not null)
      {
        unit.Destroy();
        TakeAHit();
      }
    }

    public void Heal(int heal = 1)
    {
      hearts += heal;

      if (hearts > MaxHearts)
      {
        hearts = MaxHearts;
      }
    }
  }
}
```

Файл InputProvider.cs

```
namespace _ROOT.Scripts.Fight
{
    using System;
    using System.Linq;
    using UnityEngine;
    public class InputProvider : MonoBehaviour
    {
        public event Action<char> OnInput;

        private readonly char[] excludeChars = {'\b', '\n', ' '};

        private void Update()
        {
            var inputString = Input.inputString;
            foreach (var inputChar in inputString)
            {
                if (excludeChars.All(c => c != inputChar))
                {
                    OnInput?.Invoke(inputChar);
                }
            }
        }
    }
}
```

Файл RangeChecker.cs

```
namespace _ROOT.Scripts.Fight
{
    using Saves.Player;
    using UnityEngine;
    public class RangeChecker : MonoBehaviour
    {
        private void OnTriggerEnter2D(Collider2D other)
        {
            ChangeState(other);
        }

        private void ChangeState(Collider2D other)
        {
            var unit = other.GetComponentInParent<Unit>();
            if (unit)
            {
                unit.MakeAvailable();
            }
        }

        public void Init()
        {
            var range = PlayerSavable.Instance.Range;
            transform.localScale = Vector3.one * range;
        }
    }
}
```

Файл Unit.cs

```
namespace _ROOT.Scripts.Fight
{
    using System;
    using TMPro;
    using UnityEngine;
    public class Unit : MonoBehaviour
    {
        [SerializeField] private TMP_Text label;
```

```
[SerializeField] public float speed;

[SerializeField] public int lives = 1;

[SerializeField] public SpriteRenderer sprite;

public bool IsRestoreUnit;

public bool makeAllAvailable;

public string word;
public event Action<Unit> OnDeath;

public char TargetChar { get; private set; }

private Vector3 Target;

public bool isActive;

public void Init(char targetChar, Vector3 target, float _speed)
{
    TargetChar = targetChar;
    label.SetText(targetChar.ToString());
    label.faceColor = Color.black;
    Target = target;
    isActive = false;
    speed = _speed;
}

public void MakeAvailable()
{
    isActive = true;
    label.faceColor = Color.green;
}

public void MakeAllAvailable()
{
    makeAllAvailable = true;
    sprite.color = Color.yellow;
}

public void MakeRestore()
{
    IsRestoreUnit = true;
    sprite.color = Color.red;
}

private void OnInput(char inputChar)
{
    if (inputChar == TargetChar && isActive)
    {
        TakeHit();
        Die();
    }
}

public void Die()
{
    if (lives <= 0)
    {
        OnDeath?.Invoke(this);
```

```
            Destroy(gameObject);
        }
    }

    public void Destroy()
    {
        OnDeath?.Invoke(this);
        Destroy(gameObject);
    }
    public Vector3 GetTarget()
    {
        return Target;
    }

    public void TakeHit()
    {
        lives--;
    }
  }
}
```

Файл UnitModificator.cs

```
namespace _ROOT.Scripts.Fight
{
  using System;
  using Settings;
  using UnityEngine;
  using Random = UnityEngine.Random;
  public class UnitModificator
  {
    public float restoreChance;

    public float allAvailableChance;

    public void Init(EnemyStats settings)
    {
      restoreChance = settings.restoreChance;
      allAvailableChance = settings.allAvailableChance;
    }

    public void ModifyUnit(Unit unit)
    {
      if (Random.value < restoreChance)
      {
        unit.MakeRestore();
      }
      else if (Random.value < allAvailableChance)
      {
        unit.MakeAllAvailable();
      }
    }
  }
}
```

Файл UnitMovement.cs

```
namespace _ROOT.Scripts.Fight
{
  using UnityEngine;
  public class UnitMovement : MonoBehaviour
  {
    [SerializeField] private Unit unit;

    private void Update()
```

```
        {
            var position = transform.position;
            var target = unit.GetTarget();
            var direction = (target - position).normalized;
            var translation = direction * unit.speed * Time.deltaTime;
            transform.position = position + translation;
        }
    }
}
```

Файл UnitSpawner.cs

```
using Unity.VisualScripting;

namespace _ROOT.Scripts.Fight
{
    using System;
    using System.Collections;
    using System.Collections.Generic;
    using System.Linq;
    using Dictionary;
    using Game;
    using GlobalWorld;
    using Settings;
    using UnityEngine;
    using Random = UnityEngine.Random;
    public class UnitSpawner : MonoBehaviour
    {
        [SerializeField] private float spawnTimeLetter;

        [SerializeField] private float spawnTimeWord;

        [SerializeField] public int letterSpawnDelim;

        [SerializeField] public float restoreUnit;

        [SerializeField] public float availableUnit;

        [SerializeField] private Unit unitPrefab;

        [SerializeField] public InputProvider inputProvider;

        [SerializeField] public Character character;

        private UnitModificator modificator;

        private EnemyStats enemyStats;

        private WordsGenerator wordsGenerator;

        private Camera camera;

        private int wordsLeft;

        private bool finishSpawning;

        private Dictionary<string, List<Unit>> spawnedUnits;

        private int charactersLeft;

        private float calculatedSpeed;

        private int calculatedWordsAmount;
```

```
private IEnumerator SpawnRoutine()
{
    var waitLetter = new WaitForSeconds(spawnTimeLetter);
    wordsLeft = enemyStats.wordsAmount;
    inputProvider.OnInput += OnInput;
    while (true)
    {
        var word = GetWord();
        if (spawnedUnits.ContainsKey(word))
        {
            yield return new WaitForSeconds(1);
            continue;
        }
        spawnedUnits.Add(word, new List<Unit>());
        wordsLeft--;
        var position = GetRandomPosition();

        foreach (var c in word)
        {
            yield return waitLetter;
            var unit = Instantiate(unitPrefab, position, Quaternion.identity, transform);
            unit.Init(c, character.transform.position, calculatedSpeed);
            modificator.ModifyUnit(unit);
            spawnedUnits[unit.word = word].Add(unit);
            unit.OnDeath += OnUnitDeath;
        }

        if (wordsLeft == 0)
        {
            finishSpawning = true;
            break;
        }

        spawnTimeWord = word.Length * spawnTimeLetter;
        yield return new WaitForSeconds(spawnTimeWord);
    }
}

private void OnUnitDeath(Unit unit)
{
    unit.OnDeath -= OnUnitDeath;
    if (unit.makeAllAvailable)
    {
        MakeAllAvailable();
    }

    if (unit.IsRestoreUnit)
    {
        character.Heal();
    }

    spawnedUnits[unit.word].Remove(unit);

    if (spawnedUnits[unit.word].Count == 0)
    {
        spawnedUnits.Remove(unit.word);
    }

    if (IsFinishSpawning())
    {
        inputProvider.OnInput -= OnInput;
```

```
            GameEvents.StartFightEndEvent(FightResults.Win);
        }
    }

    private string GetWord()
    {
        return wordsGenerator.GetWord(enemyStats.wordsLength[(int) (Random.value *
enemyStats.wordsLength.Count) % enemyStats.wordsLength.Count]);
    }

    private Vector3 GetRandomPosition()
    {
        var size = camera.orthographicSize;
        var randomDirection = Vector3.zero;
        randomDirection.y += size * Random.Range(0.7f, 1f);
        randomDirection.x += size * Random.Range(-1f, 1f);
        randomDirection += character.transform.position;
        return randomDirection;
    }

    public int GetWordsLeft()
    {
        return wordsLeft;
    }

    public bool IsFinishSpawning()
    {
        return finishSpawning && spawnedUnits.Count == 0;
    }

    public void Init(EnemyStats stats, int level)
    {
        modificator = new ();
        spawnedUnits = new Dictionary<string, List<Unit>>();
        camera = Camera.main;
        wordsGenerator = new WordsGenerator();
        enemyStats = stats;
        calculatedSpeed = stats.speed + stats.speedPerLevel * level;
        spawnTimeLetter = letterSpawnDelim / calculatedSpeed;
        calculatedWordsAmount = stats.wordsAmount + stats.wordsAmountPerLevel * level;
        finishSpawning = false;
        modificator.Init(stats);
        charactersLeft = stats.mistakeFactor;
    }

    public void StartSpawn()
    {
        StartCoroutine(SpawnRoutine());
    }

    private void OnInput(char inputChar)
    {
        if (spawnedUnits.Count != 0
            && Time.timeScale != 0f)
        {
            bool used = true;
            foreach (var lst in spawnedUnits)
            {
                if (lst.Value.Count != 0)
                {
                    if (lst.Value.First().IsUnityNull())
                    {
```

```
                lst.Value.Remove(lst.Value.First());
            }

            if (lst.Value.First().isActive
                && lst.Value.First().TargetChar == inputChar)
            {
                lst.Value.First().TakeHit();
                lst.Value.First().Die();
                used = false;
                break;
            }
        }
    }

    if (used)
    {
        if (charactersLeft == 0)
        {
            character.TakeAHit();
        }
        else
        {
            charactersLeft--;
        }
    }
  }
}

private void MakeAllAvailable()
{
  foreach (var lst in spawnedUnits)
  {
    if (lst.Value.Count != 0)
    {
      foreach (var unit in lst.Value)
      {
        unit.MakeAvailable();
      }
    }
  }
}

public int GetCharactersLeft()
{
  return charactersLeft;
}

private void OnDestroy()
{
  if (!IsFinishSpawning())
  {
    GameEvents.StartFightEndEvent(FightResults.Lose);
  }
}
}
}
```

Файл AppStarter.cs

```
namespace _ROOT.Scripts.Game
{
  using Saves;
  using UnityEngine;
```

```
    public class AppStarter : MonoBehaviour
    {
        private void Awake()
        {
            PrepareSaves();
            GameController.Instance.LoadMenu();
        }

        private void PrepareSaves()
        {
            SaveController.Instance.PrepareSave();
        }
    }
}
```

Файл CameraController.cs

```
namespace _ROOT.Scripts.Game
{
    using Tools;
    using Cinemachine;
    using UnityEngine;
    public class CameraController : Singleton<CameraController>
    {
        [SerializeField] private CinemachineVirtualCamera worldCamera;

        private void Start()
        {
            DisableWorldCamera();
        }

        public void FocusWorldCamera(Transform transform)
        {
            worldCamera.Follow = transform;
            worldCamera.LookAt = transform;
        }

        public void EnableWorldCamera()
        {
            worldCamera.gameObject.SetActive(true);
        }

        public void DisableWorldCamera()
        {
            worldCamera.gameObject.SetActive(false);
        }
    }
}
```

Файл GameController.cs

```
namespace _ROOT.Scripts.Game
{
    using System.Threading.Tasks;
    using GlobalWorld;
    using GlobalWorld.Enemies;
    using Saves;
    using Saves.Level;
    using Tools;
    using UnityEngine;
    public class GameController : Singleton<GameController>
    {
        [SerializeField] private SceneController sceneController;

        [SerializeField] private CameraController cameraController;
```

```
[SerializeField] private PlayerProvider playerProvider;

private bool is_demo;

private GameState state;

public GameState GetState()
{
    return state;
}

public void LoadMenu()
{
    sceneController.LoadMenu();
    state = GameState.menu;
}
public void StartFight(Enemy enemy)
{
    sceneController.SwitchToFightScene();
    cameraController.DisableWorldCamera();
    FightController.Instance.PrepareFight(enemy);
    state = GameState.fight;
}

public void StartGame()
{
    sceneController.UnloadMenu();
    sceneController.SwitchToLevelScene(LevelSavable.Instance.current_level);
    state = GameState.level;
}

public void StartDemo()
{
    sceneController.UnloadMenu();
    sceneController.SwitchToLevelScene(-1);
    is_demo = true;
    state = GameState.level;
}

public void StartNextLevel()
{
    sceneController.UnloadLevelScene(LevelSavable.Instance.current_level);
    LevelSavable.Instance.current_level++;
    sceneController.SwitchToLevelScene(LevelSavable.Instance.current_level);
    SaveController.Instance.SaveState();

}

public void Exit()
{
    switch (state)
    {
      case GameState.fight:
         sceneController.UnloadFightScene();
         cameraController.EnableWorldCamera();
         state = GameState.level;

         break;
      case GameState.level:
         int level = is_demo ? -1 : LevelSavable.Instance.current_level;
         sceneController.UnloadLevelScene(level);
```

```
                sceneController.LoadMenu();
                is_demo = false;
                state = GameState.menu;

                break;
            case GameState.menu:
                sceneController.UnloadMenu();
                state = GameState.load;

                break;
        }
    }

    public void ExitFight()
    {
        sceneController.UnloadFightScene();
        cameraController.EnableWorldCamera();
        state = GameState.level;
    }

    public void IsDialog(bool inDialog = false)
    {
        if (inDialog)
        {
            state = GameState.dialog;
        }
        else
        {
            state = GameState.level;
        }
    }
  }
}
```

Файл GameEvents.cs

```
namespace _ROOT.Scripts.Game
{
  using System;
  using GlobalWorld;
  using GlobalWorld.Enemies;

  public static class GameEvents
  {
    public static event Action<string> OnFightStart;

    public static event Action<Enemy> OnEnemySlayed;

    public static event Action<FightResults> OnFightEnd;

    public static void StartFightEndEvent(FightResults res)
    {
      OnFightEnd?.Invoke(res);
    }

    public static void SlayedEnemyEvent(Enemy enemy)
    {
      OnEnemySlayed?.Invoke(enemy);
    }
  }
}
```

Файл GameState.cs

```
namespace _ROOT.Scripts.Game
{
    public enum GameState
    {
        load = 0,
        menu,
        level,
        fight,
        dialog
    }
}
```

Файл PlayerProvider.cs

```
namespace _ROOT.Scripts.Game
{
    using GlobalWorld;
    using Tools;
    using UnityEngine;
    public class PlayerProvider : Singleton<PlayerProvider>
    {
        public Player Player => player == null ? player = FindObjectOfType<Player>() : player;

        private Player player;
    }
}
```

Файл BaseEnemy.cs

```
namespace _ROOT.Scripts.GlobalWorld.Enemies
{
    public class BaseEnemy : Enemy
    {

    }
}
```

Файл Enemy.cs

```
namespace _ROOT.Scripts.GlobalWorld.Enemies
{
    using System;
    using System.Collections.Generic;
    using UnityEngine;
    public abstract class Enemy : MonoBehaviour
    {
        [SerializeField] public string Name;

        [SerializeField] public int level = 1;

        [SerializeField] public int Reward;

        public virtual void DestroySelf()
        {
            Destroy(gameObject);
        }
    }
}
```

Файл EnemySpawner.cs

```
namespace _ROOT.Scripts.GlobalWorld.EnemySpawner
{
    using Saves.Level;
    using UnityEngine;
    using System;
    using System.Collections.Generic;
```

```
using Random = System.Random;
using Enemies;

public class EnemySpawner : MonoBehaviour
{
    private Random rand;

    [SerializeField] public int enemyCount = 1;

    [SerializeField] public int deltaEnemyCount = 0;

    [SerializeField] public List<GameObject> unitsForSpawn;

    [SerializeField] public Vector3 spawnFrom;

    [SerializeField] public Vector3 spawnTo;

    [SerializeField] public GameObject enemyList;

    private const int MAX_ATTEMPTS = 100;

    private void Awake()
    {
        rand = new Random(LevelSavable.Instance.level_seed * LevelSavable.Instance.current_level);
    }

    public void SpawnEnemies()
    {
        var enemyToSpawn = enemyCount + rand.Next(0, LevelSavable.Instance.current_level +
deltaEnemyCount);
        int attempts = 0;
        while (enemyToSpawn > 0)
        {
            attempts++;
            if (attempts >= MAX_ATTEMPTS)
            {
                break;
            }

            var position = GetRandomVector3();

            var unit = Instantiate(unitsForSpawn[rand.Next(0,unitsForSpawn.Count)], position, Quaternion.identity,
transform);

            bool noCollisions = CheckForCollision(unit);

            if (noCollisions)
            {
                Destroy(unit.gameObject);
            }
            else
            {
                enemyToSpawn--;
                unit.transform.parent = enemyList.transform;
                var enemy = unit.GetComponentInParent<Enemy>();
                enemy.level = LevelSavable.Instance.current_level;
                enemy.Reward *= LevelSavable.Instance.current_level;
            }
        }
    }

    private Vector3 GetRandomVector3()
```

```
        {
            var randomDirection = Vector3.zero;
            randomDirection.y += rand.Next((int)(spawnFrom.y * 100), (int)(spawnTo.y * 100)) / 100f;
            randomDirection.x += rand.Next((int)(spawnFrom.x * 100), (int)(spawnTo.x * 100)) / 100f;
            return randomDirection;
        }

        private bool CheckForCollision(GameObject unit)
        {
            var unitCollider = unit.GetComponent<Collider2D>();

            if (!unitCollider)
            {
                return true;
            }

            foreach (var spawned in FindObjectsOfType<GameObject>())
            {

                var spawnedCollider = spawned.GetComponent<Collider2D>();

                if (spawnedCollider)
                {
                    if (spawnedCollider.bounds.Intersects(unitCollider.bounds))
                    {
                        if (ReferenceEquals(spawnedCollider, unitCollider))
                        {
                            continue;
                        }
                        return true;
                    }
                }
            }

            return false;
        }
    }
}
```

Файл LevelStarterWithGenerator.cs

```
namespace _ROOT.Scripts.GlobalWorld.EnemySpawner
{
    using Game;
    using GlobalWorld;
    using UnityEngine;
    public class LevelStarterWithGenerator : LevelStarter
    {
        [SerializeField] public EnemySpawner Spawner;

        private void Awake()
        {
            CameraController.Instance.FocusWorldCamera(PlayerProvider.Instance.Player.transform);
            CameraController.Instance.EnableWorldCamera();
            Spawner.SpawnEnemies();
        }
    }
}
```

Файл PlayerInterface.cs

```
namespace _ROOT.Scripts.GlobalWorld.Interface
{
    using System;
    using Saves.Player;
```

```
using TMPro;
using UnityEngine;
public class PlayerInterface : MonoBehaviour
{
    [SerializeField] private TMP_Text HeartLabel;
    [SerializeField] private TMP_Text GoldLabel;
    [SerializeField] private TMP_Text RangeLabel;

    private void FixedUpdate()
    {
        HeartLabel.SetText(PlayerSavable.Instance.Hearts.ToString());
        GoldLabel.SetText(PlayerSavable.Instance.Gold.ToString());
        RangeLabel.SetText(PlayerSavable.Instance.Range.ToString());

    }
}
}
```

Файл Shop.cs

```
namespace _ROOT.Scripts.GlobalWorld.Shops
{
    using Saves.Player;
    using TMPro;
    using UnityEngine;
    public class Shop : MonoBehaviour
    {
        [SerializeField] public GameObject OtherGameObject;

        [SerializeField] public int basePrice;

        [SerializeField] public int priceMultiplier;

        [SerializeField] public int rangeDelim;

        [SerializeField] private TMP_Text HeartLabel;

        [SerializeField] private TMP_Text RangeLablePanel;

        [SerializeField] private TMP_Text ItemsLeft;

        [SerializeField] public int BuyLimit;
        private void Awake()
        {
            OtherGameObject.SetActive(false);
            ItemsLeft.SetText("Items left: " + BuyLimit);
        }

        private void Update()
        {
            HeartLabel.SetText(CalculatePrice(PlayerSavable.Instance.Hearts).ToString());
            RangeLablePanel.SetText(CalculatePrice(PlayerSavable.Instance.Range - rangeDelim).ToString());
        }

        private void OnTriggerEnter2D(Collider2D other)
        {
            var otherPlayer = other.GetComponentInParent<Player>();
            if (otherPlayer)
            {
                OtherGameObject.SetActive(true);
            }
        }
```

```
    private void OnTriggerExit2D(Collider2D other)
    {
        var otherPlayer = other.GetComponentInParent<Player>();
        if (otherPlayer)
        {
            OtherGameObject.SetActive(false);
        }
    }

    public void BuyHeart()
    {
        if (BuyLimit > 0 && PlayerSavable.Instance.Gold >= CalculatePrice(PlayerSavable.Instance.Hearts))
        {
            PlayerSavable.Instance.Gold -= CalculatePrice(PlayerSavable.Instance.Hearts);
            PlayerSavable.Instance.Hearts++;
            BuyLimit--;
            ItemsLeft.SetText("Items left:" + BuyLimit);
        }
    }

    public void BuyRange()
    {
        if (BuyLimit > 0 && PlayerSavable.Instance.Gold >= CalculatePrice(PlayerSavable.Instance.Range -
rangeDelim))
        {
            PlayerSavable.Instance.Gold -= CalculatePrice(PlayerSavable.Instance.Range - rangeDelim);
            PlayerSavable.Instance.Range++;
            BuyLimit--;
            ItemsLeft.SetText("Items left:" + BuyLimit);
        }
    }

    public int CalculatePrice(int value)
    {
        return basePrice * value * priceMultiplier;
    }
  }
}
```

Файл ClearLocationWatcher.cs

```
namespace _ROOT.Scripts.GlobalWorld
{
  using System;
  using System.Collections.Generic;
  using DialogSystem;
  using Game;
  using GlobalWorld.Enemies;
  using Saves.Player;
  using Unity.VisualScripting;
  using UnityEngine;
  public class ClearLocationWatcher : Watcher
  {
    [SerializeField] List<GameObject> objectsToDestroy;

    [SerializeField] private Dialog dialogForMission;

    [SerializeField] private DialogWithReward dialogForComplete;

    [SerializeField] private DialogTrigger dialogTrigger;

    private bool isQuestCompleted;
```

```
        private void Awake()
        {
            isQuestCompleted = false;
            GameEvents.OnEnemySlayed += DoOtherStuff;
        }

        public void DoOtherStuff(Enemy enemy)
        {
            if (!isQuestCompleted)
            {
                var itemToRemove = objectsToDestroy.Find(obj => GameObject.ReferenceEquals( obj,
enemy.gameObject));
                if (itemToRemove)
                {
                    objectsToDestroy.Remove(itemToRemove);
                }

                if (IsMissionComplete())
                {
                    dialogTrigger.SetDialog(dialogForComplete);
                    isQuestCompleted = true;
                    GameEvents.OnEnemySlayed -= DoOtherStuff;
                }
            }
        }

        private bool IsMissionComplete()
        {
            return objectsToDestroy.Count == 0;
        }
    }
}
```

Файл Watcher.cs

```
namespace _ROOT.Scripts.GlobalWorld
{
    using System;
    using Game;
    using UnityEngine;

    public class Watcher : MonoBehaviour
    {
        [SerializeField] GameObject GlobalInterface;
        private void FixedUpdate()
        {
            if (NeedHideInterface())
            {
                GlobalInterface.SetActive(false);
            }
            else
            {
                GlobalInterface.SetActive(true);
            }
        }

        private bool NeedHideInterface()
        {
            return GameController.Instance.GetState() == GameState.fight
                || GameController.Instance.GetState() == GameState.dialog;
        }
    }
}
```

Файл FightController.cs

```
namespace _ROOT.Scripts.GlobalWorld
{
    using System;
    using Game;
    using UnityEngine;

    public class Watcher : MonoBehaviour
    {
        [SerializeField] GameObject GlobalInterface;
        private void FixedUpdate()
        {
            if (NeedHideInterface())
            {
                GlobalInterface.SetActive(false);
            }
            else
            {
                GlobalInterface.SetActive(true);
            }
        }

        private bool NeedHideInterface()
        {
            return GameController.Instance.GetState() == GameState.fight
                || GameController.Instance.GetState() == GameState.dialog;
        }
    }
}
```

Файл FightResults.cs

```
namespace _ROOT.Scripts.GlobalWorld
{
    public enum FightResults
    {
        Lose = 0,
        Win
    }
}
```

Файл LevelStarter.cs

```
namespace _ROOT.Scripts.GlobalWorld
{
    using System;
    using Game;
    using UnityEngine;
    public class LevelStarter : MonoBehaviour
    {
        private void Awake()
        {
            CameraController.Instance.FocusWorldCamera(PlayerProvider.Instance.Player.transform);
            CameraController.Instance.EnableWorldCamera();
        }
    }
}
```

Файл NextLevelController.cs

```
namespace _ROOT.Scripts.GlobalWorld
{
    using System;
    using Game;
    using Saves.Player;
    using TMPro;
```

```
using UnityEngine;
public class NextLevelController : MonoBehaviour
{
    [SerializeField] public GameObject plateGameObject;

    [SerializeField] private int Price;

    [SerializeField] private TMP_Text PriceLabel;

    private void Awake()
    {
        plateGameObject.SetActive(false);
    }

    private void FixedUpdate()
    {
        PriceLabel.SetText(Price.ToString());
    }

    private void OnTriggerEnter2D(Collider2D other)
    {
        var otherPlayer = other.GetComponentInParent<Player>();
        if (otherPlayer)
        {
            plateGameObject.SetActive(true);
        }
    }

    private void OnTriggerExit2D(Collider2D other)
    {
        var otherPlayer = other.GetComponentInParent<Player>();
        if (otherPlayer)
        {
            plateGameObject.SetActive(false);
        }
    }

    public void GoToNextLevel()
    {
        if (PlayerSavable.Instance.Gold >= Price)
        {
            PlayerSavable.Instance.Gold -= Price;
            GameController.Instance.StartNextLevel();
        }
    }
}
}
```

Файл Player.cs

```
namespace _ROOT.Scripts.GlobalWorld
{
    using System;
    using Game;
    using GlobalWorld.Enemies;
    using Saves;
    using Saves.Player;
    using Tools;
    using UnityEngine;
    public class Player : MonoBehaviour
    {
        [SerializeField] public float Speed;
```

```
[SerializeField] public float ShiftSpeed;

private bool IsSprint;

public bool CanMove;

private Vector2 desireMovement;

private void Awake()
{
    CanMove = true;
}

private void Update()
{
    IsSprint = Input.GetKey(KeyCode.LeftShift);
    var horizontalMovement = Input.GetAxis("Horizontal");
    var verticalMovement = Input.GetAxis("Vertical");
    desireMovement = new Vector2(horizontalMovement, verticalMovement);
}

private void FixedUpdate()
{
    if (CanMove)
    {
        Move();
    }
}

private void Move()
{
    Vector3 direction = desireMovement.normalized;
    var speed = Speed;
    if (IsSprint)
    {
        speed *= ShiftSpeed;
    }

    transform.position += direction * speed * Time.fixedDeltaTime;
}

private void OnTriggerEnter2D(Collider2D other)
{
    var enemy = other.gameObject.GetComponentInParent<Enemy>();
    if (CheckForFight(enemy))
    {
        DisableMove();
        GameController.Instance.StartFight(enemy);
    }
}

private bool CheckForFight(Enemy enemy)
{
    return enemy;
}

public void DisableMove()
{
    CanMove = false;
}

public void EnableMove()
```

```
        {
            CanMove = true;
        }

        [EditorButton]
        public void AddGold()
        {
            PlayerSavable.Instance.Gold += 100;
        }

        public void SubscribeForMove(bool isDialogEnd)
        {
            CanMove = !isDialogEnd;
        }
    }
}
```

Файл GameMenu.cs

```
namespace _ROOT.Scripts.Menus
{
    using System;
    using Game;
    using Unity.VisualScripting;
    using UnityEditor;
    using UnityEngine;
    public class GameMenu : MonoBehaviour
    {
        [SerializeField] public GameObject menu;

        private bool isStoped;

        private void Awake()
        {
            isStoped = false;
            menu.SetActive(false);
        }

        private void FixedUpdate()
        {
            if (Input.GetKey(KeyCode.Escape) && GameController.Instance.GetState() != GameState.menu)
            {
                if (isStoped)
                {
                    Resume();
                }
                else
                {
                    Pause();
                }
            }
        }

        public void Resume()
        {
            menu.SetActive(false);
            Time.timeScale = 1f;
            isStoped = false;
        }

        private void Pause()
        {
            menu.SetActive(true);
```

```
            Time.timeScale = 0f;
            isStoped = true;
        }

        public void ExitToMenu()
        {
            Time.timeScale = 1f;
            GameController.Instance.Exit();
            Resume();
        }

        public void Exit()
        {
            Time.timeScale = 1f;
            Application.Quit();
        }
    }
}
```

Файл StartMenu.cs

```
namespace _ROOT.Scripts.Menus
{
    using Game;
    using Saves;
    using Saves.Level;
    using UnityEngine;
    public class StartMenu : MonoBehaviour
    {
        public void ContinueGame()
        {
            SaveController.Instance.LoadSave();
            StartGame();
        }

        public void StartGame()
        {
            GameController.Instance.StartGame();
        }

        public void StartNewGame()
        {
            SaveController.Instance.PrepareSave(true);
            ContinueGame();
        }

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

        public void DemoGame()
        {
            SaveController.Instance.PrepareDemo();
            GameController.Instance.StartDemo();
        }
    }
}
```

Файл LevelSavable.cs

```
namespace _ROOT.Scripts.Saves.Level
{
    using System;
    public class LevelSavable : SavableComponent<LevelSave>
```

```
    {
        public static LevelSavable Instance => instance ??= new LevelSavable();
        private static LevelSavable instance;

        public int current_level { get; set; }

        public int level_seed { get; set; }

        public virtual LevelSave PrepareInitial()
        {
            return new LevelSave()
            {
                current_level = 1,
                level_seed = new Random().Next()
            };
        }

        public override LevelSave Serialize()
        {
            return new LevelSave
            {
                current_level = current_level,
                level_seed = level_seed,
            };
        }

        public override void Deserialize(LevelSave save)
        {
            current_level = save.current_level;
            level_seed = save.level_seed;
        }
    }
}
```

Файл LevelSave.cs

```
namespace _ROOT.Scripts.Saves.Level
{
    public class LevelSave : SaveComponent
    {
        public int current_level;

        public int level_seed;

    }
}
```

Файл PlayerSavable.cs

```
namespace _ROOT.Scripts.Saves.Player
{
    using Settings;
    using UnityEngine;
    public class PlayerSavable : SavableComponent<PlayerSave>
    {
        public static PlayerSavable Instance => instance ??= new PlayerSavable();
        private static PlayerSavable instance;

        public int Gold { get; set; }
        public int Range { get; set; }
        public int Hearts { get; set; }

        public virtual PlayerSave PrepareInitial()
        {
            var player = Resources.Load<PlayerDefaultSettings>($"Settings/{nameof(PlayerDefaultSettings)}").stats;
```

```
            return new PlayerSave()
            {
                Gold = player.Gold,
                Range = player.Range,
                Hearts = player.Hearts,
            };
        }

        public override PlayerSave Serialize()
        {
            return new PlayerSave
            {
                Gold = Gold,
                Range = Range,
                Hearts = Hearts,
            };
        }

        public override void Deserialize(PlayerSave save)
        {
            Gold = save.Gold;
            Range = save.Range;
            Hearts = save.Hearts;
        }
    }
}
```

Файл PlayerSave.cs

```
namespace _ROOT.Scripts.Saves.Player
{
    public class PlayerSave : SaveComponent
    {
        public int Gold;
        public int Range;
        public int Hearts;
    }
}
```

Файл PlayerPrefsStoredValue.cs

```
namespace _ROOT.Scripts.Saves
{
    using System;
    using UnityEngine;

    public class PlayerPrefsStoredValue<T>
    {
        public T Value
        {
            get => value;
            set
            {
                this.value = value;
                Save(value);
                onValueChanged(value);
            }
        }

        private string FullKey => $"local:{key}";

        private readonly string key;
        private readonly Action<T> onValueChanged;
        private T value;
```

```
public PlayerPrefsStoredValue(string key, T defaultValue) : this(key, _ => { }, defaultValue)
{
}

public PlayerPrefsStoredValue(string key, Action<T> onValueChanged, T defaultValue)
{
    this.key = key;
    this.onValueChanged = onValueChanged;
    value = (T) Restore(defaultValue);

    onValueChanged(value);
}

private void Save(T value)
{
    PlayerPrefs.SetString(FullKey, value.ToString());
}

private object Restore(T defaultValue)
{
    if (!PlayerPrefs.HasKey(FullKey))
    {
        Save(defaultValue);
        return defaultValue;
    }

    var stringValue = PlayerPrefs.GetString(FullKey);
    if (TryParseString(stringValue, out var result))
    {
        return result;
    }

    throw new Exception($"{typeof(T)} is not supported in PlayerPrefsStoredValue");
}

private bool TryParseString(string stringValue, out T value)
{
    try
    {
        return ParseString(stringValue, out value);
    }
    catch (Exception e)
    {
        Debug.LogError($"Failed to convert type {typeof(T)} to string");
        Debug.LogException(e);
        value = default;
        return false;
    }
}

private bool ParseString(string stringValue, out T value)
{
    var type = typeof(T);
    if (type.IsEnum)
    {
        value = (T) Enum.Parse(type, stringValue);
        return true;
    }

    value = (T) Convert.ChangeType(stringValue, type);
    return true;
}
```

```
    }
}
```

Файл SavableComponent.cs

```
namespace _ROOT.Scripts.Saves
{
    public interface ISavable
    {
    }

    public abstract class SavableComponent<T> : ISavable where T : SaveComponent, new()
    {
        public virtual T PrepareInitial()
        {
            return new T();
        }

        public abstract T Serialize();

        public abstract void Deserialize(T save);
    }
}
```

Файл SaveComponent.cs

```
namespace _ROOT.Scripts.Saves
{
    public class SaveComponent
    {

    }
}
```

Файл SaveController.cs

```
namespace _ROOT.Scripts.Saves
{
    using System;
    using Player;
    using Level;
    public class SaveController
    {
        public static SaveController Instance => instance ??= new SaveController();
        private static SaveController instance;
        private bool is_demo = false;

        private readonly SaveWorker saveWorker;

        private SaveController()
        {
            saveWorker = new SaveWorker();
        }

        public void LoadSave()
        {
            is_demo = false;
            var saveModel = saveWorker.Load();
            PlayerSavable.Instance.Deserialize(saveModel.PlayerSave);
            LevelSavable.Instance.Deserialize(saveModel.LevelSave);
        }

        public void SaveState()
        {
            if (is_demo)
            {
```

```
            return;
        }
        var saveModel = new SaveModel();
        saveModel.header = new Header();
        saveModel.header.time = DateTime.Now;
        saveModel.header.version = 1;
        saveModel.PlayerSave = PlayerSavable.Instance.Serialize();
        saveModel.LevelSave = LevelSavable.Instance.Serialize();
        saveWorker.Save(saveModel);
    }

    public void PrepareSave(bool delete = false)
    {
        if (saveWorker.Prepare() || delete)
        {
            var saveModel = new SaveModel();
            saveModel.header = new Header();
            saveModel.header.time = DateTime.Now;
            saveModel.header.version = 1;

            saveModel.PlayerSave = PlayerSavable.Instance.PrepareInitial();

            saveModel.LevelSave = LevelSavable.Instance.PrepareInitial();
            saveWorker.Save(saveModel);
        }
    }

    public void PrepareDemo()
    {
        is_demo = true;

        var saveModel = saveWorker.LoadDemo();

        PlayerSavable.Instance.Deserialize(saveModel.PlayerSave);
    }
  }
}
```

Файл SaveModel.cs

```
namespace _ROOT.Scripts.Saves
{
  using System;
  using Level;
  using Player;
  public class SaveModel
  {
    public Header header;

    public PlayerSave PlayerSave;

    public LevelSave LevelSave;
  }

  public class Header
  {
    public DateTime time;

    public int version;
  }
}
```

Файл SaveWorker.cs

```
using _ROOT.Scripts.Saves.Level;

namespace _ROOT.Scripts.Saves
{
    using System.IO;
    using System.Linq;
    using Saves.Player;
    using Newtonsoft.Json;
    using UnityEngine;
    public class SaveWorker
    {
        private const string Path = "save.dat";

        private string FullPath => $"{Application.persistentDataPath}/{Path}";

        public void Save(SaveModel saveModel)
        {
            var obj = JsonConvert.SerializeObject(saveModel);
            File.WriteAllText(FullPath, obj);
        }

        public SaveModel Load()
        {
            var content = File.ReadAllText(FullPath);
            var saveModel = JsonConvert.DeserializeObject<SaveModel>(content);
            return saveModel;
        }

        public bool Prepare()
        {
            if (File.Exists(FullPath))
            {
                return false;
            }
            return true;
        }

        public SaveModel LoadDemo()
        {
            var saveModel = new SaveModel();
            saveModel.PlayerSave = new PlayerSave();
            saveModel.header = new Header();
            saveModel.LevelSave = new LevelSave();

            saveModel.PlayerSave.Gold = 1000;
            saveModel.PlayerSave.Hearts = 5;
            saveModel.PlayerSave.Range = 10;

            return saveModel;
        }
    }
}
```

Файл EnemySettings.cs

```
namespace _ROOT.Scripts.Settings
{
    using System;
    using System.Collections.Generic;
    using UnityEngine;

    [CreateAssetMenu(fileName = nameof(EnemySettings), menuName = "LetsType/EnemySettings")]
    public class EnemySettings : ScriptableObject
```

```
    {
        public List<EnemyStats> stats;
    }

    [Serializable]
    public class EnemyStats
    {
        public string id;

        public int wordsAmount;

        public List<int> wordsLength;

        public float speed;

        public float deltaSpeed;

        public float restoreChance = 0;

        public float allAvailableChance = 0;

        public int mistakeFactor = 2;

        public float speedPerLevel = 0;

        public int wordsAmountPerLevel = 0;

    }
}
```

Файл PlayerDefaultSettings.cs

```
namespace _ROOT.Scripts.Settings
{
    using System;
    using UnityEngine;

    [CreateAssetMenu(fileName = nameof(PlayerDefaultSettings), menuName = "LetsType/PlayerDefautSettings")]
    public class PlayerDefaultSettings : ScriptableObject
    {
        public PlayerDefault stats;
    }

    [Serializable]
    public class PlayerDefault
    {
        public int Gold;

        public int Range;

        public int Hearts;
    }
}
```

Файл SceneController.cs

```
namespace _ROOT.Scripts
{
    using Tools;
    using UnityEngine;
    using UnityEngine.SceneManagement;
    public class SceneController : MonoBehaviour
    {
        [SerializeField] private string menuScene;
```

```
[SerializeField] private string fightScene;

[SerializeField] private string levelScene;

[SerializeField] private int lastSceneWithoutAutoGenerate;

[SerializeField] private string levelGenerateScene;

private int currentLevel;
public void LoadMenu()
{
    LoadScene(menuScene);
}

public void UnloadMenu()
{
    UnloadScene(menuScene);
}

public void SwitchToFightScene()
{
    LoadScene(fightScene);
}

public void UnloadFightScene()
{
    UnloadScene(fightScene);
}

public void SwitchToLevelScene(int level)
{
    string str;
    if (level == -1)
    {
        str = "demo";
    }
    else if (level < 10)
    {
        str = levelScene + "0" + level;
    }
    else
    {
        str = levelScene + level;
    }
    if (lastSceneWithoutAutoGenerate <= level)
    {
        str = levelGenerateScene;
    }
    LoadScene(str);
    currentLevel = level;
}

public void UnloadLevelScene(int level)
{
    string str;
    if (level == -1)
    {
        str = "demo";
    }
    else if (level < 10)
    {
        str = levelScene + "0" + level;
```

```
            }
            else
            {
                str = levelScene + level;
            }

            if (lastSceneWithoutAutoGenerate <= level)
            {
                str = levelGenerateScene;
            }
            UnloadScene(str);
        }

        private void LoadScene(string sceneName)
        {
            SceneManager.LoadScene(sceneName, LoadSceneMode.Additive);
        }

        private void UnloadScene(string sceneName)
        {
            SceneManager.UnloadSceneAsync(sceneName);
        }
    }
}
```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

Комп'ютерна дитяча гра для розвитку друкування та моторики рук
Програма та методика тестування
КПІ.ІП-8202.045480.04.51

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олена ХАЛУС

Нормоконтроль:
_____ Катерина ЛІЩУК

Виконавець:
_____ Олександр БЕБЕХ

Київ – 2023

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є комп'ютерна дитяча гра для розвитку друкування та моторики рук.

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

- перевірка правильності роботи комп’ютерної гри у відповідно до функціональних вимог;
- знаходження проблем та недоліків з метою їх усунення;
- перевірка коректності роботи зіткнення об’єктів;
- перевірка збереження даних об’єктів;
- перевірка діалогової системи;
- перевірка обробки вводу користувача;
- перевірка роботи модуля бойової сцени;
- перевірка роботи модуля збереження прогресу користувача;
- перевірка роботи модуля завантаження налаштувань об’єктів;
- перевірка роботи модуля завантаження та вивантаження сцени;
- перевірка взаємодії між модулями.

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовуються такі методи:

- статичне тестування – перевіряється уся документація, яка аналізується на предмет дотримання стандартів програмування;
- динамічне тестування – застосовується в процесі виконання програми. Коректність програмного засобу перевіряється на безлічі тестів. При прогоні кожного з них збираються та аналізуються дані про проблеми в роботі програми;
- тестування «білої скриньки» – об’єктом тестування тут є внутрішня поведінка програми. Перевіряється коректність побудови всіх елементів програми та правильність їхньої взаємодії один з одним;
- навантажувальне тестування – перевіряється коректна робота програмного забезпечення при великий кількісті об’єктів в ігровій сцені;
- комбінаторнt тестування – аналізує та ретельно перевіряє всі вихідні та вхідні дані комп’ютерної гри, щоб можна було отримати чітке уявлення про різні можливі комбінації та результати. Цей висновок має важливе значення для тестування, оскільки припускає, що тестування комбінацій параметрів може забезпечити високоефективне виявлення несправностей;
- регресійне тестування – перевіряє, чи функції гри працюють оптимально, що допомагає контролювати якість, яка є важливою для досвіду користувача. На практиці відновлюється перевірка гри і порівнюють поточні результати з отриманими в тестах для того щоб перевірити чи не виникли нові помилки через зміни, і чи є старі помилки.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконується вручну, з метою знаходження помилок та недоліків як у функціональній частині програмного забезпечення так і в зручності в користуванні. Для того, щоб перевірити працездатність та відмовостійкість додатку, необхідно провести наступні тестування:

- динамічне тестування на відповідність функціональним вимогам;
- тестування зіткнення об’єктів;
- тестування керування персонажем;
- тестування завантаження ігрових сцен;
- тестування діалогової системи;
- тестування бойової механіки;
- тестування збереження прогресу гравця;
- тестування переходу до наступних рівнів;
- тестування покращення характеристик гравця;
- тестування системи завдань та винагород;
- тестування зручності використання.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

Комп'ютерна дитяча гра для розвитку друкування та моторики рук

Керівництво користувача

КПІ.ІП-8202.045480.05.34

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олена ХАЛУС

Нормоконтроль:
_____ Катерина ЛІЩУК

Виконавець:
_____ Олександр БЕБЕХ

Київ – 2023

ЗМІСТ

1 ЗАГАЛЬНІ ВІДОМОСТІ

«Let’s Type» - це комп'ютерна дитяча гра для розвитку друкування та моторики рук. Комп’ютерна гра призначена для розвитку навичок печаті та моторики рук у ігровій формі, що дозволить користувачу пришвидчити опанування та запам’ятовування клавіш на пристрої воду.

Комп’ютерна гра розділена на два основних стани:

– ігровий рівень, у який гравець може досліджувати, вступати у діалоги, вдосконалювати ігрового персонажа, виконувати прост завдання та переходити на наступні рівні;
– бойова сцена, у якій потрібно задіювати навички печаті, для захисту свого персонажа від ворожої атаки.

Програмне забезпечення дозволяє зберігати прогрес користувача при переході на новий рівень. З переходом на новий рівень складність бойових сцен поступово збільшується, через що гравцю потрібно підвищувати свій рівень друкування та швидше реагувати на події.

2 ПІДГОТОВКА ДО РОБОТИ

2.1 Системні вимоги для коректної роботи

Для успішної роботи даного застосунку необхідне виконання наступних вимог:

- наявність персонального комп’ютера з операційною системою Windows;
- наявність 256 МБ вільної пам’яті;
- наявність графічного процесора Nvidia серії не пізніше G8x.

2.2 Завантаження застосунку

На даний момент застосунок можна встановити власноруч, завантаживши основні компоненти з репозиторію на платформі GitHub.

2.3 Перевірка коректної роботи

По завершенню завантаження програмного забезпечення з платформи GitHub, програмне забезпечення можна запустити.

3 РОБОТА ІЗ ЗАСТОСУНКОМ

Після запуску ігрового застосунку, гравець бачить стартове меню Рисунок 3.1. Користувач з допомогою лівої клавіші комп'ютерної миші може виконувати наступні дії:

– Continue - продовжити гру із збережень гравця;

– New Game - розпочати нову гру;

– Exit - вийти із застосунку.

Рисунок 3.1 – Стартове меню

При відкритті ігрової сцени, користувача зустрічає ігроий рівень, на якому є персонаж підконтрольний гравцю. Приклад щойно завантаженого рівня є на Рисунку 3.2. Після завантаження ігрової сцени, гравець може пересуватись по ігровій локації за допомогою стрілок або клавіш WASD, де:

– W та стрілка вгору – рух вверх;

– A та стрілка вліво – рух вліво;

– S та стрілка вниз – рух ввниз;

– D та стрілка вправо – рух вправо.

Натиснувши та утискаючи клавішу SHIFT користувач може прискорювати свого персонажа.

Рисунок 3.2 – Ігровий рівень

При потряпляння у зону діалогу розпочинаєтсья діалог, який складається із скінченної кількості фраз. Для переходу до настуної фрази потрібно натиснути ліву клавішу комп'ютерної миші. Текст виводиться поступово, для моментального виводу тексту користувач має можливість натиснути натиснути ліву клавішу комп'ютерної миші. Приклад на Рисунку 3.3.

Рисунок 3.3 – Діалогове вікно

У будь-який момент комп’ютерної гри, окрім стартового меню, користувач може зупинити гру та відкрити ігрове меню з допомогою клавіші ESC. Після зупинки гри, на екрані з'являється наступні клавіші, як показано на Рисунку 3.4:

– Continue - продовжити гру;

– Menu – перейти до стартового меню;

– Exit - вийти із застосунку.

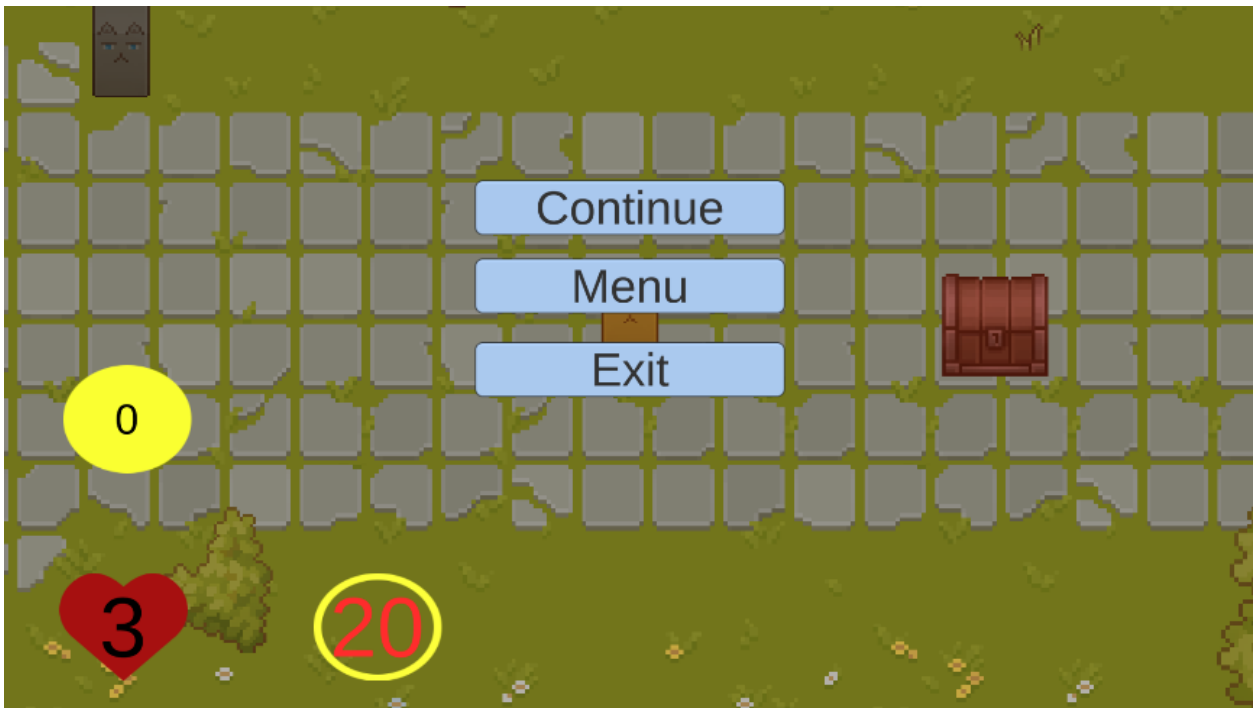


Рисунок 3.4 – Ігрове меню

При зіткненні із ворогом розпочинається бойова сцена. У цій сцені користувач повинен натискати літери, які відображуютья на прямуючих до ігрового персонажа об’єктах, до того як вони з ним зіткнуться. При зіткненні об’єкта-літери із персонажем гравця, об’єкт знищується, а у гравця зменшується кількість очок здоров’я. Кількість очок здоров’я відображається зліва у низу.

При натисканні літери, користувач може помилитись або випадково натиснути неправильний варіант. У такому разі існує механізм покарання, який зменшує кількість доступних помилок за кожну неправильно введену літеру. Лічильник доступних помилок відображається у правій нижчній частині екрану, показаний на Рисунку 3.5. Якщо лічильник досягнув 0 то за кожну помилку при введенні літери ігровий персонаж втрачає очки здоров’я.

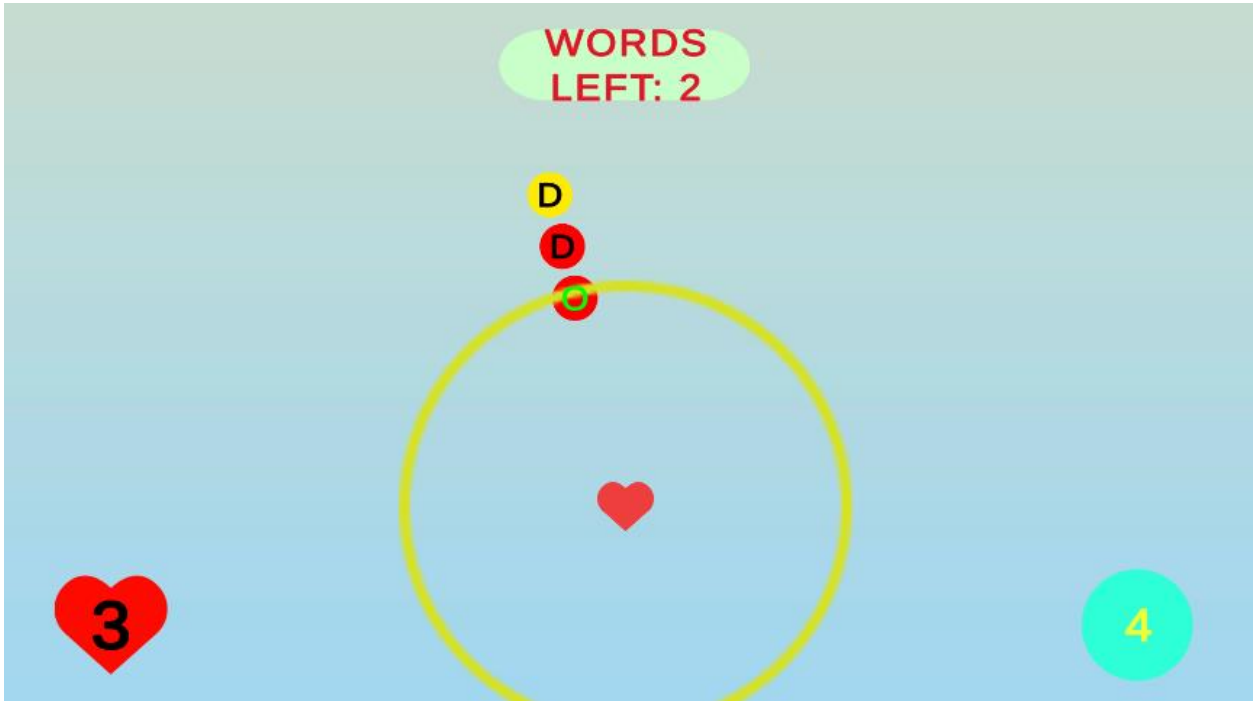


Рисунок 3.5 – Бойова сцена

Програти бойову сцену можливо лише за умови втрати усіх очок здоров’я (нижній правий кут на Рисунку 3.5). Після програшу користувач має змогу знову вступити у бій із противником.

При дослідженні ігрового світу, гравець може зіткнутись із магазином, де він має дві опції, а саме збільшити кількість здоров’я або збільшити радіус атаки, як показано на Рисунку 3.6. Виконати сказані дії гравець може обмежену кількість разів та в обмін на ігрову валюту. Покупка покращення виконується з допомогою натискання лівої клавіші комп'ютерної миші. Кожна наступна покупка коштує більше та кількість покупок вказані та обмежені.

Рисунок 3.6 – Магазин

Гравець може перейти до наступного рівня за певну кількість ігрової валюти, потрапивши у певну зону, яка позначена табличкою, як на Рисунку 3.7. Перехід виконується з допомогою натискання лівої клавіші комп'ютерної миші.

Рисунок 3.7 – Табличка переходу на наступний рівень.

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“___” _____ 2023 р.

Комп'ютерна дитяча гра для розвитку друкування та моторики рук

Графічні матеріали

КПІ.ІП-8202.045480.06.99

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Олена ХАЛУС

Нормоконтроль:

_____ Катерина ЛІЩУК

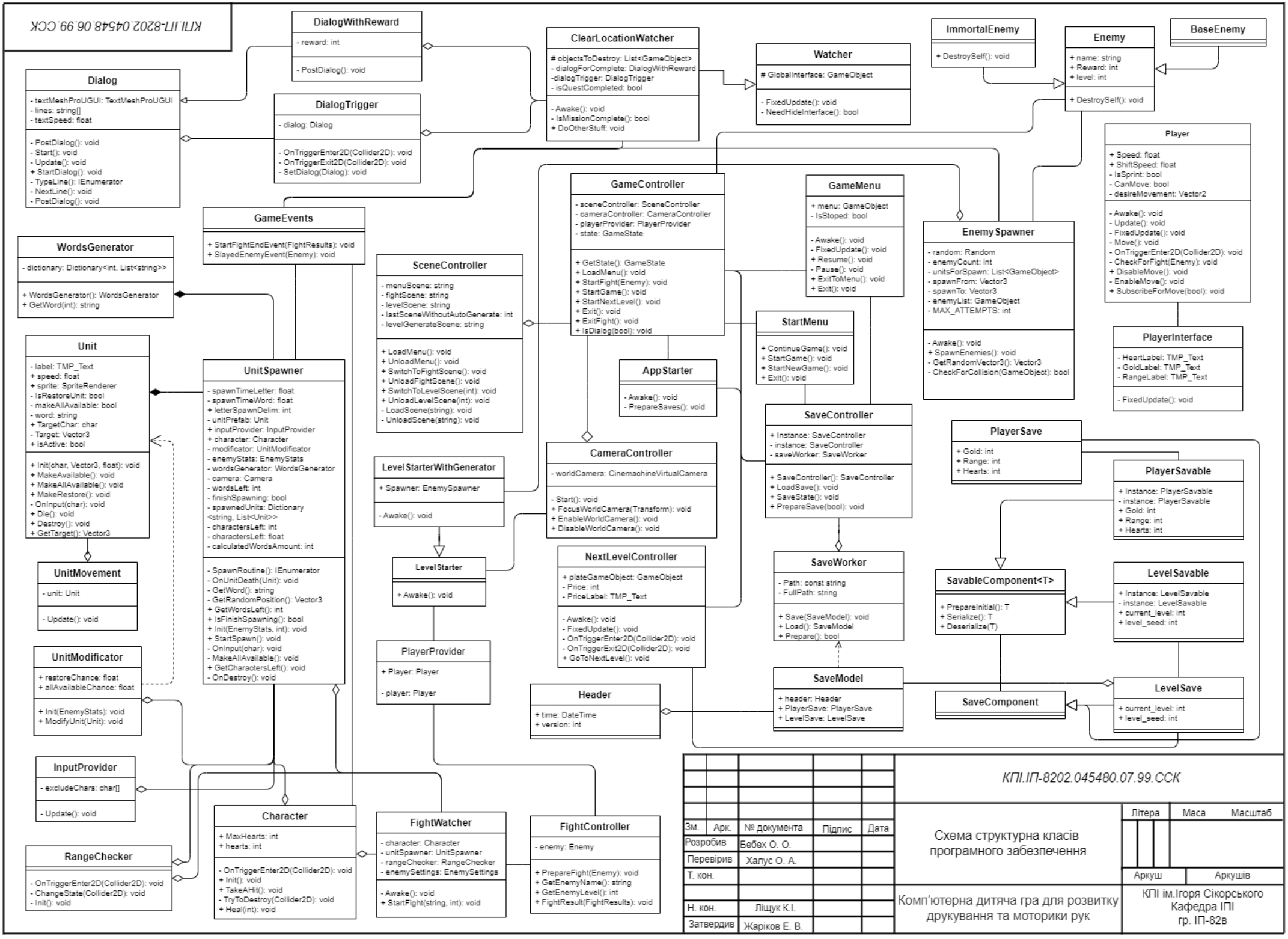
Виконавець:

_____ Олександр БЕБЕХ

Київ – 2023

					КПІ.ІП-8202.045480.06.99.ССВ	
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна варіантів використання	Літера: — Маса: — Масштаб: —
Розробив		Бебех О. О.				Аркуш: — Аркушів: —
Перевірив		Халус О. А.				
Т. кон.						
Н. кон.		Ліщук К.І.			Комп'ютерна дитяча гра для розвитку друкування та моторики рук	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-82в
Затвердив		Жаріков Е. В.				

КПІ.ІП-8202.045480.07.99.ССК



					КПІ.ІП-8202.045480.07.99.ССК
Зм.	Арк.	№ документа	Підпис	Дата	
Розробив		Бебех О. О.			Схема структурна класів програмного забезпечення
Перевірив		Халус О. А.			
Т. кон.					
Н. кон.		Ліщук К.І.			Комп'ютерна дитяча гра для розвитку друкування та моторики рук
Затвердив		Жаріков Е. В.			

Літера	Маса	Масштаб
Аркуш	Аркушів	
КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-82в		

КПІ.ІП-8202.045480.08.99.ССС

					КПІ.ІП-8202.045480.08.99.ССС	
Зм.	Арк.	№ документа	Підпис	Дата		
Розробив		Бебех О. О.			Схема структурна станів системи	Літера / Маса / Масштаб
Перевірив		Халус О. А.				
Т. кон.						Аркуш / Аркушів
Н. кон.		Ліщук К.І.			Комп'ютерна дитяча гра для розвитку друкування та моторики рук	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-82в
Затвердив		Жаріков Е. В.				